Софийски Университет „Св. Климент Охридски“
Факултет по Математика и Информатика

ДИПЛОМНА РАБОТА

на тема

Паралелен алгоритъм за MIC(0) преобуславяне на елиптични задачи

Дипломант: Явор Иванов Вутов, фак. № 42274

Специалност: Информатика
Специализация: Математическо моделиране
Научен ръководител: ст.н.с. II ст. д-р Светозар Маргенов

София, 2002 г.

Съдържание

Увод

Математическите модели на голяма част от физичните процеси са частни диференциални уравнения. Тъй като даже най-простите от тях се решават много трудно аналитично и това в повечето случаи не е възможно, се прибягва към приближеното им решаване с помощта на числени методи.

Основните средства за дискретизация на диференциални уравнения са мрежовите методи – този на крайните елементи (МКЕ) и на крайните разлики. Резултатът от дискретизацията е система линейни уравнения. Главното предимство на тези методи е, че матрицата на получената система уравнения е разредена. Тоест, броят на ненулевите елементи във всеки ред и стълб е ограничен от константа, която не зависи от параметъра на дискретизация.

За получаването на необходимата ни точност на решението, се налага дискретизация при голяма гъстота на мрежите. Това води до системи с брой на неизвестните от порядъка на хиляди, милиони и даже милиарди. Разработени са различни алгоритми за такива задачи.

Намирането на ефективни алгоритми за решаването на големи системи с разредени матрици е много важно. Най-добрият пряк метод е този на вложените сечения. Неговата изчислителна сложност е $O(N^{3/2})$, където N е броят на неизвестните. Разработват се също така и итерационни методи. При тях решението се получава като граница на последователни приближения $x^0, x^1, \ldots$, всяко следващо от които се строи с помощта на предишните (най-често само чрез последното). Най-добрият известен итерационен метод за системи с положително определени и симетрични матрици е този на спрегнатия градиент (conjugated gradient, CG). На всяка итерация при разредена матрица се извършват $O(N)$ операции. Броят на итерациите за достигане на желаната от нас точност е пропорционален на квадратния корен от числото на обусловеност на матрицата

на системата. След дискретизация на задачата

$$-\Delta u \;=\; f \tag{0.1}$$

за числото на обусловеност на матрицата е в сила:

$$\kappa = O(N)\,,$$

където N е броят на неизвестните. Получаваме, че броят на итерациите при прилагане на метода на спрегнатия градиент е $O(N^{1/2})$. Общата изчислителна сложност става $O(N^{3/2})$, което е колкото и най-добрият резултат от преките методи.

Този резултат може да бъде подобрен още чрез преобуславяне. Тоест, вместо решаването на системата $Ax = b$, решаваме системата ${E^{-1}}^T AE{-}1y = E^{-1}b$, $y = Ex$, където E е неособена матрица и числото на обусловеност на матрицата ${E^{-1}}^T AE^{-1}$ е по-малко от числото на обусловеност на A. Така намаляваме броя на итерациите. Този метод се нарича метод на спрегнатия градиент с преобуславяне (preconditioned conjugated gradient, PCG). При него, в една от модификациите му, е необхдимо решаване на системи с матрица $C = E^T E$ на всяка стъпка. Тази матрица се нарича преобусловител. Общата стратегията за конструиране на ефективни преобусловители е да съществува ефективен алгоритъм за решаване на системи с преобусловителя C и числото на обусловеност на $C^{-1}A$ да е много по малко от това на A, тоест:

$$\mathcal{N}\left(C^{-1}x\right) << \mathcal{N}\left(A^{-1}x\right)$$

и

$$\kappa\left(C^{-1}H\right) << \kappa\left(H\right)$$

В тази работа разгледахме преобусловителя от тип $MIC(0)$ (модифицирана непълна факторизация на Холецки). При прилагането на метода на спрегнатия градиент с $MIC(0)$ преобусловител за задачата (0.1) броят на операциите е $O(N^{5/4})$. Въпреки че съществуват и по-добри преобусловители, този се откроява със съотношението между простота при реализацията и изчислителната му сложност.

Даже при добрите алгоритми изчисленията са големи по обем и често единственият начин за тяхното ускоряване са паралелните изчисления. Нашата цел бе да реализираме ефективен паралелен алгоритъм, реализиращ $MIC(0)$ преобуславянето. В същността си решаването на системи с $MIC(0)$ преобусловител е рекурсивно, което затруднява паралелната реализация. Така че ние приложихме метода на крайните елементи върху завъртяна триъгълна мрежа, с цел да получим матрица на коравина със специална структура, която да ни позволи да разпаралелим добре изчисленията.

Паралелният алгоритъм бе реализиран, като тук сме привели резултатите от числените експерименти както по отношение на точността, така и по отношение на скоростта и паралелната ефективност.

1 Преобусловител от тип MIC(0)

Нека A е симетрична и положително определена реална матрица с размери $N \times N$.

$$A = D - L - L^T$$

Тук D е диагоналната част на матрицата A, а $(-L)$ е строго долно триъгълната част на A. Тогава ще търсим MIC(0) преубословител C във вида:

$$C = (X - L)\, X^{-1} \left(X - L^T\right) \tag{1.1}$$

Диагоналната матрица X ще определим от условието:

$$Ae = Ce,\ e^T = (1, 1, \ldots, 1)$$

Тъй като ще използваме матрицата C за преобусловител, се интересуваме от случая, когато $x_{i,i} > 0$. В сила е следната теорема:

Теорема 1.1 *Нека $A = (a_{i,j})$ е симетрична реална матрица с размери $N \times N$ и $A = D - L - L^T$ е разлагане на A и*

$$\begin{aligned} L &\geq 0 \\ Ae &\geq 0 \\ Ae + L^T e &> 0\ ,\ \ e = (1, 1, \ldots, 1)^T \in R^N \end{aligned}$$

Тогава намирането на матрица X, такава че $x_{i,i} > 0$, е възможно и то става по следния начин:

$$x_{i,i} = a_{i,i} - \sum_{k=1}^{i-1} \frac{a_{i,k}}{x_{k,k}} \sum_{j=k+1}^{N} a_{k,j}$$

Доказателството на тази теорема може да бъде намерено в [3].

За задачата

$$-\Delta u \ = \ f \tag{1.2}$$

е в сила следната оценка за числото на обусловеност на матрицата $C^{-1}A$, получена при дискретизацията:

$$\kappa\left(C^{-1}A\right) = O\left(N^{\frac{1}{2}}\right)$$

Както се вижда от (1.1), решаването на системи с матрица C се свежда до решаване на две системи с триъгълни матрици и една с диагонална. Следователно, ако матрицата A е разредена, то за решаването на система с матрицата C трябва се извършват $O(N)$ операции. Трябва да подчертаем, че в общия случай, ако не знаем нищо за структурата на A, решаването на получаващите се триъгълни системи в C е рекурсивно по рода си и не се поддава на ефективно разпаралеляване.

В следващата таблица сме илюстрирали сходимостта на методите на най-бързото спускане (SD), спрегнатия градиент (CG) и спрегнатия градиент с MIC(0) преубословител (PCG), като сме показали броя на итерациите, в зависимост от размера на дискретната задача. Тук размерът на задачата $n \in \{4, 8, 16, 32, 64, 128, 256\}$ съответства на двукратно сгъстяване на мрежата и по двете направления (дискретизацията е извършена върху мрежата представена в следващата глава). Критерият за спиране бе $\|Ax^i - b\|_C \leq 10^{-9}\|Ax^0 - b\|_C$

n	SD	CG	PCG
4	123	6	7
8	517	18	13
16	2119	39	19
32	8585	80	29
64	34577	162	42
128	138811	316	63
256	556273	630	91

Броят на неизвестните в системите е $N = O(n^2)$. И при трите метода броят на операциите на итерация е $O(N)$. Вижда се ясно, че броят

на итерациите при метода на най-бързото спускане е $O(n^2) = O(N)$, при метода на спрегнатия градиент е $O(n) = O\left(N^{1/2}\right)$ и при метода на спрегнатия градиент с $MIC(0)$ преобусловител е $O\left(n^{1/2}\right) = O\left(N^{1/4}\right)$. Превъзходството на метода на спрегнатия градиент с преобуславяне е очевидно.

2 Триангулация върху завъртяна мрежа. Метод на крайните елементи

Да разгледаме двумерната елиптична задача:

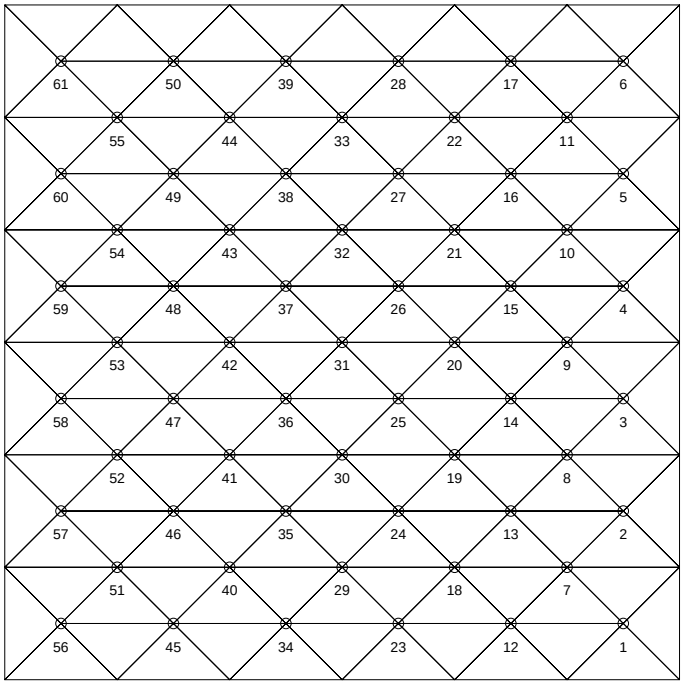
$$\begin{aligned} Lu &\equiv -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + qu = f \qquad (2.1) \\ u|_{\partial\Omega} &= 0 \qquad (2.2) \\ \Omega &= [0,1] \times [0,1] \end{aligned}$$

За дискретизацията ѝ (2.1) приложихме метода на крайните елементи (МКЕ) върху завъртяна мрежа. Използвахме линейни триъгълни елементи. Идеята за тази триангулация дойде от [1].

За построяването ѝ прекарваме линии, успоредни на диагоналите на квадрата, така че те да делят страните на квадрата на части, равни на $h = \frac{1}{n-1}$ (n е параметър за размера на триангулацията). След това всяко от получените квадратчета разполовяваме с хоризонтална линия (виж фиг. 1). Получават се $4n^2$ еднакви равнобедрени правоъгълни триъгълника. Номерираме вътрешните възли, започвайки от 1, по колони, от дясно на ляво. Ред от възли ще наричаме възлите разположени върху една хоризонтална линия, а колона – възлите върху една вертикална линия. Така, общият брой на възлите (и съответно на неизвестните) е $n^2 + (n-1)^2$ (n колони по n възела и $n-1$ колони по $n-1$ възела).

Ще минимизираме функционала на Риц $I\left(v^h\right)$ в крайно мерното пространство S_h, съставено от на части линейни функции v^h:

$$\begin{aligned} I\left(v^h\right) &= \iint_\Omega \left(\left(\frac{\partial v^h}{\partial x}\right)^2 + \left(\frac{\partial v^h}{\partial y}\right)^2 + q{v^h}^2\right) dx\, dy - \\ &- 2\iint_\Omega f v^h \, dx\, dy \end{aligned}$$



Фиг. 1: Мрежата при $n = 6$

$$
\begin{aligned}
S^h \subset H_L &= \left\{ v^h, \frac{\partial v^h}{\partial x}, \frac{\partial v^h}{\partial y} \in L_2\left(\Omega\right), v^h|_{\partial\Omega} = 0 \right\} \\
I\left(v^h\right) &= \sum_{i=1}^{4n^2} \iint_{e_i} \left(\left(\frac{\partial v^h_{e_i}}{\partial x} \right)^2 + \left(\frac{\partial v^h_{e_i}}{\partial y} \right)^2 + q {v^h_{e_i}}^2 \right) dx\, dy - \\
&- \sum_{i=1}^{4n^2} \iint_{e_i} f v^h_{ei}\, dx\, dy \\
v^h_{e_i}(x,y) &= b_0^{e_i} + b_1^{e_i} x + b_2^{e_i} y = P(x,y) b^{e_i} \\
P(x,y) &= \begin{pmatrix} 1 & x & y \end{pmatrix} \\
b^{e_i} &= \begin{pmatrix} b_0^{e_i} & b_1^{e_i} & b_2^{e_i} \end{pmatrix}^T \\
v^h_{e_i}\left(x_1^{e_i}, y_1^{e_i}\right) &= q_1^{e_i} = b_0^{e_i} + b_1^{e_i} x_1^{e_i} + b_2^{e_i} y_1^{e_i} \\
v^h_{e_i}\left(x_2^{e_i}, y_2^{e_i}\right) &= q_2^{e_i} = b_0^{e_i} + b_1^{e_i} x_2^{e_i} + b_2^{e_i} y_2^{e_i} \\
v^h_{e_i}\left(x_3^{e_i}, y_3^{e_i}\right) &= q_3^{e_i} = b_0^{e_i} + b_1^{e_i} x_3^{e_i} + b_3^{e_i} y_3^{e_i}
\end{aligned}
$$

Тук с $\left(x_j^{e_i}, y_j^{e_i}\right)$ и $q_j^{e_i}$ са означени съответно координатите и стойността на приближеното решение във върховете на триъгълния елемент e_i.

$$\begin{aligned}
\begin{pmatrix} q_1^{e_i} \\ q_2^{e_i} \\ q_3^{e_i} \end{pmatrix} &= \underbrace{\begin{pmatrix} 1 & x_1^{e_i} & y_1^{e_i} \\ 1 & x_2^{e_i} & y_2^{e_i} \\ 1 & x_3^{e_i} & y_3^{e_i} \end{pmatrix}}_{H_{e_i}^{-1}} \begin{pmatrix} b_0^{e_i} \\ b_1^{e_i} \\ b_2^{e_i} \end{pmatrix} \\
q^{e_i} &= H_{e_i}^{-1} b^{e_i} \\
b^{e_i} &= H_{e_i} q^{e_i} \\
v_{e_i}^h &= P(x,y) H_{e_i} q^{e_i} \\
E &= \begin{pmatrix} \frac{\partial v_{e_i}^h}{\partial x} \\ \frac{\partial v_{e_i}^h}{\partial y} \end{pmatrix} = \begin{pmatrix} \frac{\partial P}{\partial x} \\ \frac{\partial P}{\partial y} \end{pmatrix} H_{e_i} q^{e_i} = \\
&= \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{B} H_{e_i} q^{e_i} = B H_{e_i} q^{e_i} \\
\left(\frac{\partial v_{e_i}^h}{\partial x}\right)^2 + \left(\frac{\partial v_{e_i}^h}{\partial y}\right)^2 &= E^T E = q^{e_i T} H_{e_i}^T B^T B H_{e_i} q^{e_i} \\
{v_{e_i}^h}^2 &= q^{e_i T} H_{e_i}^T P^T P H_{e_i} q^{e_i}
\end{aligned}$$

Тогава, като заместим, получаваме:

$$\begin{aligned}
I\left(v^h\right) &= \sum_{i=1}^{4n^2} \Bigg(q^{e_i T} \underbrace{H_{e_i}^T \iint_{e_i} B^T B \, dx \, dy H_{e_i}}_{K_{e_i}^1} q^{e_i} + \\
&+ q^{e_i T} \underbrace{H_{e_i}^T \iint_{e_i} q(x,y) P^T P \, dx \, dy H_{e_i}}_{K_{e_i}^0} q^{e_i} \Bigg) - \\
&- 2 \sum_{i=1}^{4n^2} \underbrace{\iint_{e_i} f(x,y) P \, dx \, dy H_{e_i}}_{F_{e_i}^T} q^{e_i} = \\
&= \sum_{i=1}^{4n^2} q^{e_i T} \left(K_{e_i}^1 + K_{e_i}^0\right) q^{e_i} - 2 \sum_{i=1}^{4n^2} q^{e_i T} F_{e_i}
\end{aligned}$$

$$I\left(v^h\right) = Q^T K Q - 2F^T Q \tag{2.3}$$

Интегралите пресмятаме приближено, прилагайки повторни квадратурни формули на Гаус.

Като минимизираме (2.3) по метода на Риц, получаваме системата:

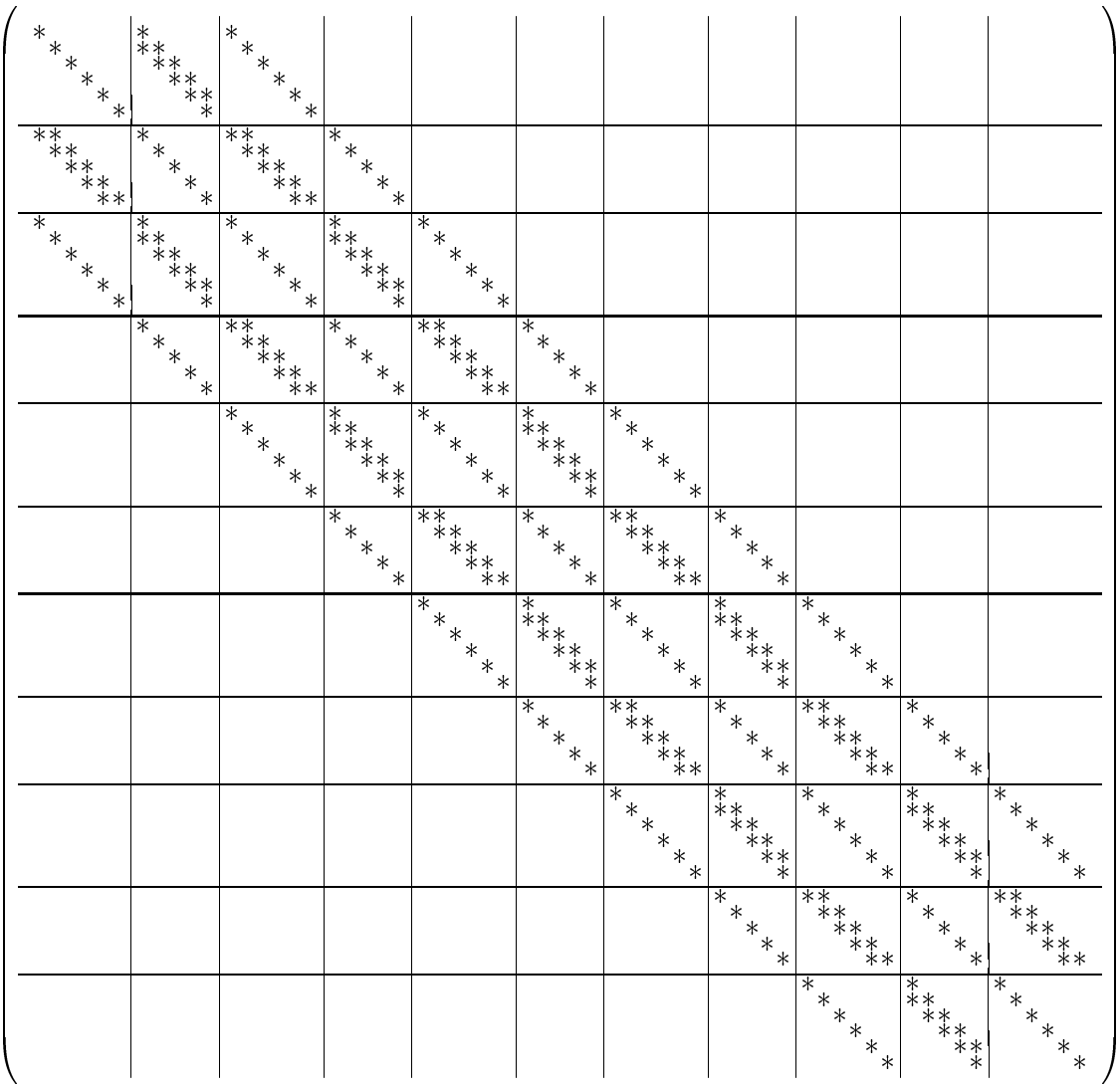
$$KQ = F$$

Сглобената матрица на коравина K има ненулеви елементи по главния диагонал, както и по диагоналите $n-1$, n и $2n-1$ над и под него. Тя има следната блочна структура:

$$K = \begin{pmatrix} K_{1,1} & K_{1,2} & K_{1,3} & & & & \dots & & & \\ K_{2,1} & K_{2,2} & K_{2,3} & K_{2,4} & & & \dots & & & \\ K_{3,1} & K_{3,2} & K_{3,3} & K_{3,4} & K_{3,5} & & \dots & & & \\ & K_{4,2} & K_{4,3} & K_{4,4} & K_{4,5} & K_{4,6} & \dots & & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots \\ & & & & & & \dots & K_{2n-2,2n-2} & K_{2n-1,2n-1} & \\ & & & & & & \dots & K_{2n-1,2n-2} & K_{2n-1,2n-1} & \end{pmatrix}$$

Тук блоковете $K_{i,j}$ са с размери както следва: $K_{2k,2k}(n-1 \times n-1)$, $K_{2k,2k+2}(n-1 \times n-1)$; $K_{2k+2,2k}(n-1 \times n-1)$; $K_{2k+1,2k+1}(n \times n)$; $K_{2k+1,2k+3}(n \times n)$; $K_{2k+3,2k+1}(n \times n)$; $K_{2k,2k+1}(n-1 \times n)$; $K_{2k+1,2k}(n-1 \times n)$; $K_{2k+1,2k+2}(n \times n-1)$; $K_{2k+2,2k+1}(n \times n-1)$. Матриците $K_{i,j}$, $K_{i,i-2}$ и $K_{i,i+2}$ са диагонални. Матриците $K_{i,i+1}$ и $K_{i+1,i}$ са двудиагонални.

На фиг. 2 е показана матрицата K за $n = 6$. Ненулевите елементи са означени с „*“. За моделната задача – когато $q \equiv 0$ – матриците $K_{i,i-2}$ и $K_{i,i+2}$ са нулеви. Ненулевите елементи по диагонала са равни на 4, а тези извън него – на (-1). Следователно са изпълнени условията от теорема 1.1, за съществуването на стабилна MIC(0) факторизация.



Фиг. 2: Така изглежда матрицата K за $n = 6$

Структурата на триъгълните матрици в MIC(0) факторизацията на K е същата – с диагонални блокове по диагонала. Това бе и основната цел при използването на тази триангулация.

За да проверим точността на метода, проведохме числени експери-

менти при:

$$\begin{aligned} q(x,y) &= 2 \\ f(x,y) &= 2\left(x(1-x)+y(1-y)+xy(1-x)(1-y)\right) \end{aligned}$$

и за различни стойности на n. Ето и резултатите:

n	$\|R_n\|_C$	$\|R_n\|_{K_n}$
4	$0,00192474$	$0,0057688$
8	$0,000511352$	$0,00284628$
16	$0,000132256$	$0,00141618$
32	$3,36684.10^{-5}$	$0,000707091$
64	$8,49629.10^{-6}$	$0,000353414$
128	$2,13421.10^{-6}$	$0,00017669$
256	$5,34838.10^{-7}$	$8,83428.10^{-5}$
512	$1,33872.10^{-7}$	$4,41711.10^{-5}$
1024	$3,34887.10^{-8}$	$2,20855.10^{-5}$
2048	$8,37169.10^{-9}$	$1,10428.10^{-5}$

Тук $R_n = Q_n - U_n$, където U_n е точното решение $u(x,y) = x(1-x)y(1-y)$ в точките от мрежата, а Q_n е приближеното решение.

Вижда се, че $\|R_n\|_C$ намалява както $O(h^2)$, а $\|R_n\|_{K_n}$ намалява както $O(h)$, което и очаквахме от теорията на МКЕ.

3 Анализ на изчислителната сложност на MIC(0) PCG

Алгоритъмът на спрегнатия градиент с преобусловител за системата $Ax = b$ изглежда по следния начин:

$$\begin{aligned}
x^0 &= 0,\ g^0 = Ax^0 - b,\ h^0 = C^{-1}g^0,\ d^0 = -h^0,\ \gamma_0 = \left(g^0, h^0\right) \\
k &= 0,\ 1,\ 2, \ldots \\
t^k &= Ad^k \\
\tau_k &= \frac{\gamma_k}{(d^k, t^k)} \\
x^{k+1} &= x^k + \tau_k d^k \\
g^{k+1} &= g^k + \tau_k t^k \\
h^{k+1} &= C^{-1} g^{k+1} \\
\gamma_{k+1} &= \left(g^{k+1}, h^{k+1}\right) \\
\beta_k &= \frac{\gamma_{k+1}}{\gamma_k} \\
d^{k+1} &= \beta_k d^k - h^{k+1}
\end{aligned}$$

На всяка итерация се извършва едно умножение на матрицата A с вектор, две скаларни произведения на вектори, три операции от тип умножение на скалар с вектор плюс вектор, едно решаване на система с преобусловяващата матрица C и две деления. Тоест:

$$\mathcal{N}_{it}^{PCG-MIC(0)}\left(A^{-1}b\right) = \mathcal{N}\left(Ax\right) + 2\mathcal{N}\left((a,b)\right) + 3\mathcal{N}\left(\alpha a + b\right) + \mathcal{N}\left(C^{-1}x\right) + 2$$

За произведението Ax имаме: $N = n^2 + (n-1)^2$ операции за произведението на главния диагонал с вектора x – $(N - (2n-1))2.2$ за произведенията на диагоналите $2n-1$ и $-2n+1$ с x и прибавянето им към резултата и още $(2n-2).2.(n-1).2.2$ операции ($2(2n-2)$ матрици $K_{i,i+1}$ и

$K_{i,i=1}$ по $2*(n-1)$ елемента и за всеки елемент 2 операции – умножение и прибавяне). Накрая получаваме:

$$\mathcal{N}(Ax) = 5(n^2+(n-1)^2)-4(2n-1)+2.2.2(2n-2)(n-1) = 26n^2-50n+25$$

В термините на N получаваме:

$$\mathcal{N}(Ax) \approx 13N$$

При решаването на системата с преобусловителя се решават две системи с триъгълни матрици и една с диагонална. За решаването на една триъгълна система с горнотриъгълна част от матрицата A са необходими $(2n-2)(n-1)2.2 + (N-(2n-1)).2) + N$ операции. Като цяло получаваме:

$$\mathcal{N}\left(C^{-1}x\right) = 2(3N + 2.2(2n-2)(n-1) - 2(2n-1)) + N = 30n^2 - 54n + 27$$

или

$$\mathcal{N}\left(C^{-1}x\right) \approx 15N$$

Вижда се, че броят на операциите при решаването на системата с преобусловителя е приблизително равен на техния брой при умножението на матрица по вектор. За скаларното произведение имаме $2N-1 = 4n^2-4n+1$ операции, за операцията скалар по вектор плюс вектор – $2N = 4n^2-4n+2$.

В крайна сметка получаваме:

$$\mathcal{N}_{it}^{PCG-MIC(0)}\left(A^{-1}b\right) = 76n^2 - 124n + 62$$

$$\mathcal{N}_{it}^{PCG-MIC(0)}\left(A^{-1}b\right) \approx 38N$$

4 Паралелна реализация

При съставянето на паралелния алгоритъм най-важният въпрос е как да бъдат разпределени данните между процесорите.

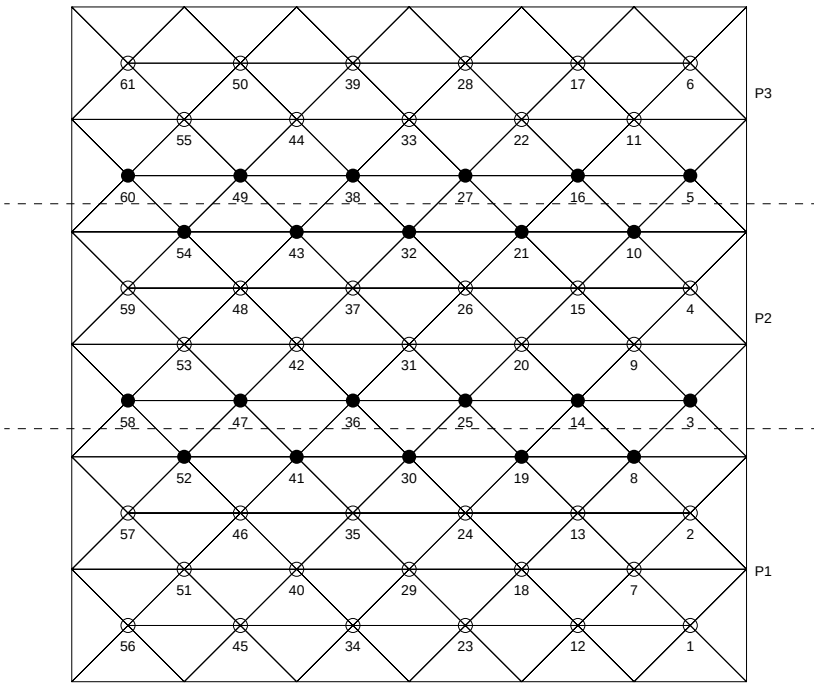
Както видяхме в глава 2, структурата на триъгълните матрици в преобусловителя е с диагонални блокове по диагонала. Решението на система с такава матрица може да се раздели на $2n - 1$ етапа. Като на всеки етап се намират неизвестните, съответстващи на всеки диагонален блок. Всяко неизвестно зависи от неизвестните в другите блокове, но не и от тези в своя, така че те могат да бъдат пресмятани независимо едно от друго. Това ни навежда на мисълта за разделянето на всеки блок между всички процесори. Всъщност се получава така, че всеки процесор трябва да получи по един или няколко последователни реда от неизвестни от мрежата (виж фиг. 3). Максималният брой натоварени по този начин процесори е колкото и броят на редовете – $2n - 1$.

Следващият проблем, който трябва да разрешим, е как точно да бъдат представени частите от векторите в паметта. Специалната структура на мрежата, този път ни създава известни трудности – във всеки ред, както и във всяка колона, броят на неизвестните е различен. Факторите, които трябва да вземем предвид, са:

- *Ефективно разпределяне на паметта* – трябва да заемаме възможно най-малко памет (тоест не трябва да имаме много неизползвани елементи);

- *Ефективно разпределение на неизвестните* – трябва да можем да ги обхождаме възможно най-бързо. Също така, трябва да можем да намираме бързо съседите на даден елемент;

- *Място за елементи, „принадлежащи“ на други процесори.* Част от операциите (умножаване на матрица по вектор, както и решаване



Фиг. 3: Разделяне на областта между три процесора

на триъгълна система уравнения) изискват данни и от съседните процесори;

- *Лесно определяне на мястото на елемента от номера му и обратното.*

За обхождането на елементите в даден процесор използваме специален показалец (итератор). Той всъщност е указател към даден елемент. Само че за разлика от обикновения указател, той може да изпълнява малко повече функции, освен стандартната – да осигурява достъп до елемента, към който сочи. Те са:

- Позициониране в първия елемент в процесора;
- Преместване към елемента в ляво и в дясно на същия ред;

- Преместване към елемента отгоре на същата колона;
- Проверка дали не е излязъл извън границите (в ляво, в дясно, или отгоре);
- Преместване в началото на следващия ред;
- Преместване в началото на следващата колона;
- Достъп до съседните елементи, без промяна на позицията.

Този показалец, осен че ни позволява да реализираме бърз, също както и логичен, достъп до елементите в даден процесор, ни позволява и да използваме различни физически подредби на елементите. Ето например как се реализира функция, която прибавя един вектор към друг:

```
// X = X + Y
template<typename Vector>
void AddTo(Vector&X, const Vector &Y) // X и Y са вектори.
{
    typename Vector::iter Xit = X.GetIter();
    typename Vector::const_iter Yit = Y.GetIter();
    // Xit и Yit са показалци: единият към елементи от вектора X,
    // другият - към елементи от вектора Y

    // ++Xit премества показалеца към
    // елемента в ляво на същия ред.

    for(;Xit.NotEnd(); Xit.NextLine(),Yit.NextLine())
        for(;Xit.NotEndOfLine(); ++Xit, ++Yit)
            *Xit += *Yit;
}
```

Ние използвахме две различни представяния на векторите (на техните части в процесорите). Първото е плътно подреждане на елементите

по редове (`LineVector`). Второто представяне е също по редове, само че се пазят по два допълнителни реда отгоре и отдолу, за да може в тях да се прехвърлят елементи от съседните процесори, както и по две допълнителни колони в ляво и в дясно, за да се избегнат проверките в граничните случаи (излишните елементи – редът отдолу в първия процесор, редът отгоре в последния процесор, както и допълнителните колони – се нулират)(`BoundedLineVector`).

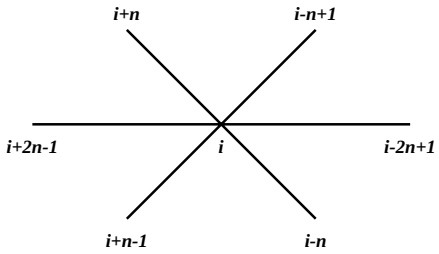
В алгоритъма участва умножение на матрица по вектор. Тъй като тя е разредена, я представяме като вектор от ненулевите елементи във всеки ред. По-точно за всеки ред се пазят елементите от седемте диагонала: $-2n+1$, $-n$, $-n+1$, 0, $n-1$, n, $2n-1$. Използваме повече памет отколкото ни е необходима (като се вземе предвид, че матрицата K е симетрична), но това ни позволява да умножаваме бързо матрицата по вектор. Физическото представяне е като `LineVector` от масиви по седем елемента. Това ни позволява да обхождаме редовете на матрицата по същия начин, както обхождаме елементите на векторите.

След като сме си изяснили представянето на данните, остава да реализираме и необходимите ни операции.

За операциите от типа умножаване на вектор със скалар събиране с друг вектор не ни трябва нищо друго, овен да извършим операцията поелементно във всеки процесор (става както в примера с прибавянето на вектор).

За скаларното произведение, след като сумираме произведенията на съответните елементи във всеки процесор, се налага да се сумират и сумите от всеки процесор.

Умножаването на матрицата с вектор е малко по-сложно. Тъй като ни трябват данни от съседните процесори, се налага да се извърши прехвърлянето им преди самото умножение. На фиг. 3 елементите от възлите със запълнени кръгчета на мрежата трябва да се прехвърлят в съседния процесор. След като е извършено прехвърлянето, се извършва самото умножение. (Умножението може да бъде извършено за елемен-



Фиг. 4: Възлите, които съответстват на ненулевите елементи от i-тия ред на матрицата

тите от вътрешните линии за процесора – тези, които на фигурата са с незапълнени кръгчета – преди да бъдат получени елементите от съседния процесор). Използваните елменти от вектора, по който умножаваме, са тези от съседните върхове на върха i (виж фиг. 4).

Остава да се реши паралелно и системата с преобуславящата матрица. Решението е всъщност решаване на три системи – едната с долнотриъгълна матрица, другата с горнотриъгълна, а третата с диагонална. Решаването на системата с диагонална матрица е тривиално.

Да разгледаме системата с долнотриъгълна матрица. Поради специалната ѝ структура, пресмятането на всяка колона от елементи може да бъде извършено паралелно. Започвайки от първата колона (от дясно на ляво),пресмятаме стойностите на елементите от нея, след това извършваме комуникации, за да прехвърлим необходимите за пресмятане на следващата колона елементи (отново трябва да бъдат прехвърлени елементите от върховете със запълнени кръгчета от фиг. 3), след това преминаваме към следващата колона.

Решаването на система с горнотриъгълна матрица е аналогично с решаването на система с долнотриъгълна.

Сега да се спрем по-подробно на комуникациите. За умножението на матрицата K по вектор се прехвърлят по n елемента към всеки от съседните процесори. За решаването на една триъгълна система от преобусловителя трябва n пъти да се прехвърли по един елемент към съседните

процесори. По-точно за всяка колона един процесор, или изпраща, или получава към всеки един от съседите си. При това всички тези комуникации между различните процесори могат да бъдат извършени едновременно. За скаларното произведение е нужна една операция за събиране (gather) и една операция за разпръскване (broadcast) на всяка итерация. Тъй като те не зависят от n, тях ще ги пренебрегнем в по-нататъшните разглеждания.

Времето за прехвърлянето на N елемента между два съседни процесора (такива са всичките комуникации, които използваме, освен тези при скаларното произведение) можем да разглеждаме като:

$$T^{comm} = \tau + bN$$

Тук τ е така нареченият *startup time*. Това е времето необходимо за започване на прехвърлянето, b е времето за прехвърляне на един елемент.

Така за времето за комуникации на итерация получаваме:

$$T_{it}^{comm} = 2(\tau + bn) + 2(2n-1)(\tau + b.1) = n(6b + 4\tau) - 2b$$

Както видяхме в глава 3, броят на аритметическите операции на итерация е:

$$\mathcal{N}_{it}^{PCG-MIC(0)}\left(A^{-1}b\right) \approx 38N$$

Броят на операиците на итерация във всеки процесор приблизително е равен на:

$$\mathcal{N}_{it/proc}^{PCG-MIC(0)}\left(A^{-1}b\right) \approx \frac{38N}{k}\ ,$$

където k е броят на процесорите. Ако M операции се извършват за Mt време, то за времето за изчисления е в сила:

$$T_{it}^{comp} \approx \frac{38Nt}{k} + n(6b + 4\tau) - 2b$$

За ускорението и ефективността се получава, че:

$$\begin{aligned} \lim_{n\to\infty} \mathcal{S}(n,k) &= k \\ \lim_{n\to\infty} \mathcal{E}(n,k) &= 1 \end{aligned}$$

За по-реалистична оценка трябва да знаем малко повече за параметрите τ, b и t. На практика $\tau >> b$ и $\tau >> t$ и за получаването на добро ускорение и добра ефективност трябва:

$$n >> \frac{k\tau}{t}.$$

5 MPI реализация

Паралелният алгоритъм бе реализиран посредством библиотеката MPI (Message Passing Interface). MPI е представител на парадигмата за програмиране чрез предаване на съобщения (message passing). Тази парадигма се използва много широко в разработването на паралелни програми, особено при системи с разпределена памет. Преди появата на MPI, всеки производител на суперкомпютри предоставяше собствена библиотека за работа с тях. Това, разбира се, пречеше на преносимостта на програмите. Целта на създателите на MPI бе да се създаде единен интерфейс за паралелно програмиране. MPI е взел положителните черти на съществуващите преди това библиотеки (PVM, PICL, Vertex и др.). Чрез MPI се постига една много добра преносимост на програмите. Също така, MPI поддържа хетерогенни системи, тоест върху системи с процесори с различна архитектура. Постигайки добра преносимост на програмите, MPI позволява и създаването на много ефективни реализации, което е и най-голямото му преимущество и причина за все по-нарастващата му популярност.

За реализиране на паралелния алгоритъм, ние използвахме една не голяма част от възможностите на MPI. Това са функциите `MPI_Allreduce`, `MPI_Send_init`, `MPI_Recv_init`, `MPI_Startall` и `MPI_Waitall`. Основни функции, които използвахме са `MPI_Comm_rank` и `MPI_Comm_size`. Следва кратко описание на тяхното действие:

`MPI_Comm_size` – Тази функция връща броя на процесорите, които ще изпълняват програмата. Този брой ни е необходим, за да разберем на колко части да разделим данните;

`MPI_Comm_rank` – Връща номера на процесора. Всички функции за приемане или изпращане на съобщения приемат и параметър номер на процесор, на който или от който се праща. Освен това, с помощта на този номер всеки процесор разбира коя част от данните трябва да обработва;

С помощта на горните две функции всеки процесор по време на изпълнение на програмата „разбира“ колко са всичките процесори, както и собствения си номер. Процесорите са номерирани с последователни номера, започвайки от нула до броя на процесорите минус един. Това е важно, защото всички функции за предаване на съобщения приемат като параметър номера на получателя или изпращача. Всеки процесор в нашия алгоритъм комуникира само със съседите си. Съседи нарича-ме процесори с последователни номера. Първият и последният процесор осъществяват комуникации само с по един друг процесор (ако се различават), а останалите – с по два.

`MPI_Allreduce` – В нашата програма се използва от скаларното произведние, за да събере частичните суми от всички процесори;

`MPI_Send_init` и `MPI_Recv_init` – Създават заявки за изпращане и получаване на данни. По-късно зая-вките могат да бъдат изпълнявани многократно;

`MPI_Startall` – Пуска една или няколко заявки за пращане или получаване за изпълнение. Тази функция се връща веднага, позволявайки да се извършват изчисления, докато се извършва комуникацията. Например, когато умножаваме матрица по вектор, след като пуснем заявките за приемане и изпращане на данните, ние можем, докато траят комуникациите, да извършим пресмятанията за тези елементи, които не зависят от получаваните;

`MPI_Waitall` – Изчаква приключването на изпълнението на заявките. След като се върне, ние вече сме получили или изпратили данните;

В нашия случай е мнго удобно използването на команди за приготвяне на заявките и такива за тяхното изпълнение, защото ние през цялото време изпращаме и получаваме данните от и на едни и същи места. Заявките яснотата на програмата, а също така позволяват на MPI реализацията да направи допълнителни оптимизации, за по-бързото им изпълнение.

6 Числени експерименти

Проведохме числени експерименти за изпълнението на паралелния алгоритъм. Експериментите бяха проведени на клъстер, съставен от четири двупроцесорни комютъра Power Macintosh, всеки от които със 512 MB RAM и два процесора G4 на 450 Mhz. Комуникациите между отделните машини се извършва по локална мрежа, а между процесорите в един комютър – посредством обща памет. Използвахме LAM-MPI реализацията на MPI [6].

Задачата, която решихме е:

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + 2u = 2(x(1-x) + y(1-y)) + 2xy(1-x)(1-y)$$

в $[0,1] \times [0,1]$ с гранични условия на Дирихле.

Следват получените резултати. В първата таблица са показани времената за решаване на получената система уравнения на един процесор, а в другите две – само ускоренията и ефективността.

Времена и брой итерации

n	32	64	128	256	512	1024	1500
N	1985	8065	32513	130561	523265	2095105	4497001
брой итерации	30	43	62	89	126	179	214
врме в секунди	0.10	0.58	4.45	30.7	206	1374	3649

Ускорения

n	32	64	128	256	512	1024	1500
$\mathcal{S}(n,2)$	1.21	1.68	1.96	1.85	1.92	2.03	2.02
$\mathcal{S}(n,3)$	0.24	0.46	0.97	1.72	2.45	2.97	2.86
$\mathcal{S}(n,4)$	0.22	0.46	1.11	1.97	2.88	3.76	3.95
$\mathcal{S}(n,5)$	0.20	0.40	0.96	1.99	3.25	4.48	4.86
$\mathcal{S}(n,6)$	0.18	0.39	1.03	1.99	3.55	5.23	5.73
$\mathcal{S}(n,7)$	0.19	0.38	0.95	1.78	3.63	6.02	6.31
$\mathcal{S}(n,8)$	0.19	0.39	1.00	2.28	3.97	6.37	6.76

Ефективност

n	32	64	128	256	512	1024	1500
$\mathcal{E}(n,2)$	0.60	0.84	0.98	0.93	0.96	1.02	1.01
$\mathcal{E}(n,3)$	0.08	0.15	0.32	0.57	0.81	0.99	0.95
$\mathcal{E}(n,4)$	0.06	0.12	0.27	0.49	0.72	0.94	0.98
$\mathcal{E}(n,5)$	0.04	0.08	0.19	0.40	0.65	0.90	0.97
$\mathcal{E}(n,6)$	0.03	0.07	0.17	0.33	0.59	0.87	0.96
$\mathcal{E}(n,7)$	0.03	0.05	0.14	0.25	0.51	0.86	0.90
$\mathcal{E}(n,8)$	0.02	0.05	0.13	0.29	0.50	0.80	0.84

Вижда се, че при фиксиран брой процесори и при увеличаване на размера на задачата, ускорението расте към броя на процесорите, а ефективността клони към едно.

Поради драстичната разлика между *startup time*-а и времето за операция, както и очаквахме, ускорението и ефективността при малки стойности на n са много малки.

Заради физическата организация на паметта – достъпът до нея е организиран на няколко нива с различно бързодействие – се получава, че достъпът до по-малка по обем памет е по-бърз. При паралелния алгоритъм, разделяйки изчисленията между процесорите, ние също така разделяме и данните, като съответно използваме по-малко памет във всеки един процесор. Това е една от причините за по-голямата скорост на изпълнение на програмата на повече от един процесор, както е и причината за ефективностите по-големи от 1. Този ефект е и причината за по-голямото увеличение на времето за изпълнение, в сравнение с броя на операциите.

В следващата таблица са показани времената за изчисления и за комуникации на итерация. Трябва да отбележим обаче, че при програмната реализация, комуникациите се застъпват с изчисленията във времто. Реалното им времетраене е по-голямо. Времето показано тук е времто за изчакване до приключване на комуникациите – когато получените (или изпратените) данни ни трябват за по-нататъшния ход на изчисленията.

n	64	128	256	512	1024	1500
$T_{it}^{comp}(n,1)$	0.139	0.0771	0.3821	1.7985	8.4497	16.983
$T_{it}^{comp}(n,2)$	0.0064	0.03538	0.1967	0.8803	3.8242	8.2251
$T_{it}^{comp}(n,4)$	0.0038	0.01457	0.0867	0.4202	1.7834	3.6846
$T_{it}^{comp}(n,8)$	0.0025	0.0076	0.0378	0.1907	0.8397	1.7712
$T_{it}^{comm}(n,2)$	0.0032	0.0063	0.0160	0.0401	0.1052	0.2251
$T_{it}^{comm}(n,4)$	0.0301	0.0590	0.1166	0.2236	0.3831	0.6361
$T_{it}^{comm}(n,8)$	0.038	0.0744	0.1343	0.2673	0.4972	0.7486

Наблюдава се, че когато n е малко, времето за комуникации е много по-голямо от времето за изчисленията – затова и се получава лошата ефективност. Обаче с нарастването на n, както и очаквахме, времето за комуникации расте по-бавно от времето за изчисленията и съответно се получава по-добра ефективност. Също така, много добре се вижда, че времето за комуникации в случая с два процесора, е много по-малко от времето при повече процесори. Това е така заради структурата на клъстера.

Много добре се вижда, че времето за изчисленията намалява обратно пропорционално на броя на процесорите. Тъй като комуникациите се застъпват с изчисленията във времето, а времето за тях намалява, се наблюдава леко увеличение на времето за изчакване между тестовете с четири и тези с осем процесора.

Заключение

В тази работа разгледахме непълната MIC(0) факторизация, като метод за преобуславяне на двумерна елиптична задача. Видяхме, че това е един добър метод, прилагането на който дава отлични резултати. Въпреки рекурсивната му същност, посредством завъртяна мрежа и прилагайки МКЕ върху нея, постигнахме специална структура на матрицата на коравина, съставена от диагонални блокове по диагонала.

Тази структура ни позволи да разпаралелизираме изчисленията във всеки такъв диагонален блок, съставяйки по този начин един много добър паралелен алгоритъм.

Получените резултати, при направените от нас тестове, напълно потвърдиха нашите очаквания, както относто точността на метода, така и относно неговата ефективност.

Разбира се, ефективността на алгоритъма може още да се подобрява. Ако например сгъстяваме мрежата само във вертикално направление, ще постигнем увеличение на точността на решението, без никакво нараставане на размера на комуникациите, което ще промени съотношението изчисления - комуникации, което е всъщност и причината за не особено голямата ефективност на алгоритъма при малки размери на дискретната задача.

Използвана литература

[1] On Parallel Solution of Linear Elasticity Problems, Part II: Methods and Some Computer Experiments. I. Gustafsson and G. Lindskog. *Numerical Linear Algebra with Applications 2000.*

[2] MPI: The Complete Refference. Marc Sair, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongara *The MIT Press Cambridge Massachusetts, London England 1996.*

[3] Displacement Decompozition - Incomplete Factorization Preconditioning Techniques for Linear Elasticity problems, R. Blaheta, *Numerical Linear Algebra with Applications 1994.*

[4] Числени Методи, Благовест Сендов, Васил Попов, *Университетско Издателство „Св. Климент Охридски“ 1996.*

[5] Теория Метода Конечных Элементов, Г. Стренг, Дж. Фикс, *Издательство МИР, Москва 1977.*

[6] Local Area Multicomputer. *(http://www.lam-mpi.org).*