УДК 519.63

MPI+OpenMP РЕАЛИЗАЦИЯ МЕТОДА СОПРЯЖЕННЫХ ГРАДИЕНТОВ С ФАКТОРИЗОВАННЫМИ ЯВНЫМИ ПРЕДОБУСЛОВЛИВАТЕЛЯМИ

И. Е. Капорин, О. Ю. Милюкова
(ВЦ им. А. А. Дородницына РАН, ИПМ им. М. В. Келдыша РАН, г. Москва)

Для предобусловливания симметричной положительно определенной разреженной матрицы рассматриваются ее приближенные обратные матрицы, представленные в виде произведения двух взаимно сопряженных разреженных треугольных матриц. Предложен способ параллельной реализации метода сопряженных градиентов с факторизованными явными предобусловливателями с использованием (MPI+OpenMP)-подхода. Проводится сравнение времени решения тестовых задач из коллекции университета Флориды с использованием MPI- и (MPI+OpenMP)-подходов методом сопряженных градиентов с рассматриваемыми предобусловливателями и с предобусловливанием Якоби, а также предобусловливанием при помощи блочного неполного обратного треугольного разложения второго порядка.

Ключевые слова: разреженные матрицы, метод сопряженных градиентов, явное предобусловливание, неполная обратная треугольная факторизация, параллельные вычисления.

Введение

Рассмотрим задачу приближенного решения системы линейных алгебраических уравнений (СЛАУ) большого размера

$$A\vec{x} = \vec{b} \tag{1}$$

с симметричной положительно определенной разреженной матрицей общего вида $A = A^\top > 0$.

Проблема построения соответствующих эффективных численных методов решения сохраняет свою актуальность, так как во многих важных прикладных областях продолжают возникать новые постановки таких задач. При этом наблюдается тенденция к росту размера матриц n, увеличению их заполненности ненулевыми элементами, усложнению структуры разреженности, а также к ухудшению обусловленности. Примерами такой постановки являются конечно-элементные модели пространственных задач вычислительной механики, ставшие источником использованных в настоящей работе тестовых матриц [1].

В настоящей работе для решения СЛАУ большого размера применяется предобусловленный метод сопряженных градиентов (CG), итерации которого осуществляются до выполнения условия

$$\left\|\vec{b} - A\vec{x}_k\right\| \le \varepsilon \left\|\vec{b} - A\vec{x}_0\right\|, \quad 0 < \varepsilon \ll 1. \tag{2}$$

Для решения СЛАУ большого размера требуется применение параллельных компьютеров. Ниже будут рассмотрены итерационные методы решения задачи (1), существенно использующие матрицы $H \approx A^{-1}$, называемые явными предобусловливателями. В этих методах основная доля вычислительной работы приходится на повторные умножения разреженной матрицы на вектор, а операции решения систем с треугольными матрицами отсутствуют. Поэтому реализующие их параллельные алгоритмы хорошо приспособлены к параллельной реализации с использованием (MPI+OpenMP)-подхода.

В формуле (1) предполагается, что матрица A уже переупорядочена, $A = A_P = P\widehat{A}P^\top$, где P — матрица перестановки; $\widehat{A}$ представляет собой матрицу коэффициентов исходной задачи. В настоящей работе применяются переупорядочения, уменьшающие среднюю ширину ленты матрицы, предложенные в работах [2, 3] и являющиеся обобщениями упорядочения [4]. Подход, предложенный в этих работах, позволяет одновременно произвести разбиение области расчета на подобласти.

В настоящей работе рассматривается предобусловливание, предложенное в [5, 6] (см. также [7]), основанное на использовании приближенного обратного симметрично-треугольного разложения

$$\widehat{G}A\widehat{G}^\top = I_n + E$$

матрицы A (уже переупорядоченной), где $\widehat{G}$ — разреженная нижняя треугольная матрица с положительными диагональными элементами; I_n — единичная матрица; E — матрица погрешности. Соответствующий предобусловливатель имеет вид

$$H = \widehat{G}^\top\widehat{G} \approx A^{-1},$$

причем структура разреженности $\widehat{G}$ задается как множество ненулевых позиций, определяемых специальным образом [8]. Другие способы выбора структуры расположения ненулевых элементов $\widehat{G}$ обсуждались, в частности, в [9—11]. При построении предобусловливателя будем использовать отмасштабированную матрицу $A_{SP} = D_{A_P}^{-1/2}A_PD_{A_P}^{-1/2}$, где D_{A_P} — диагональная часть матрицы A_P. Диагональные элементы матрицы A_{SP} равны единице. Значения ненулевых элементов матрицы G определяются из условия оптимизации K-числа обусловленности матрицы $GA_{SP}G^\top$ (см. ниже). Очевидно, что $\widehat{G} = GD_{A_P}^{-1/2}$. Такое предобусловливание сокращенно принято называть IIC (Inverse Incomplete Cholesky).

В настоящей работе рассматривается также предобусловливание при помощи блочного метода Якоби в сочетании с неполным обратным треугольным разложением блоков (BJIIC), предложенного в работе [3]. В этом методе сначала находится предобусловливатель Якоби блочного метода CG, а затем для каждого блока строится неполное обратное треугольное разложение. При построении этого предобусловливателя и при его обращении не требуется обмена информацией между процессорами.

Аналогичный подход рассматривался в недавней статье [12] (см. также цитированную в [12] литературу). Представленное в ней предобусловливание основано на ILU-разложении матрицы коэффициентов с последующим построением приближенных обратных матриц для соответствующих нижнего и верхнего треугольных сомножителей.

В настоящей работе описаны методы построения предобусловливателей IIC, BJIIC, предложены алгоритмы параллельной реализации методов IIC-CG, BJIIC-CG с использованием (MPI+OpenMP)-подхода. Приводятся результаты расчетов нескольких тестовых задач из коллекции университета Флориды [1]. Проводится сравнение времени решения задач с использованием MPI- и (MPI+OpenMP)-подходов методами IIC-CG, BJIIC-CG и методом CG с предобусловливанием Якоби (J-CG) (в котором $H = D_A^{-1}$, где D_A — диагональная часть матрицы A), а также с предобусловливанием при помощи блочного неполного обратного треугольного разложения второго порядка BIIC2-CG [4, 13].

1. Предобусловленный метод сопряженных градиентов

Пусть требуется решить СЛАУ (1) и для матрицы $A = A_P$ построена приближенная обратная матрица вида $H = \widehat{G}^\top\widehat{G}$. Алгоритм предобусловленного метода CG (см., например, [14]) имеет следующий вид:

$\vec{r}_0 = \vec{b} - A\vec{x}_0; \quad \vec{p}_0 = H\vec{r}_0; \quad \gamma_0 = \vec{r}_0^\top\vec{p}_0;$

для $k = 0, \ldots,$ *пока* $(\vec{r}_k^\top\vec{r}_k) \le \varepsilon^2(\vec{r}_0^\top\vec{r}_0)$, *выполнять*

$\vec{q}_k = A\vec{p}_k; \quad \alpha_k = \gamma_k/(\vec{p}_k^\top\vec{q}_k);$

$\vec{x}_{k+1} = \vec{x}_k + \alpha_k\vec{p}_k; \quad \vec{r}_{k+1} = \vec{r}_k - \alpha_k\vec{q}_k; \quad \vec{z}_{k+1} = H\vec{r}_{k+1};$

$\gamma_{k+1} = \vec{r}_{k+1}^\top\vec{z}_{k+1}; \quad \beta_k = \gamma_{k+1}/\gamma_k; \quad \vec{p}_{k+1} = \vec{z}_{k+1} + \beta_k\vec{p}_{k+1},$

где $0 < \varepsilon \ll 1$. В [6, 15] доказана оценка сходимости предобусловленного метода CG:

$$\|\vec{r}_k\|_H \le \left(K(HA)^{1/k} - 1\right)^{k/2} \|\vec{r}_0\|_H, \quad K(B) = \frac{(n^{-1}\mathrm{trace}B)^n}{\det B}, \tag{3}$$

где $K(B)$ представляет собой K-число обусловленности симметричной матрицы $B = \widehat{G}A_P\widehat{G}^\top = GA_{SP}G^\top$. При этом для метода CG справедлива следующая (упрощенная) верхняя граница числа итераций:

$$k_{\varepsilon_1} \le \log_2 K(HA) + \log_2(\varepsilon_1^{-1}), \tag{4}$$

где $0 < \varepsilon_1 \ll 1$ задает требуемое уменьшение H-нормы невязки $\left\|\vec{b} - A\vec{x}_k\right\|_H$.

2. Построение матрицы G, обеспечивающее K-оптимальное предобусловливание

Предположим, что матрица A переупорядочена и отмасштабирована ($A = A_{SP}$). Опишем алгоритм [16] построения нижнетреугольной невырожденной матрицы G, оптимальной в смысле минимизации величины $K(H_sA) = K(GAG^\top)$, где $H_s = G^\top G$. Допустим, что заранее заданы позиции структурно ненулевых элементов i-й строки нижнетреугольной матрицы G:

$$(G)_{i,j_i(1)}, \ldots, (G)_{i,j_i(m_i)}, \quad 1 \le i \le n,$$

где $1 \le j_i(1) < \ldots < j_i(m_i) = i$ — i-й список соответствующих столбцовых индексов. Тогда, как доказано в [16], минимум K-числа обусловленности $K(H_sA)$, определенного соотношением (3), достигается при

$$(G)_{i,j_i(p)} = \frac{(S_i^{-1})_{p,m_i}}{\sqrt{(S_i^{-1})_{m_i,m_i}}}.$$

Здесь S_i представляет собой главную подматрицу размером m_i матрицы A, построенную на указанном i-м подмножестве столбцовых индексов, т. е.

$$(S_i)_{p,q} = (A)_{j_i(p),j_i(q)}, \quad 1 \le p \le m_i, \quad 1 \le q \le m_i.$$

Пусть $\vec{z}_i$ — вектор структурно ненулевых элементов i-й строки матрицы G длиной m_i ($z_i(p) = (G)_{i,j_i(p)}$). Нетрудно убедиться в том, что справедлива еще более простая формула для вычисления $\vec{z}_i$, выражающая его через симметричную треугольную факторизацию [16]

$$S_i = L_iL_i^\top$$

в виде

$$\vec{z}_i = L_i^{-1}\vec{\nu}_i,$$

где вектор $\vec{\nu}_i$ длиной m_i имеет вид $\vec{\nu}_i = (0, \ldots, 0, 1)^\top$. Таким образом, для каждого i достаточно вычислить треугольное разложение $S_i = L_iL_i^\top$ матрицы порядка m_i и затем решить одну треугольную систему $L_i^\top\vec{z}_i = \vec{\nu}_i$ того же порядка. Напомним, что наличие свойства K-оптимальности построенного предобусловливания отвечает минимизации соответствующей верхней границы оценки числа итераций (4) предобусловленного метода CG.

Для выбора позиций ненулевых элементов матрицы G будем использовать способ [8], состоящий из двух этапов. На первом этапе множество позиций ненулевых элементов искомой матрицы выбирается в виде множества позиций ненулевых элементов матрицы A^q, где q — показатель степени. Затем предварительно строится матрица $\widetilde{G}$, как описано выше. На втором этапе осуществляется прореживание множества позиций ненулевых элементов матрицы $\widetilde{G}$: отбрасываются позиции, в которых $0 < |g_{ij}| \le \tau_0 g_{ii}$, где $j < i$; g_{ij} — элементы матрицы $\widetilde{G}$; $0 < \tau_0 \ll 1$. Затем строится матрица G, как описано выше, в которой используется полученное прореженное множество ненулевых позиций. Показатель степени q подбирается так, чтобы время решения СЛАУ (1) методом IIC-CG в сумме

со временем построения предобусловливателя было минимальным. Для достаточно заполненных матриц выбор $q = 1$ обычно приводит к хорошим результатам.

Необходимость указанного выше повторного вычисления матрицы G объясняется возможностью резкого ухудшения качества предобусловливания при обнулении ее элементов с относительно малыми значениями. Действительно, при этом нарушается оптимальность выбора значений оставшихся неизмененными ненулевых элементов, что обычно приводит к недопустимому возрастанию числа итераций метода CG, если матрица A плохо обусловлена.

3. Алгоритм параллельной реализации

Пусть матрица A ($A = A_P$) переупорядочена и разбита на блоки, причем на блочной диагонали расположены p квадратных блоков размерами $n_s \times n_s$, $1 \le s \le p$. Обозначим $k_s = n_1 + \ldots + n_s$. Ниже описан алгоритм параллельной реализации, предложенный в [17].

На каждом процессоре с номером $s = 1, \ldots, p$ будем строить матрицу G_s, содержащую соответствующие n_s строк матрицы G. Сначала на каждом процессоре с номером s создадим матрицу $\bar{S}_s$ размером $m_s \times m_s$, где $m_s \ge n_s$, которая будет содержать все элементы матрицы A_P, необходимые для построения всех строк матрицы G с номерами $k_{s-1} + 1 \le i \le k_s$. Для этого нужно осуществить соответствующие пересылки. Затем производится масштабирование матриц $\bar{S}_s$, что осуществляется всеми процессорами одновременно и независимо. При этом находятся элементы диагональных матриц $D_{A_P,s}$, которые являются соответствующими элементами матрицы D_{A_P}.

На каждом процессоре при построении матрицы G_s используется алгоритм, аналогичный алгоритму построения матрицы G (см. разд. 2), только вместо матрицы A ($A = A_{SP}$) берется отмасштабированная матрица $\bar{S}_s$. При этом для всех строк внутри каждой подобласти все вычисления выполняются независимо с применением OpenMP-технологии. Для распределения вычисления значений элементов матрицы G_s по строкам используются простые циклы с опцией `schedule static` (директива `do` с опцией `schedule static`), задающей, каким образом итерации цикла распределяются по нитям. Затем на всех процессорах одновременно вычисляются матрицы $\widehat{G}_s = G_s D_{A_P,s}^{-1/2}$ и $\widehat{G}_s^\top$. Как показывают расчеты задач, приведенные в [17], построение матрицы $\widehat{G}_s^\top$ занимает ничтожно малое время по сравнению со временем вычисления предобусловливателя и выполнением итерационного процесса метода IIC-CG.

При параллельной реализации вычислений $\vec{p} = H\vec{r} = \widehat{G}^\top \widehat{G}\vec{r}$ сначала производится пересылка с процессоров с меньшими номерами на процессоры с большими номерами значений r_m (здесь $m \le$ $\le k_{s-1}$ — индекс компоненты вектора $\vec{r}$), необходимых для вычисления $\vec{\tilde{z}}_s = \widehat{G}_s\vec{r}$ на каждом процессоре с номером s. Затем на процессоре с номером s ($1 \le s \le p$) вычисляются $\vec{\tilde{z}}_s = \widehat{G}_s\vec{r}$ и соответствующие слагаемые $\vec{p}_l = \widehat{G}_s\vec{\tilde{z}}_s$ ($l \le s$ — номер процессора, содержащего соответствующую компоненту вектора $\vec{p}$).

Как показали расчеты, использование для вычисления $\vec{p} = \widehat{G}^\top \widehat{G}\vec{r}$ алгоритма 1 из [17], в котором не требуется явно заданной матрицы $\widehat{G}_s^\top$ и который успешно применялся при MPI-распараллеливании, часто оказывается неэффективно при (MPI+OpenMP)-подходе. Поэтому вычисления будем проводить в два этапа: $\vec{\tilde{z}}_s = \widehat{G}_s\vec{r}$ и $\vec{p}_l = \widehat{G}_s^\top\vec{\tilde{z}}_s$, используя алгоритм 2 из работы [17]. Для распределения вычислений элементов векторов $\vec{\tilde{z}}$, $\vec{p}$ по строкам i будем в обоих случаях использовать директиву `do` с опцией `schedule static`.

Далее, после необходимых пересылок (с процессоров с большими номерами на процессоры с меньшими номерами), на каждом процессоре с номером s выполняется суммирование вида $\vec{p} = \vec{p}_s + \sum\limits_{l>s} \vec{p}_l$, где $\vec{p}_l$ — нужная часть $\vec{p}$, которая вычисляется на процессоре с номером l при выполнении на нем своей доли вычислений для получения $\vec{p} = \widehat{G}^\top(\widehat{G}\vec{r})$.

Рассмотрим, как осуществляется вычисление $\vec{q} = A\vec{p}$. Матрица A хранится в памяти в распределенном CRS-формате [1], содержит как верхний, так и нижний треугольники. Сначала выполняется пересылка значений компонент $\vec{p}$, необходимых для вычисления части вектора $\vec{q}$ на рассматриваемом процессоре, хранящихся в памяти других процессоров. Компоненты q_i вектора $\vec{q}$ на каждом

процессоре для своей группы индексов i вычисляются с применением OpenMP-технологии, для чего используется директива `do` с опцией `schedule static`. Заметим, что, как показали расчеты методом J-CG задач, приведенных в разд. 7, использование параметров `dynamic`; `guided`; `guided, 6` в этой опции в большинстве случаев не позволяет ускорить вычисления по сравнению с использованием опции `static`. MPI-реализация векторных операций и вычислений скалярных произведений хорошо известна. В случае (MPI+OpenMP)-подхода для выполнения векторных операций и вычислений частичных сумм в скалярных произведениях использовалась директива `do` с опцией `schedule static`.

4. Предобусловливание при помощи блочного метода Якоби в сочетании с неполным обратным треугольным разложением

Пусть матрица A переупорядочена и разбита на блоки, причем на блочной диагонали расположены p квадратных блоков размерами $n_s \times n_s$, $1 \le s \le p$. Обозначим, как и ранее, $k_s = n_1 + \ldots + n_s$. Определим прямоугольные матрицы

$$W_s = \left(e_{k_{s-1}+1} | \ldots | e_{k_s}\right),$$

столбцы которых являются единичными n-векторами, где $k_{s-1}+1, \ldots, k_s$ представляют собой индексы s-го блока. Построим матрицы $W_s^\top A W_s = A_s$ размерами $n_s \times n_s$. Проведем масштабирование матриц A_s, получим матрицы A_{0s}. Построим неполные обратные треугольные разложения для этих матриц: $\bar{G}_s^\top \bar{G}_s \approx A_{0s}^{-1}$. В качестве предобусловливателя будем использовать

$$H = \sum_{s=1}^{p} W_s D_{A_s}^{-1/2} \bar{G}_s^\top \bar{G}_s D_{A_s}^{-1/2} W_s^\top. \tag{5}$$

Предобусловливание (5) ранее предложено в [3] и названо предобусловливанием при помощи блочного метода Якоби в сочетании с неполным обратным треугольным разложением блоков (BJIIC). Вычисление элементов матриц $\bar{G}_s$ $(s = 1, \ldots, p)$ осуществляется аналогично описанному в разд. 2, но вместо отмасштабированной матрицы A используются матрицы A_{0s}. При вычислении матрицы $\bar{G}_s$ на каждом процессоре не требуется информации, хранящейся на других процессорах, все процессоры могут работать одновременно. Кроме того, каждая строка матрицы $\bar{G}_s$ вычисляется независимо от других строк этой матрицы с применением OpenMP-технологии аналогично описанному в разд. 3. Вычисление матриц $\breve{G}_s = G_s D_{A_s}^{-1/2}$ и $\breve{G}_s^\top$ на всех процессорах осуществляется одновременно, причем формирование $\breve{G}_s^\top$ занимает пренебрежимо малое время [17]. При выполнении операции $\vec{p} = \breve{G}_s^\top (\breve{G}_s \vec{r})$ все процессоры выполняют работу одновременно, пересылок не требуется. При этом применяется OpenMP-технология, аналогичная описанной в разд. 3 (см. там же параллельную реализацию остальных этапов алгоритма).

5. Оценки уменьшения K-числа обусловленности в методах BJIIC, IIC

Предполагаем, что матрица A предварительно переупорядочена и отмасштабирована. Заметим, что предобусловливания BJIIC и IIC отвечают одному и тому же методу, но с разным выбором структуры заполненности матрицы G, а именно: из структуры ненулей матрицы G метода IIC удаляются все блочно-внедиагональные позиции, и получается структура матрицы $\bar{G}$ метода BJIIC.

Согласно [15] справедливо следующее неравенство:

$$\frac{K(G^\top G A)}{K(A)} \le \exp\left(-\frac{1}{\|A\|}\sum_{i=1}^{n} a_i^\top a_i\right) = \exp\left(-\frac{1}{\|A\|}\sum_{i=1}^{n}\sum_{s=1}^{m_i-1} A_{i,j_i(s)}^2\right), \tag{6}$$

где $j_i(s)$ для метода IIC были определены в разд. 2. Неравенство (6) содержит оценку уменьшения K-числа обусловленности при применении IIC-предобусловливания по сравнению с K-числом

обусловленности при применении J-предобусловливания. Заметим, что при $q \geq 1$ для выбранного способа построения структуры G и способа масштабирования матрицы A в методе IIC справедлива формула $\sum\limits_{i=1}^{n}\sum\limits_{s=1}^{m_i-1} A^2_{i,j_i(s)} = 0{,}5\,\|I-A\|_F^2$, где $\|*\|_F$ обозначает фробениусову норму матрицы. Если обозначить теперь индексы ненулевых элементов $\bar{G}$ через $\bar{j}_i(s)$, то для метода BJIIC получим

$$\frac{K(\bar{G}^\top \bar{G}A)}{K(A)} \leq \exp\left(-\frac{1}{\|A\|}\sum_{i=1}^{n}\sum_{s=1}^{m_i-1} A^2_{i,\bar{j}_i(s)}\right). \tag{7}$$

Обозначая $A_0 = BlockDiag(A)$, можно воспользоваться очевидными соотношениями $\sum\limits_{i=1}^{n}\sum\limits_{s=1}^{m_i-1} A^2_{i,\bar{j}_i(s)} = 0{,}5\,\|I-A_0\|_F^2$ и $\|I-A\|_F^2 = \|I-A_0\|_F^2 + \|A-A_0\|_F^2$. Так что с учетом (6), (7) сравнение двух предобусловливаний дает следующие два неравенства:

– для метода IIC

$$\frac{K(G^\top GA)}{K(A)} \leq \exp\left(-\frac{\|I-A\|_F^2}{2\,\|A\|}\right);$$

– для метода BJIIC

$$\frac{K(\bar{G}^\top \bar{G}A)}{K(A)} \leq \exp\left(-\frac{\|I-A_0\|_F^2}{2\,\|A\|}\right) = \exp\left(-\frac{\|I-A\|_F^2}{2\,\|A\|}\right)\exp\left(\frac{\|A-A_0\|_F^2}{2\,\|A\|}\right).$$

Таким образом, учитывая (4), получаем оценки чисел итераций:

– в методе IIC-CG

$$\widehat{k}_{\varepsilon_1} \leq \log_2(K(A)) - \frac{\|I-A\|_F^2}{2\,\|A\|}\log_2 e + \log_2(\varepsilon_1^{-1}); \tag{8}$$

– в методе BJIIC-CG

$$\widehat{k}_{\varepsilon_1} \leq \log_2(K(A)) - \frac{\|I-A\|_F^2}{2\,\|A\|}\log_2 e + \frac{\|A-A_0\|_F^2}{2\,\|A\|}\log_2 e + \log_2(\varepsilon_1^{-1}). \tag{9}$$

Оценки чисел итераций в обоих методах меньше, чем в методе J-CG. Ухудшение оценки (9) по сравнению с (8) связано с величиной $\|A-A_0\|_F$. Если матрица A незначительно отличается от A_0, можно ожидать, что правые части этих оценок отличаются не очень сильно. Незначительное отличие матрицы A_0 от A наблюдается при достаточно хорошем разбиении, когда вне блочной диагонали оказывается не слишком много ненулевых элементов A (см., например [3]). Кроме того, на величину $\|A-A_0\|_F$ оказывают влияние также абсолютные значения элементов блочно-внедиагональной части матрицы A. При разумном числе процессоров и хорошем разбиении оценки (8) и (9) отличаются не очень сильно. Можно ожидать, что рост числа итераций с ростом числа процессоров в методе BJIIC-CG будет приемлемым.

6. Предобусловливание при помощи блочного неполного обратного треугольного разложения второго порядка

Рассмотрим метод предобусловливания BIIC2, предложенный в [13]. Пусть матрица A переупорядочена и разбита на блоки, причем на блочной диагонали расположены p квадратных блоков размерами $n_s \times n_s$, $1 \leq s \leq p$. Обозначим, как и ранее, $k_s = n_1 + \ldots + n_s$, и пусть $m_s \geq n_s$ — размеры расширенных блоков. Определим прямоугольные матрицы

$$V_s = \left(e_{j_s(1)}\,|\ldots|\,e_{j_s(m_s-n_s)}\,|e_{k_{s-1}+1}\,|\ldots|e_{k_s}\right),$$

столбцы которых являются единичными n-векторами, где $k_{s-1}+1, \ldots, k_s$ представляют собой индексы s-го блока, а $j_s(1), \ldots, j_{s(m_s-n_s)}$ являются индексами перекрытия, причем $j_s(.) \le k_{s-1}$. Используя приближенное треугольное разложение второго порядка "по значению", описанное в [13, 15], для аппроксимации подматриц размером $m_s \times m_s$

$$V_s^\top A V_s = U_s^\top U_s + U_s^\top R_s + R_s^\top U_s - S_s,$$

где $\|R_s\| = O(\tau), \|S_s\| = O(\tau^2)$ и $0 < \tau \ll 1$ — порог отсечения, определим предобусловливатель

$$H = \sum_{s=1}^{p} V_s U_s^{-1} \begin{pmatrix} 0 & 0 \\ 0 & I_{n_s} \end{pmatrix} U_s^{-\top} V_s^\top.$$

При параллельной реализации метода BIIC2-CG при вычислении матрицы предобусловливания и ее обращении использовался только MPI-подход, так как применение OpenMP-технологии при реализации этих этапов сильно затруднено (в частности, из-за рекурсивного характера вычислений при обращении матрицы предобусловливания). При осуществлении итерационного процесса (MPI+OpenMP)-подход был использован при умножении матрицы на вектор, при выполнении векторных операций и при вычислении скалярных произведений (см. разд. 3).

7. Результаты расчетов

Все программы, реализующие применение методов IIC-CG, BJIIC-CG, J-CG, BIIC2-CG для решения системы уравнений (1), были написаны на языке FORTRAN с использованием (MPI+OpenMP)-подхода. Расчеты проводились на многопроцессорной вычислительной системе МВС-10П, установленной в Межведомственном суперкомпьютерном центре (МСЦ) РАН. Для тестирования рассматриваемых параллельных методов использовались некоторые матрицы из коллекции университета Флориды [1].

Перечислим используемые тестовые матрицы и укажем источники их происхождения: `x104` — моделирование конструкций (балочное сочленение); `m_t1` — моделирование конструкций (трубчатое сочленение); `hood` — моделирование конструкций; `pwtk` — модель воздуховода под давлением; `msdoor` — моделирование конструкций (дверь среднего размера). В табл. 1 указаны некоторые свойства этих матриц, причем значения обусловленности $Cond(A_S)$, где $A_S = (D_A)^{-1/2} A (D_A)^{-1/2}$ — матрица системы уравнений после масштабирования, взяты из работы [16]; Id — количество строк с диагональным преобладанием; Ip — количество положительных внедиагональных элементов; NZA — число ненулевых элементов матрицы A; $nz_{\min}$, $nz_{\max}$ — минимальное и максимальное числа ненулевых элементов в строках матрицы A.

Решалось уравнение $A\vec{x} = \vec{b}$, где в правой части $b_i \equiv 1$, начальное приближение $\vec{x}_0 \equiv 0$, счет продолжался до выполнения условия (2), при $\varepsilon = 10^{-8}$. Для разбиения области расчета использовался способ [3]. При построении предобусловливателей IIC и BJIIC во всех задачах использовались значения параметров $\tau_0 = 0{,}01$, $q = 1$. При построении предобусловливателей BIIC2 в задачах с матрицами `hood`, `pwtk`, `msdoor` использовались значения параметров $q_1 = 2$, $\tau = 0002$, а с матрицами

Таблица 1

Свойства использованных матриц из коллекции университета Флориды

Матрица	N	NZA	Id	Ip	$nz_{\min}$	$nz_{\max}$	$Cond(A_S)$
`m_t1`	97 578	9 753 570	1	4 648 398	48	237	$0{,}47{\cdot}10^{10}$
`hood`	220 542	9 895 422	9 910	4 879 422	1	77	$0{,}55{\cdot}10^{6}$
`pwtk`	217 918	11 524 432	325	5 407 348	2	180	$0{,}26{\cdot}10^{9}$
`msdoor`	415 863	19 173 163	11 125	9 350 756	1	77	$0{,}19{\cdot}10^{9}$
`x104`	108 384	8 713 602	2 255	4 059 880	8	270	$0{,}1{\cdot}10^{11}$

`m_t1`, `x104` $q_1 = 1$, $\tau = 0{,}0002$, где q_1 — глубина налегания. Выбор параметров q, q_1, τ продиктован минимизацией времени счета методами IIC-CG,BJIIC-CG, BIIC2–CG при использовании разбиения [3] и MPI.

Каждый вычислительный модуль решающего поля на многопроцессорной вычислительной системе МВС-10П имеет в своем составе два процессора Xeon E5-2690, 64 Гб оперативной памяти, два сопроцессора Intel Xeon Phi 7110X. Расчеты на многопроцессорной вычислительной системе МВС-10П показали, что решение СЛАУ (1) методом J-CG для задачи с матрицей `m_t1` происходит быстрее всего, если в каждом вычислительном модуле решающего поля использовать два процессора Xeon E5-2690 и отказаться от использования сопроцессоров Intel Xeon Phi 7110X. Поэтому расчеты всех задач методами J-CG, IIC-CG, BJIIC-CG, BIIC2-CG на вычислительной системе МВС-10П производились с использованием в каждом вычислительном модуле только двух процессоров Xeon E5-2690.

В табл. 2—4 приведены время счета в секундах и число итераций при решении задач на p процессорах методами J-CG, IIC-CG и BJIIC-CG без использования OpenMP-технологии (в скобках указано число итераций), а также минимальное время счета (по указанному в скобках числу нитей 4, 8 или 16) при использовании OpenMP-технологии, как описано выше. В работе [17] приведено время счета этих задач методом J-CG при использовании 4, 8 или 16 нитей OpenMP. Там же приведены время счета этих задач методами IIC-CG и BJIIC-CG на p процессорах при использовании 4, 8 или 16 нитей OpenMP, время вычисления матрицы $\widehat{G}_s$ и матрицы $\breve{G}_s$, время вычисления матриц $\widehat{G}_s^\top$ и $\breve{G}_s^\top$, время счета итерационных процессов в методах IIC-CG и BJIIC-CG и общее время решения. Заметим, что дальнейшее увеличение числа используемых процессоров может привести к ухудшению масштабируемости.

При использовании метода BJIIC-CG для решения задач из коллекции университета Флориды наблюдается некоторый (не всегда монотонный) рост числа итераций с ростом числа процессоров. Это соответствует приведенным выше результатам теоретического исследования. Немонотонность роста числа итераций, возможно, связана с особенностями разбиения области. Однако благодаря значительному уменьшению числа пересылок время счета методом BJIIC-CG обычно не очень сильно отличается от времени счета методом IIC-CG, а алгоритм параллельной реализации существенно проще.

Использование (MPI+OpenMP)-подхода в методе BIIC2-CG при решении указанных выше задач не дало ускорения счета по сравнению с использованием только MPI. Поэтому в табл. 5 для метода BIIC2-CG приведено время счета и число итераций без использования OpenMP-технологии.

На рис. 1—5 приведены графики зависимости времени счета рассматриваемых задач разными методами от числа процессоров с применением MPI- и (MPI+OpenMP)-подходов в логарифмическом масштабе.

Таблица 2

Время счета (в секундах) и число итераций при решении тестовых задач методом J-CG на p процессорах без использования и с использованием OpenMP-технологии

Матрица	$p = 8$		$p = 16$		$p = 32$		$p = 64$		$p = 100$	
	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP
`m_t1`	540,9 (536 545)	125,15 (16)	282,0 (536 667)	82,4 (8)	162,0 (534 807)	70,09 (4)	95,52 (536 474)	53,6 (4)	76,6 (537 466)	73,94 (4)
`hood`	7,04 (6 078)	2,17 (16)	3,87 (6 056)	1,74 (16)	2,14 (6 052)	2,42 (16)	1,3 (6 061)	2,57 (8)	1,19 (6 059)	3,55 (8)
`pwtk`	134,58 (74 111)	37,76 (4)	48,43 (74 065)	12,69 (4)	48,43 (74 065)	12,69 (4)	16,14 (74 692)	10,66 (8)	12,71 (74 776)	15,04 (4)
`msdoor`	283,9 (83 510)	92,2 (8)	111,12 (83 496)	24,14 (16)	53,59 (83 531)	17,44 (8)	32,52 (83 493)	14,76 (16)	21,71 (83 466)	16,01 (8)
`x104`	894,61 (884 621)	192,77 (8)	449,99 (888 104)	140,06 (4)	257,20 (882 639)	112,48 (8)	187,28 (884 943)	118,65 (4)	123,5 (882 803)	121,58 (4)

Таблица 3

Время счета (в секундах) и число итераций при решении тестовых задач методом IIC-CG на p процессорах без использования и с использованием OpenMP-технологии

Матрица	$p = 8$		$p = 16$		$p = 32$		$p = 64$		$p = 100$	
	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP
m_t1	14,71	6,81	7,27	3,21	3,85	2,30	2,46	2,17	2,08	1,76
	(3 053)	(16)	(3 069)	(8)	(3 055)	(8)	(2 984)	(4)	(2 938)	(8)
hood	3,58	1,42	1,90	0,69	1,01	0,45	0,70	0,43	0,72	0,50
	(424)	(8)	(409)	(16)	(415)	(8)	(421)	(8)	(415)	(8)
pwtk	16,03	8,54	6,19	3,23	3,18	2,19	1,95	1,88	1,66	1,82
	(3 696)	(8)	(3 571)	(8)	(3 600)	(8)	(3 581)	(8)	(3 543)	(8)
msdoor	30,60	13,37	15,48	7,98	6,20	3,65	3,81	2,50	2,68	2,14
	(3 934)	(8)	(3 889)	(8)	(3 854)	(8)	(3 881)	(8)	(3 800)	(8)
x104	19,78	13,4	9,31	6,08	5,37	4,67	3,74	3,97	2,67	3,85
	(8 355)	(16)	(8 326)	(8)	(8 057)	(8)	(7 928)	(8)	(7 775)	(8)

Таблица 4

Время счета (в секундах) и число итераций при решении тестовых задач методом BJIIC-CG на p процессорах без использования и с использованием OpenMP-технологии

Матрица	$p = 8$		$p = 16$		$p = 32$		$p = 64$		$p = 100$	
	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP	Без OpenMP	С OpenMP
m_t1	15,25	5,56	6,49	2,45	3,58	2,07	2,11	2,19	1,44	1,70
	(3 335)	(8)	(3 296)	(16)	(3 439)	(8)	(3 649)	(4)	(3 769)	(8)
hood	3,42	0,97	1,69	0,56	0,89	0,36	0,51	0,28	0,72	0,37
	(426)	(16)	(449)	(8)	(460)	(8)	(463)	(8)	(454)	(8)
pwtk	16,16	6,21	6,04	2,45	3,26	1,86	1,94	1,76	1,41	1,69
	(3 784)	(8)	(3 776)	(16)	(3 964)	(4)	(4 120)	(4)	(4 158)	(4)
msdoor	31,74	11,42	16,48	6,83	8,35	4,16	4,83	3,67	3,41	2,71
	(4 135)	(8)	(4 427)	(8)	(5 638)	(16)	(5 609)	(4)	(5 216)	(4)
x104	19,51	8,70	9,25	4,76	5,37	4,31	3,69	4,04	2,41	3,68
	(8 240)	(8)	(8 633)	(16)	(9 465)	(8)	(9 355)	(4)	(9 522)	(4)

Таблица 5

Время счета (в секундах) и число итераций при решении разных задач методом BIIC2-CG на p процессорах с использованием только MPI-подхода

Матрица	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 100$
m_t1	22,18 (1 506)	8,67 (1 300)	3,31 (1 144)	1,48 (1 111)	1,19 (1 139)
hood	4,09 (71)	2,14 (60)	0,98 (55)	0,56 (57)	0,42 (46)
pwtk	12,06 (1 512)	5,62 (1 326)	2,58 (1 260)	1,4 (1 263)	1,14 (1 345)
msdoor	22,33 (1 106)	9,84 (822)	6,22 (921)	2,69 (759)	1,7 (671)
x104	59,49 (15 542)	27,6 (14 105)	14,43 (13 283)	9,39 (11 286)	5,61 (10 456)

Как видно из табл. 2 и рис. 1—5, при решении задач методом J-CG применение OpenMP-технологии позволяет существенно ускорить вычисления в случае достаточно больших и не очень сильно разреженных матриц. В случае сильно разреженных матриц использование OpenMP-технологии позволяет ускорить расчет лишь для небольшого числа процессоров (см. результаты для задачи с матрицей

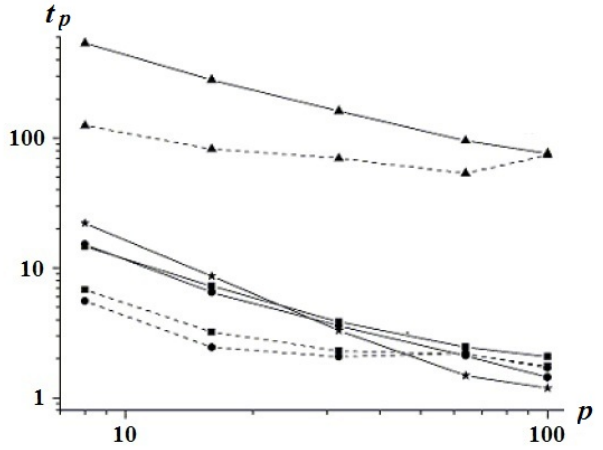

Рис. 1. Время счета задачи с матрицей `m_t1` различными методами: —▲— — J-CG, MPI; --▲-- — J-CG, MPI+OpenMP; —■— — IIC-CG, MPI; --■-- — IIC-CG, MPI+OpenMP; —●— — BJIIC-CG, MPI; --●-- — BJIIC-CG, MPI+OpenMP; —⋆— — BIIC2-CG, MPI

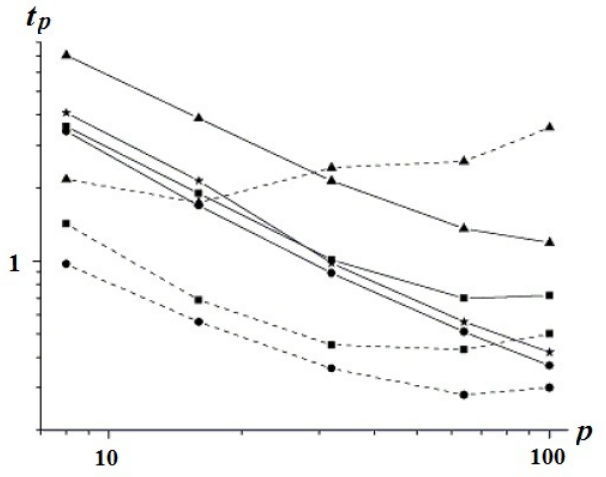

Рис. 2. Время счета задачи с матрицей `hood` различными методами: —▲— — J-CG, MPI; --▲-- — J-CG, MPI+OpenMP; —■— — IIC-CG, MPI; --■-- — IIC-CG, MPI+OpenMP; —●— — BJIIC-CG, MPI; --●-- — BJIIC-CG, MPI+OpenMP; —⋆— — BIIC2-CG, MPI

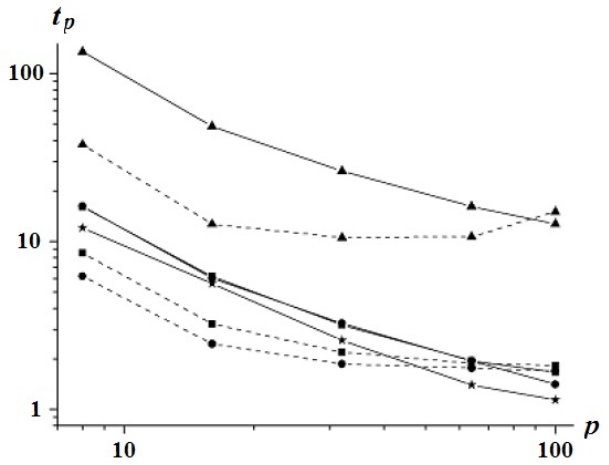

Рис. 3. Время счета задачи с матрицей `pwtk` различными методами: —▲— — J-CG, MPI; --▲-- — J-CG, MPI+OpenMP; —■— — IIC-CG, MPI; --■-- — IIC-CG, MPI+OpenMP; —●— — BJIIC-CG, MPI; --●-- — BJIIC-CG, MPI+OpenMP; —⋆— — BIIC2-CG, MPI

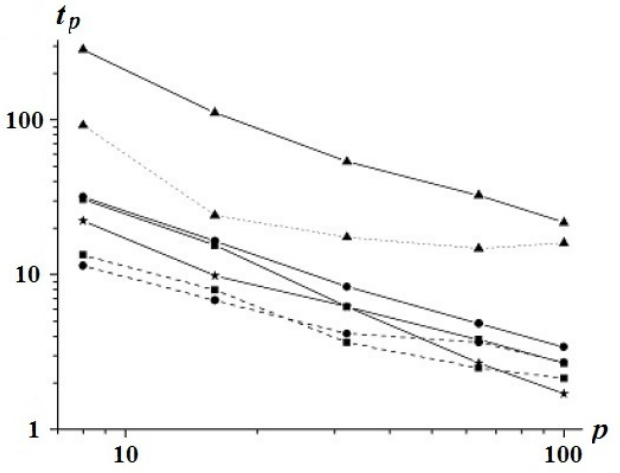

Рис. 4. Время счета задачи с матрицей `msdoor` различными методами: —▲— — J-CG, MPI; --▲-- — J-CG, MPI+OpenMP; —■— — IIC-CG, MPI; --■-- — IIC-CG, MPI+OpenMP; —●— — BJIIC-CG, MPI; --●-- — BJIIC-CG, MPI+OpenMP; —⋆— — BIIC2-CG, MPI

`hood`). В случае не очень больших, но заполненных матриц (`m_t1`, `x104`) OpenMP-технология тоже позволяет существенно ускорить вычисления для разумного числа p.

Как видно из табл. 3 и рис. 1—5, при решении задач методом IIC-CG применение OpenMP-технологии позволяет существенно ускорить вычисления в случае достаточно больших и не очень сильно разреженных матриц. В случае не очень больших, но более заполненных матриц (`m_t1`) OpenMP-технология тоже позволяет существенно ускорить вычисления для разумного числа p.

Как видно из табл. 4 и рис. 1—5, при решении задач методом BJIIC-CG применение OpenMP-технологии позволяет существенно ускорить вычисления в случае достаточно больших и не очень сильно разреженных матриц. Для задачи с матрицей `m_t1` OpenMP-технология позволяет ускорить

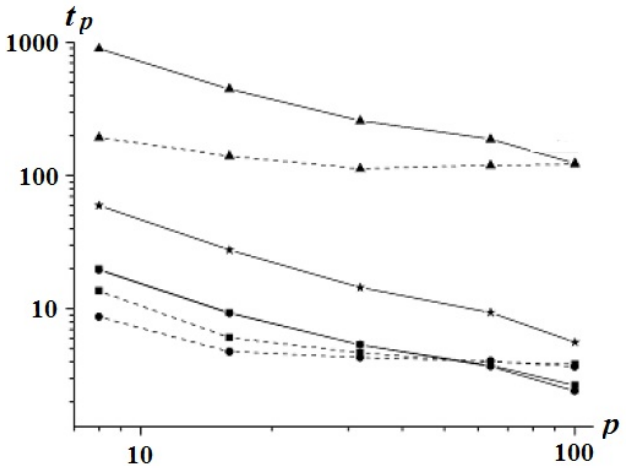

Рис. 5. Время счета задачи с матрицей x104 различными методами: —▲— — J-CG, MPI; --▲-- — J-CG, MPI+OpenMP; —■— — IIC-CG, MPI; --■-- — IIC-CG, MPI+OpenMP; —●— — BJIIC-CG, MPI; --●-- — BJIIC-CG, MPI+OpenMP; —⋆— — BIIC2-CG, MPI

расчет при использовании до 64 процессов. Более слабый эффект по сравнению с J-CG при применении OpenMP-технологии для решения задачи с матрицей x104 методами IIC-CG, BJIIC-CG связан с недостаточным ускорением при обращении предобусловливателя.

Расчет задачи с матрицей hood для любого рассмотренного числа процессоров быстрее всего происходил методом BJIIC-CG с применением (MPI+OpenMP)-подхода (см. рис. 2). Как видно из рис. 1—5, при небольшом числе процессоров быстрее всего происходят расчеты рассматриваемых задач методом BJIIC-CG с применением MPI+OpenMP.

Из рис. 1—4 видно, что там, где применение BIIC2-CG делает счет быстрее, чем IIC-CG и BJIIC-CG (без OpenMP), как правило, использование OpenMP в реализациях методов IIC-CG, BJIIC-CG позволяет по ним считать быстрее, чем по BIIC2-CG . Это не всегда так, когда имеется большое число процессоров и использование нитей в реализациях IIC-CG и BJIIC-CG либо не дает эффекта, либо этот эффект невелик.

Уменьшение эффекта от использования OpenMP-технологии с увеличением числа процессоров объясняется уменьшением числа строк матрицы, приходящихся на каждый процессор. Заметим, что в настоящей работе в качестве тестовых матриц использовались матрицы относительно небольшого размера. При расчетах реальных физических задач размер матриц, как правило, значительно больше. Следует ожидать, что потеря эффективности от использования OpenMP-технологии наступит при значительно большем числе процессоров.

На рис. 6, 7 представлены графики зависимости ускорения счета по сравнению со счетом на 8 процессорах с использованием только MPI-подхода для методов IIC-CG, BJIIC-CG при решении указанных выше задач. Как видно из рис. 6, 7, наблюдается хорошее ускорение счета с ростом числа процессоров. При не слишком большом числе процессоров использование (MPI+OpenMP)-подхода в расчетах указанных задач дает сверхлинейное ускорение. В ряде случаев сверхлинейное ускорение получено при использовании только MPI-подхода, что связано с изменением числа полученных в расчете итераций и, возможно, с особенностями разбиения на подобласти (так как сравнение происходит со временем счета на 8 процессорах).

Итак, в настоящей работе рассмотрены методы IIC-CG и BJIIC-CG. Предложен способ параллельной реализации методов IIC-CG и BJIIC с использованием (MPI+OpenMP)-подхода. Расчеты рассмотренных тестовых задач из коллекции университета Флориды показывают, что использование OpenMP-технологии при параллельной реализации методов позволяет значительно ускорить вычисления при решении задач с достаточно большими разреженными матрицами на умеренном числе процессоров.

Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (проекты 17-01-00973-а, 17-07-00510-а, 18-07-00841-а) и программы Президиума РАН №26 "Фундаментальные основы создания алгоритмов и программного обеспечения для перспективных высокопроизводительных вычислений".

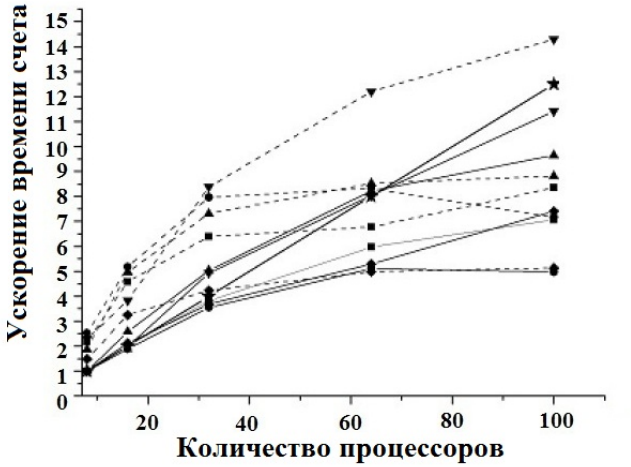

Рис. 6. Ускорения счета задач в случае использования метода IIC-CG: —■— — `m_t1`, MPI; --■-- — `m_t1`, MPI+OpenMP; —●— — `hood`, MPI; --●-- — `hood`, MPI+OpenMP; —▲— — `pwtk`, MPI; --▲-- — `pwtk`, MPI+OpenMP; —▼— — `msdoor`, MPI; --▼-- — `msdoor`, MPI+OpenMP; —♦— — `x104`, MPI; --♦-- — `x104`, MPI+OpenMP; —⋆— — идеальное ускорение

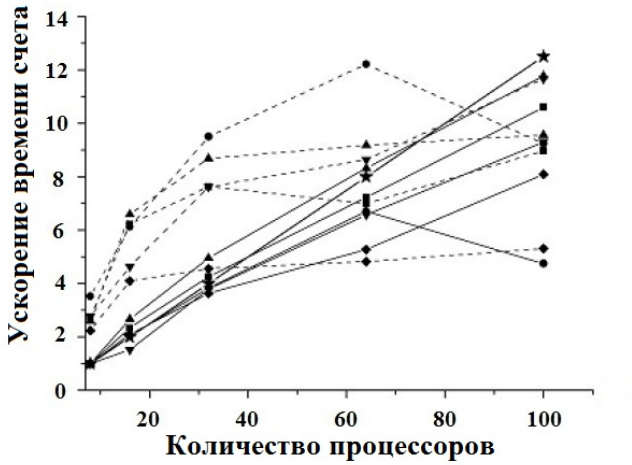

Рис. 7. Ускорения счета задач в случае использования метода BJIIC-CG: —■— — `m_t1`, MPI; --■-- — `m_t1`, MPI+OpenMP; —●— — `hood`, MPI; --●-- — `hood`, MPI+OpenMP; —▲— — `pwtk`, MPI; --▲-- — `pwtk`, MPI+OpenMP; —▼— — `msdoor`, MPI; --▼-- — `msdoor`, MPI+OpenMP; —♦— — `x104`, MPI; --♦-- — `x104`, MPI+OpenMP; —⋆— — идеальное ускорение

Список литературы

1. *Davis T., Hu Y. F.* University of Florida sparse matrix collection // ACM Trans. on Math. Software. 2011. Vol. 38, No 1. // http://www.cise.ufl.edu/research/sparse/matrices.
2. *Kaporin I. E.* Reordering and splitting of sparse matrices into overlapping blocks for massively parallel preconditioning of iterative methods // NUMGRID-2012, A. A. Dorodnicyn Computing Center RAS, Moscow, Dec. 17—18, 2012.
3. *Капорин И. Е., Милюкова О. Ю.* Неполное обратное треугольное разложение в параллельных алгоритмах предобусловленного метода сопряженных градиентов: Препринт № 37. М.: ИПМ им. М. В. Келдыша РАН, 2017. doi:10.20948/prepr-2017-37.
4. *Капорин И. Е., Милюкова О. Ю.* Массивно-параллельный алгоритм предобусловленного метода сопряженных градиентов для численного решения систем линейных алгебраических уравнений // Сб. трудов отдела проблем прикладной оптимизации ВЦ РАН / Под ред. В. Г. Жадана. М.: Изд-во ВЦ РАН, 2011. С. 132—157.
5. *Капорин И. Е.* Предобусловленный метод сопряженных градиентов для решения дискретных аналогов дифференциальных задач // Дифференц. ур-ния. 1990. Т. 26, № 7. С. 1225—1236.
6. *Kaporin I. E.* New convergence results and preconditioning strategies for the conjugate gradient method // Numer. Linear Algebra Appls. 1994. Vol. 1. P. 179—210.
7. *Kolotilina L. Yu., Yeremin A. Yu.* Factorized sparse approximate inverse preconditionings. I. Theory // SIAM J. Matrix Anal. Appl. 1993. Vol. 14. P. 45—58.
8. *Kaporin I.* Fast matrix scaling, fine-coarse grid partitioning, and highly parallel preconditioning // Numerical geometry, grid generation, and high performance computing // Proc. Int. Conf. NUMGRID-2010. Moscow, Oct. 11—13, 2010. M.: A. A. Dorodnicyn Computing Centre RAS, 2010. P. 99—106.
9. *Chow E.* A priori sparsity patterns for parallel sparse approximate inverse preconditioners // SIAM J. Sci. Comput. 2000. Vol. 21, No 5. P. 1804—1822.

10. *Chow E.* Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns // Int. J. High Performance Comput. Appl. 2001. Vol. 15, No 1. P. 56—74.

11. *Kolotilina L. Yu., Nikishin A. A., Yeremin A. Yu.* Factorized sparse approximate inverse preconditionings. IV. Simple approaches to rising effciency // Numer. Lin. Alg. Appl. 1999. Vol. 6. P. 515—531.

12. *Anzt H., Huckle T. K., Brackle J., Dongarra J.* Incomplete sparse approximate inverses for parallel preconditioning // Parallel Computing. 2018. Vol. 71. P. 1—22.

13. *Kaporin I. E.* High quality preconditionings of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$-decomposition // Numer. Lin. Alg. Appl. 1998. Vol. 5. P. 483—509.

14. *Axelsson O.* Iterative Solution Methods. New York: Cambridge Univ. Press, 1994.

15. *Капорин И. Е.* Предобусловливание итерационных методов решении систем линейных алгебраических уравнений: Автореф. дис. на соиск. ученой степени доктора физ.-мат. наук. М., 2011.

16. *Капорин И. Е.* Использование полиномов Чебышева и приближенного обратного треугольного разложения для предобусловливания метода сопряженных градиентов // Журнал вычисл. мат. и мат. физ. 2012. Т.52, № 2. С. 1—26.

17. *Капорин И. Е., Милюкова О. Ю.* MPI+OpenMP параллельная реализация метода сопряженных градиентов с некоторыми явными предобусловливателями: Препринт № 8. М.: ИПМ им. М. В. Келдыша РАН, 2018.

Статья поступила в редакцию 23.04.18.

MPI+OpenMP PARALLEL IMPLEMENTATION OF THE CONJUGATE GRADIENT METHOD WITH FACTORIZED EXPLICIT PRECONDITIONINGS / I. E. Kaporin, O. Yu. Milyukova (A. A. Dorodnitsyn's DPC of RAS, M.V. Keldysh's IAM of RAS, Moscow).

For the preconditioning of a symmetric positive definite sparse matrix, its approximate inverse is considered in the form of a product of an upper triangular sparse matrix by its transpose. Several choices of sparsity structure for the preconditioners are considered and analyzed. Parallel implementation of the corresponding linear solvers within the MPI+OpenMP framework is proposed. Comparative timing results are presented based on preconditioned conjugate gradient solution of test problems from the University of Florida sparse matrix collection using MPI, or MPI+OpenMP algorithms. The comparison also includes the point Jacobi preconditioning and the Block Incomplete Inverse preconditioning with the 2nd order Cholesky factorization within the blocks.

Keywords: sparse matrices, conjugate gradient method, explicit preconditioning, incomplete inverse triangular factorization, parallel computing.