ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 8 за 2018 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

Капорин И.Е., Милюкова О.Ю.

MPI+OpenMP параллельная реализация метода сопряженных градиентов с некоторыми явными предобусловливателями

Рекомендуемая форма библиографической ссылки: Капорин И.Е., Милюкова О.Ю. MPI+OpenMP параллельная реализация метода сопряженных градиентов с некоторыми явными предобусловливателями // Препринты ИПМ им. М.В.Келдыша. 2018. № 8. 28 с. doi:10.20948/prepr-2018-8
URL: http://library.keldysh.ru/preprint.asp?id=2018-8

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М. В. Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

И.Е. Капорин, О.Ю. Милюкова

MPI+OpenMP параллельная реализация метода сопряженных градиентов с некоторыми явными предобусловливателями

Москва — 2018

Капорин И.Е., Милюкова О.Ю.

MPI+OpenMP параллельная реализация метода сопряженных градиентов с некоторыми явными предобусловливателями

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 рассматриваемых методов с использованием MPI+OpenMP подхода. Проводится сравнение времени решения с использованием MPI и MPI+OpenMP технологии рассматриваемыми методами и методом сопряженных градиентов с предобусловливанием Якоби модельной задачи и тестовых задач из коллекции университета Флориды.

Ключевые слова: итерационное решение систем линейных алгебраических уравнений, разреженные матрицы, неполная обратная треугольная факторизация, параллельное предобусловливание, метод сопряженных градиентов

Igor Evgenevich Kaporin, Olga Yurievna Milyukova

MPI+OpenMP parallel implementation of explicitly preconditioned conjugate gradient method

A preconditioner for large sparse symmetric positive definite coefficient matrix is considered based on its approximate inverse in the form of product of a lower triangular sparse matrix by its transpose. A parallel algorithm for the construction and application of the preconditioner is proposed with the use of MPI+OpenMP techniques. Comparative timing results for the MPI+OpenMP and MPI implementations of the considered preconditioning and the Jacobi preconditioning used with the conjugate gradient method for a model problem and the University of Florida collection test problems is presented.

Keywords: iterative solution of linear systems, sparse matrices, incomplete inverse triangular factorization, parallel preconditioning, conjugate gradient method

Работа выполнена при финансовой поддержке РФФИ (номера проектов 17-01-00973-а, 17-07-00510-а, 18-07-00841-а)

1. Введение

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

$$Ax = b \tag{1.1}$$

с симметричной положительно определенной разреженной матрицей A общего вида

$$A = A^T > 0.$$

Проблема построения соответствующих эффективных численных методов 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. В качестве примера такой постановки приведем конечно-элементные модели пространственных задач вычислительной механики, явившиеся источником большей части использованных в настоящей работе тестовых матриц [1].

Настоящая работа выполнена в Институте прикладной математики имени М.В. Келдыша РАН и в Вычислительном центре им. А.А. Дородницына ФИЦ ИУ РАН.

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

$$\|b - Ax_k\| \le \varepsilon \|b - Ax_0\|, \quad 0 < \varepsilon << 1. \tag{1.2}$$

Решение задач с матрицами очень большого размера требует применения параллельных компьютеров с большим числом процессоров. Ниже будут рассмотрены итерационные методы решения задачи (1.1), 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 оказываются хорошо приспособлены к параллельной реализации с использованием MPI+OpenMP подхода.

В формуле (1.1), предполагается, что матрица A уже переупорядочена, а вместо A_P стоит A ($A = A_P = P\widehat{A}P^T$), где P – матрица перестановки, а $\widehat{A}$ представляет собой матрицу коэффициентов исходной задачи. В настоящей работе применяются переупорядочения, уменьшающие среднюю ширину ленты матрицы, а именно предложенные в работах [2], [3], являющиеся обобщением упорядочения [4]. Подход, предложенный в этих работах, позволяет одновременно произвести разбиение области расчета на подобласти.

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

$$\hat{G}A\hat{G}^T = I_n + E \qquad (1.3)$$

матрицы A (уже переупорядоченной), где $\hat{G}$ – разреженная нижняя треугольная матрица с положительными диагональными элементами, I_n – единичная матрица, E представляет собой матрицу погрешности. Соответствующий предобусловливатель имеет вид

$$H \equiv \hat{G}^T\hat{G} \approx A^{-1}, \qquad (1.4)$$

причем структура разреженности $\hat{G}$ задается как множество ненулевых позиций, определяемых специальным образом [8], описанным ниже. Другие способы выбора структуры расположения ненулевых элементов $\hat{G}$ обсуждались, в частности, в [9-11].

При построении предобусловливателя будем также предполагать, что матрица отмасштабирована, т. е. ее диагональные элементы равны единице. Это достигается с использованием формулы: $A_{SP} = D_{A_P}^{-1/2} A_P D_{A_P}^{-1/2}$, где D_{A_P} – диагональная часть матрицы A_P. Далее при описании предобусловливателей вместо A_{SP} будем использовать обозначение A, предполагая, что переупорядочение и масштабирование уже выполнены. Значения ненулевых элементов матрицы G на каждом этапе определяются из условия оптимизации K-числа обусловленности матрицы GAG^T (см. ниже формулу (2.1)). Такое предобусловливание сокращенно принято называть IIC (inverse incomplete Cholesky). При этом очевидно, что $\hat{G} = GD_{A_P}^{-1/2}$.

В настоящей работе рассматривается также метод предобусловливания блочного Якоби неполного обратного треугольного разложения (BJIIC), предложенный в работе [3]. В этом методе сначала находится предобусловливатель блочного Якоби CG, а затем для каждого блока строится неполное обратное треугольное разложение. При построении этого предобусловливателя и при его обращении не требуется обмен информацией между процессорами.

Аналогичный подход рассматривался в недавней статье [12] (см. также цитированную там литературу), где представлено предобусловливание, основанное на ILU-разложении матрицы коэффициентов с последующим построением приближенных обратных для соответствующих нижнего и верхнего треугольных сомножителей.

В настоящей работе описаны методы построения предобусловливателей IIC, BJIIC для переупорядоченной и отмасштабированной матрицы A. В работе предложены алгоритмы параллельной реализации построения и обращения предобусловливателей IIC, BJIIC с использованием MPI+OpenMP подхода. Приводятся результаты расчетов модельной задачи и ряда тестовых задач из коллекции университета Флориды [1]. Проводится сравнение времени решения задач с использованием MPI и MPI+OpenMP рассматриваемыми

методами (IIC-CG, BJIIC-CG) и методом сопряженных градиентов с предобусловливанием Якоби (J-CG) .

2. Предобусловленный метод сопряженных градиентов

Пусть требуется решить СЛАУ (1.1), и для матрицы A ($A = A_P$) построена приближенная обратная вида $H \equiv \hat{G}^T \hat{G}$. Алгоритм предобусловленного метода CG (см., например, [13]) имеет следующий вид:

$r_0 = b - Ax_0$, $p_0 = w_0 = Hr_0$, $\gamma_0 = r_0^T p_0$,

для k=0,… пока $(r_k^T r_k) \le \varepsilon^2 (r_0^T r_0)$ выполнять

$q_k = Ap_k$,

$\alpha_k = \gamma_k / (p_k^T q_k)$,

$x_{k+1} = x_k + \alpha_k p_k$,

$r_{k+1} = r_k - \alpha_k q_k$,

$z_{k+1} = Hr_{k+1}$,

$\gamma_{k+1} = r_{k+1}^T z_{k+1}$,

$\beta_k = \gamma_{k+1} / \gamma_k$,

$p_{k+1} = z_{k+1} + \beta_k p_{k+1}$,

где $0 < \varepsilon << 1$. С учетом соотношения (1.4), очевидно, что этот алгоритм использует лишь операции умножения разреженных матриц на вектор, операции вычисления скалярных произведений и элементарные векторные операции. Поэтому принципиальная возможность его эффективной параллельной реализации не вызывает сомнений, даже при использовании большого числа процессоров и (или) применения OpenMP технологии.

В [6, 14] доказана оценка сходимости предобусловленного метода CG:

$$\|r_k\|_H \le \left(K(HA)^{1/k} - 1\right)^{k/2} \|r_0\|_H ,$$

где

$$K(B) = (n^{-1} trace B)^n / \det B \qquad (2.1)$$

представляет собой K-число обусловленности симметричной матрицы $B = \hat{G} A \hat{G}^T$. При этом для числа итераций метода CG справедлива следующая (упрощенная) верхняя граница числа итераций:

$$k_{\varepsilon_1} \le \log_2 K(HA) + \log_2 (\varepsilon_1^{-1}), \qquad (2.2)$$

где $0 < \varepsilon_1 << 1$ задает требуемое уменьшение H нормы невязки. Заметим, что в силу формул $A_{SP} = D_{A_P}^{-1/2} A_P D_{A_P}^{-1/2}$ и $\hat{G} = G D_{A_P}^{-1/2}$ верно: $K(HA_P) = K(H_s A_{SP})$, где $H_s = G^T G$.

3. Выбор значений ненулевых элементов матрицы *G*, обеспечивающий построение *K*-оптимального предобусловливания, и выбор позиций ненулевых элементов матрицы *G*

Предположим, что матрица *A* переупорядочена и отмасштабирована. Опишем алгоритм [15] построения нижнетреугольной невырожденной матрицы *G*, оптимальной в смысле минимизации величины $K(H_s A) = K(GAG^T)$. Допустим, что заранее заданы позиции структурно ненулевых элементов *i*-й строки нижнетреугольной матрицы *G:*

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

где

$$1 \le j_i(1) < ... < j_i(m_i) = i$$

представляет собой *i*-й список соответствующих столбцовых индексов. Тогда, как доказано в [15], минимум *K*-числа обусловленности $K(HA)$ (индекс *s* у H_s опущен), определенного выше соотношением (2.1), достигается при

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

Здесь S_i представляет собой главную подматрицу размера m_i матрицы *A*, построенную на указанном *i*-м подмножестве столбцовых индексов, то есть

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

Пусть 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 векторов z_i, выражающая их через симметричную треугольную факторизацию [15]

$$S_i = L_i L_i^T$$

в виде

$$z_i = L_i^{-T} v_i,$$

где вектор v_i длины m_i имеет вид: $v_i = (0,...,0,1)^T$. Таким образом, для каждого *i* достаточно вычислить треугольное разложение $S_i = L_i L_i^T$ матрицы порядка m_i и затем решить одну треугольную систему $L_i^T z_i = v_i$ того же порядка.

Для выбора позиций ненулевых элементов матрицы *G* будем использовать способ [8] (см. параграф 3.2), 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 $\tilde{G}$, как описано выше.

На втором этапе осуществляется прореживание множества позиций ненулевых элементов построенной матрицы $\tilde{G}$: отбрасываются позиции, в которых $0 < \left|g_{ij}\right| \le \tau_0 g_{ii}$, где $j < i$, g_{ij} – элементы матрицы $\tilde{G}$, $0 < \tau_0 << 1$. Затем снова

строится матрица G, как описано выше, в которой используется полученное прореженное множество ненулевых позиций.

Показатель степени q подбирается так, чтобы время решения СЛАУ (1.1) предобусловленным методом CG в сумме с временем построения предобусловливателя было минимально. Заметим, что для достаточно плотных матриц выбор $q=1$ обычно приводит к хорошим результатам.

4. Алгоритм параллельной реализации

Пусть матрица A переупорядочена и разбита на блоки, причем на блочной диагонали расположены p квадратных блоков размера $n_s \times n_s$, $1 \le s \le p$. Обозначим $k_s = n_1 + ... + n_s$.

В каждом процессоре с номером s =1,…,p будем строить матрицу G_s, содержащую соответствующие n_s строк матрицы G. Сначала в каждом процессоре с номером s =1,…,p создадим матрицы $\overline{S}_s$ размера $m_s \times m_s$, где $m_s \ge n_s$, которые будут содержать все элементы матрицы A, необходимые для построения всех строк матрицы $\tilde{G}$ с номерами $k_{s-1}+1 \le i \le k_s$. Для этого нужно осуществить соответствующие пересылки. Заметим, что матрица $\overline{S}_s$ может содержать элементы матрицы A с номерами строк $i \le k_{s-1}$, содержит строки с номерами $k_{s-1}+1 \le i \le k_s$. Затем производится масштабирование матриц $\overline{S}_s$, что осуществляется всеми процессорами одновременно и независимо. При этом находятся элементы диагональной матрицы $D_{A_p,s}$, которые являются соответствующими элементами матрицы D_{A_p}.

В каждом процессоре при построении матрицы G_s используется алгоритм, аналогичный алгоритму при построении матрицы G (см. раздел 3), только вместо матрицы A используется отмасштабированная матрица $\overline{S}_s$. При этом для всех строк внутри каждой подобласти все вычисления происходят независимо с применением OpenMP технологии. Для распределения вычисления значений элементов матрицы G_s по строкам (итерациям цикла с номерами строк) используется директива **do** с опцией **schedule static** (задающей, каким образом итерации цикла распределяются по нитям)**.** При этом все множество итераций делится на непрерывные куски примерно одинакового размера, и полученные порции итераций распределяются между нитями. Затем во всех процессорах независимо и одновременно вычисляется матрица $\hat{G}_s = G_s D_{A_p,s}^{-1/2}$.

Параллельная реализации вычислений $p = Hr = \hat{G}^T \hat{G} r$ происходит следующим образом. Сначала производится пересылка из процессоров с меньшими номерами в процессоры с большими номерами значений r_m (здесь $m \le k_{s-1}$ – индекс компоненты вектора r), необходимых для вычисления $\hat{z}_s = \hat{G}_s r$

в каждом процессоре с номером *s*. Затем в процессоре с номером *s* ($1 \le s \le p$) вычисляются $\hat{z}_s = \hat{G}_s r$ и соответствующие слагаемые $p_l = \hat{G}_s^T \hat{z}_s$ ($l \le s$, $l-$ номер процессора, содержащего соответствующую компоненту вектора *p*).

Как показали расчеты, использование для вычисления $p = \hat{G}^T \hat{G} r$ алгоритма 1, который эффективно применялся при распараллеливании с использованием только MPI, часто оказывается неэффективно при применении MPI+OpenMP технологии.

Алгоритм 1

```
For i=ni,1,-1
      w=0.0
   For l=ig(i)+1, ig(i+1)
       w=w+G(l)*r(ja(l))
    End for
    For l=ig(i)+1, ig(i+1)
       p(jg(l)) = p(jg(l)) + G(l)*w
    End for
 End for
```

В алгоритме 1 и далее массивы ig(ni+1), jg(ig(ni+1)), G(ig(ni+1)) определяют матрицу $\hat{G}_s$ в CRS формате [1], ni – количество строк в матрице $\hat{G}_s$. Поэтому вычисление элементов вектора *p* будем проводить в 2 этапа: $\hat{z}_s = \hat{G}_s r$ и $p_l = \hat{G}_s^T \hat{z}_s$, используя

Алгоритм 2

```
For i=ni,1,-1
      w(i)=0.0
End  for
For i=mi,1,-1
      p(i)=0.0
End  for

For i=ni,1,-1
   For l=ig(i)+1, ig(i+1)
       w(i)=w(i)+G(l)*r(jg(l))
    End for
End for
For i=mi,1,-1
    For l=igt(i)+1, igt(i+1)
       p(i) = p(i) + GT(l)*w(jgt(l))
    End for
 End for
```

В алгоритме 2 массивы igt(ni+1), jgt(ig(ni+1)), GT(ig(ni+1)) определяют матрицу $\hat{G}_s^T$ в CRS формате. Формирование матрицы $\hat{G}_s^T$ осуществляется до применения предобусловленного метода CG к решению СЛАУ (1.1). Как показывают

расчеты задач, приведенных в разделе 7, этот процесс занимает ничтожно малое время по сравнению с временем вычисления предобусловливателя и выполнением решения СЛАУ предобусловленным методом CG.

Для распределения вычисления значений элементов $\hat{z}(i)$ и $p(i)$ по строкам i (итерациям цикла с номером строк) будем в обоих случаях использовать директиву **do** с опцией **schedule static.**

Далее, после выполнения необходимых пересылок (из процессоров с большими номерами в процессоры с меньшими номерами), в каждом процессоре с номером s производится суммирование вида $p = p_s + \sum_{l>s} p_l$, где $p_l -$ нужная часть вектора p, которая вычисляется в процессоре с номером l при выполнении в нем своей доли вычислений для получения $p = \hat{G}^T(\hat{G}r)$.

Рассмотрим, как осуществляется вычисление $q = Ap$. Матрица A хранится в памяти в распределенном CRS-формате [1] (содержит как верхний, так и нижний треугольник). Сначала производится пересылка значений p_m, необходимых для вычисления $q_i = \sum_{m=1}^{N} a_{im} p_m$ (где a_{im} – элементы матрицы A) в рассматриваемом процессоре и хранящихся в памяти других процессоров. Вычисления $q_i = \sum_{m=1}^{N} a_{im} p_m$ в каждом процессоре для своей группы индексов i осуществляется с применением OpenMP технологии. Для распределения вычисления значений элементов q_i по строкам i будем использовать директиву **do** с опцией **schedule static.** Заметим, что, как показали расчеты задач, приведенных в разделе 7, методом CG с предобусловливанием Якоби (J-CG), использование параметров **dynamic**; **guided**; **guided, 6** в этой опции в большинстве случаев не позволяет ускорить вычисления по сравнению с использованием опции **static.** Параллельная MPI реализация вычислений векторных операций и скалярных произведений хорошо известна. Для вычислений векторных операций и частичных сумм в скалярных произведениях применялась OpenMP технология. Использовалась директива **do** с опцией **schedule static.**

5. 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 размера $n_s \times n_s$, $1 \le s \le p$. Обозначим, как и ранее, $k_s = n_1 + ... + n_s$. Определим прямоугольные матрицы

$$W_s = \left\lfloor e_{k_{s-1}+1} \middle| ... \middle| e_{k_s} \right\rfloor,$$

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

$$H = \sum_{s=1}^{p} W_s D_{A_s}^{-1/2} \overline{G}_s^T \overline{G}_s D_{A_s}^{-1/2} W_s^T \ . \qquad (6.1)$$

Предобусловливание (6.1) ранее предложено в [3] и названо блочное Якоби неполное обратное треугольное разложение (BJIIC). Вычисление элементов матриц $\overline{G}_s$ (*s*=1,…,*p*) осуществляется аналогично описанному в разделе 3, вместо отмасштабированной матрицы A используются матрицы A_{0s}. При вычислении матрицы $\overline{G}_s$ в каждом процессоре с номером *s*=1,…,*p* не требуется информации, хранящейся в других процессорах. В процессе вычисления матрицы $\overline{G}_s$ все процессоры могут работать одновременно и независимо, пересылок не требуется. Кроме того, каждая строка матрицы $\overline{G}_s$ вычисляется независимо от других строк этой матрицы с применением Open MP технологии аналогично описанному в разделе 4. Масштабирование матрицы $\overline{G}_s$ по формуле $\breve{G}_s = G_s D_{A_s}^{-1/2}$ во всех процессорах осуществляется одновременно. Затем во всех процессорах формируются матрицы $\breve{G}_s^T$, что занимает, как показывают расчеты, ничтожно малое время.

При выполнении операции

$$p = Hr = \breve{G}_s^T (\breve{G}_s r)$$

тоже все процессоры могут работать одновременно и независимо, пересылок не требуется. При этом применяется OpenMP технология аналогично описанному в разделе 4. Параллельная реализация остальных этапов алгоритма предобусловленного метода CG приведена в разделе 4.

6. Оценки уменьшения K-числа обусловленности в методах BJIIC и IIC

Мы предполагаем, что матрица *A* предварительно переупорядочена и отмасштабирована (см. выше). В работе [3] получены оценки

$$\frac{K(G^T G A)}{K(A)} \le \exp\left(-\frac{\|I - A\|_F^2}{2\|A\|} \right)$$

для метода IIC, и

$$\frac{K(\overline{G}^T \overline{G} A)}{K(A)} \le \exp\left(-\frac{\|I - A_0\|_F^2}{2\|A\|} \right) = \exp\left(-\frac{\|I - A\|_F^2}{2\|A\|} \right) \exp\left(\frac{\|A - A_0\|_F^2}{2\|A\|} \right)$$

для метода BJIIC, где $\|*\|_F$ обозначает фробениусову норму матрицы. Из последних двух формул видно, что оценка числа итераций (см. (2.2)) в обоих методах меньше, чем в методе J-CG. За ухудшение оценки BJIIC по сравнению с IIC отвечает величина фробениусовой нормы $\|A - A_0\|_F$ блочно-внедиагональной части матрицы A. Таким образом, если матрица A несильно отличается от A_0, можно ожидать, что правые части этих оценок также отличаются не очень сильно.

Матрица A_0 близка к матрице A при достаточно хорошем разбиении, когда вне блочной диагонали оказывается не слишком много ненулевых элементов A, см, например [3]. Естественно, на величину $\|A - A_0\|_F$ оказывают влияние также абсолютные значения элементов блочно-внедиагональной части матрицы A.

Так, например, для разностной задачи Дирихле для уравнения Пуассона, когда разбиение области расчета происходит на квадратные или кубические подобласти, получаем $K(\overline{G}^T \overline{G} A) > K(G^T G A)$, и оценка числа итераций в методе BJIIC-CG хуже, чем в методе IIC-CG.

7. Результаты расчетов

Все программы, реализующие применение методов IIC-CG, BJIIC-CG, J-CG для решения системы уравнений $Ax = b$, где $A = A^T > 0$, были написаны на языке FORTRAN с использованием MPI+OpenMP подхода, расчеты производились на многопроцессорной вычислительной системе МВС-10П, установленной на МСЦ РАН. Матрица A хранилась в распределенном CRS-формате.

Тестирование и сравнение методов производилось с помощью расчетов модельной задачи – разностной задачи Дирихле для уравнения Пуассона в единичном квадрате на ортогональной сетке, причем n=1048576. Использовалась стандартная 5-точечная аппроксимация лапласиана (имя матрицы 5_1048576). Для тестирования рассматриваемых параллельных методов использовались также некоторые матрицы из коллекции университета Флориды [1]. Перечислим имена используемых тестовых матриц и укажем источник 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 таблице 1 указаны некоторые свойства этих матриц, причем значения $Cond(A_S)$, где $A_S = (D_A)^{-1/2} A (D_A)^{-1/2}$ – матрица системы уравнений после масштабирования, взяты из работы [15], 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.

Таблица 1

Свойства некоторых матриц из коллекции университета Флориды

Матрица	N	NZA	Id	Ip	$nz_{\min}$	$nz_{\max}$	$Cond(A_S)$
m_t1	97578	9753570	1	4648398	48	237	0.47+10
hood	220542	9895422	9910	4879422	1	77	0.55+6
pwtk	217918	11524432	325	5407348	2	180	0.26+9
mdoor	415863	19173163	11125	9350756	1	77	0.19+9
x104	108384	8713602	2255	4059880	8	270	0.1+11

Решалось уравнение $Ax = b$, где правая часть $b_i \equiv 1$, начальное приближение $x_0 \equiv 0$, счет продолжался до выполнения условия (1.2), где $\varepsilon = 10^{-8}$. Для разбиения области расчета при решении всех задач использовался способ [3]. При построении предобусловливателей IIC во всех задачах, кроме модельной, использовались значения параметров $\tau_0 = 0.01$, $q = 1$, в модельной задаче $\tau_0 = 0.01$, $q = 2$. Такой выбор q был продиктован минимизацией времени счета задачи (без использования OpenMP). При построении предобусловливателя BJIIC во всех задачах использовались значения параметров $\tau_0 = 0.01$, $q = 1$.

Каждый вычислительный модуль решающего поля на многопроцессорной вычислительной системе МВС-10П имеет в своем составе 2 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 (1.1) методом J-CG для модельной задачи и задачи с матрицей **m_t1** происходят быстрее всего, если в каждом вычислительном модуле решающего поля использовать 2 процессора Xeon E5-2690 и отказаться от использования сопроцессоров Intel Xeon Phi 7110X. Поэтому решение всех задач методами J-CG, IIC-CG, BJIIC-CG на вычислительной системе МВС-10П производились с использованием в каждом вычислительном модуле только 2 процессоров Xeon E5-2690.

В таблицах 2, 3 приведены результаты расчетов методом J-CG различных задач на разных числах процессоров p. Во всех таблицах P=p.

В таблице 2 приведены числа итераций и времена счета задач методом J-CG без использования OpenMP технологии, времена счета задач при использовании Th=4, Th=8, Th=16 нитей OpenMP. В таблице 3 приведены числа итераций и времена счета задач методом J-CG безиспользования OpenMP технологии, минимальные времена счета при использовании OpenMP технологии, как описано выше, ускорения счета Sp_th благодаря использованию нитей, Sp_pr – ускорения счета для разного числа

Таблица 2

Числа итераций и времена счета различных задач методом J-CG на p процессорах без использования и с использованием OpenMP технологии

P	8	16	32	64	100
5_1048576	It=1898 2.76	It=1898 1.65	It=1898 0.81	It=1898 0.51	It=1898 0.40
Th=4	1.26	0.74	1.56	1.93	2.43
Th=8	0.62	1.49	1.38	2.64	2.51
Th=16	0.89	1.26	1.92	2.74	2.69
m_t1	It=536545 540.86	It=536667 282.03	It=534807 162.03	It=536474 95.52	It= 537466 76.58
Th=4	132.54	83.17	70.09	53.6	73.94
Th=8	125.41	82.40	72.58	92.38	96.28
Th=16	125.15	83.13	78.52	89.62	93.43
hood	It=6078 7.04	It=6056 3.87	It=6052 2.14	It=6061 1.36	It=6059 1.19
Th=4	2.79	2.26	2.94	3.71	4.04
Th=8	2.74	1.84	2.93	2.57	3.55
Th=16	2.17	1.74	2.42	3.98	5.25
pwtk	It=74111 134.58	It=74065 48.43	It=74119 26.21	It=74692 16.14	It=74776 12.71
Th=4	37.76	12.69	11.98	12.07	15.04
Th=8	38.04	13.07	10.54	10.66	15.78
Th=16	39.00	12.7	10.94	12.41	19.85
msdoor	It=83510 283.89	It=83496 111.12	It=83531 53.59	It=83493 32.52	It=83466 21.71
Th=4	94.66	25.12	17.81	15.15	17.04
Th=8	92.2	24.45	17.44	18.84	16.01
Th=16	94.47	24.14	17.51	14.76	24.57
x104	It=884621 894.61	It=888104 444.99	It=882639 257.20	It=884943 187.28	It=882803 123.5
Th=4	193.72	140.06	113.16	118.65	121.58
Th=8	192.77	140.39	112.48	129.3	130.33
Th=16	194.57	141.43	113.98	130.7	135.48

процессоров по сравнению со счетом на 8 процессорах в случае расчета без использования OpenMP технологии, Speedup – ускорения счета благодаря использованию MPI+OpenMP подхода по сравнению со счетом на 8 процессорах без использования OpenMP технологии.

Таблица 3

Числа итераций и времена счета различных задач методом J-CG на разном числе процессоров без использования и с использованием OpenMP технологии, ускорения счета благодаря использованию OpenMP технологии, ускорения счета по сравнению с расчетом на 8 процессорах

p p/8	8 1	16 2	32 4	64 8	100 12.5
5_1048576 Sp_th Speedup Sp_pr	It=1898 2.7 Th=8 0.62 4.45 4.45 1	It=1898 1.65 Th=4 0.74 2.22 3.72 1.67	It=1898 0.81 Th=8 1.38 Sp_th <1 3.4	It=1898 0.51 Th=4 1.93 Sp_th < 1 5.41	It=1898 0.4 Th=4 2.43 Sp_th < 1 6.9
m_t1 Sp_th Speedup Sp_pr	It=536545 540.86 Th=16 25.15 4.32 4.32 1	It=536667 282.03 Th=8 82.40 3.42 6.56 1.91	It=534807 162.03 Th=4 70.09 2.31 7.71 4.25	It=536474 95.52 Th=4 53.6 1.78 10.09 5.66	It= 537466 76.58 Th=4 73.94 1.03 7.3 7.06
hood Sp_th Speedup Sp_pr	It=6078 7.04 Th=16 2.17 3.24 3.24 1	It=6056 3.87 Th=16 1.74 2.22 4.04 1.82	It=6052 2.14 Th=16 2.42 Sp_th <1 3.28	It=6061 1.3 Th=8 2.57 Sp_th <1 5.17	It=6059 1.19 Th=8 3.55 Sp_th <1 5.91
pwtk Sp_th Speedup Sp_pr	It=74111 134.58 Th=4 37.76 3.56 3.56 1	It=74065 48.43 Th=4 12.69 3.81 10.6 2.77	It=74119 26.21 Th=8 10.54 2.48 12.76 5.13	It=74692 16.14 Th=8 10.66 1.51 12.62 8.33	It=74776 12.71 Th=4 15.04 Sp_th < 1 13.35
mdoor Sp_th Speedup Sp_pr	It=83510 283.89 Th=8 92.2 3.07 3.07 1	It=83496 111.12 Th=16 24.14 4.6 11.7 2.5	It=83531 53.59 Th=8 17.44 3.07 16.28 5.29	It=83493 32.52 Th=16 14.76 2.2 9.23 8.72	It=83466 21.71 Th=8 16.01 1.35 17.73 13.05
x104 Sp_th Speedup Sp_pr	It=884621 894.61 Th=8 192.77 4.64 4.64 1	It=888104 444.99 Th=4 140.06 3.17 6.39 2.01	It=882639 257.20 Th=8 112.48 2.29 7.95 3.48	It=884943 187.28 Th=4 118.65 1.58 7.53 4.78	It=882803 123.5 Th=4 121.58 1.02 7.35 7.35

В таблицах 4-9 приведены числа итераций и времена счета различных задач методом IIC-CG на разных числах процессоров p без использования OpenMP технологии (первый столбец), времена счета задач при использовании Th=4, Th=8, Th=16 нитей OpenMP. В первом столбце приведены числа итераций, затем времена вычисления матрицы $\hat{G}_s$, времена счета итерационного процесса решения СЛАУ методом IIC-CG и общее время. В остальных столбцах – времена вычисления матриц $\hat{G}_s$, времена вычисления матриц $\hat{G}_s^T$, времена счета итерационного процесса в методе IIC-CG и общее время.

Таблица 4

Числа итераций и времена счета модельной задачи
методом IIC-CG на разном числе процессоров
без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=1211	0.2 0.007	0.16 0.007	0.21 0.008
	0.37,4.99, 5.36	3.70 3.90	3.43 3.6	3.43 3.63
16	It=1175	0.11 0.004	0.12 0.004	0.15 0.004
	0.2, 2.5, 2.53	2.19 2.31	2.08 2.21	1.95 2.11
32	It=1201	0.06 0.002	0.05 0.002	0.24 0.001
	0.09,1.24, 1.34	1.07 1.13	0.90 0.96	1.43 1.643
64	It=1193	0.04 0.001	0.04 0.002	0.02 0.001
	0.05,0.64, 0.70	0.71 0.76	0.66 0.70	1.75 2.00
100	It=1161	0.01 0.001	0.03 0.001	0.25 0.008
	0.05,0.51, 0.56	0.66 0.69	0.6 0.64	1.69 1.95

Таблица 5

Числа итераций и времена счета задачи с матрицей **m_t1** методом IIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
	It=3053	2.58 0.003	1.49 0.004	1.22 0.004
8	6.9,7.81, 14.71	5.63 8.22	5.34 6.84	5.52 6.81
	It=3069	1.51 0.004	0.98 0.002	0.91 0.002
16	3.96,3.30, 7.27	2.91 4.43	2.22 3.21	2.67 3.58
	It=3044	0.89 0.001	0.61 0.001	0.50 0.001
32	2.0, 1.78 3.85	1.89 2.78	1.68 2.30	1.97 2.48
	It=2984	0.67 0.001	0.50 0.001	0.68 0.001
64	1.48,0.97, 2.46	1.49 2.17	1.77 2.27	2.54 3.23
	It=2938	0.66 0.001	0.5 0.001	0.56 0.001
100	1.34,0.74, 2.08	1.3 1.97	1.26 1.76	2.97 3.53

Таблица 6

Числа итераций и времена счета задачи с матрицей **hood** методом IIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=424	1.09 0.004	0.67 0.004	0.53 0.001
	2.33,1.24, 3.58	0.99 2.08	0.74 1.42	1.01 1.56
16	It=409	0.58 0.002	0.37 0.002	0.32 0.002
	0 .41,0.54, 1.90	0.43 1.01	0.33 0.70	0.36 0.69
32	It=415	0.3 0.002	0.21 0.002	0.22 0.002
	0.70,0.30, 1.01	0.32 0.62	0.24 0.45	0.27 0.50
64	It=421	0.29 0.001	0.23 0.001	0.36 0.001
	0.53,0.17, 0.70	0.21 0.51	0.19 0.43	0.36 0.72
100	It=415	0.35 0.001	0.3 0.001	0.44 0.001
	0.57,0.15, 0.72	0.19 0.54	0.19 0.50	0.47 0.92

Таблица 7

Числа итераций и времена счета задачи с матрицей **pwtk** методом IIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
	It=3696	1.15 0.005	0.75 0.005	0.78 0.003
8	2.96,13.06,16.03	9.31 10.47	7.80 8.54	9.23 10.02
	It=3571	0.58 0.002	0.38 0.002	0.39 0.003
16	1.45,4.74, 6.19	3.60 4.18	2.85 3.23	3.28 3.68
	It=3600	0.3 0 .002	0.21 0.002	0.34 0.002
32	0.72,2.46, 3.18	2.22 2.53	1.98 2.19	3.14 3.48
	It=3581	0.30 0.001	0.25 0.002	0.44 0.003
64	0.51, 1.44, 1.95	1.84 2.15	1.63 1.88	3.53 3.97
	It=3543	0.36 0.001	0.31 0.001	0.52 0.002
100	0.60,1.06, 1.66	1.6 1.96	1.50 1.82	2.6 3.13

Заметим, что дальнейшее увеличение числа используемых процессоров может привести к ухудшению масштабируемости методов в случае рассматриваемых задач.

Таблица 8

Числа итераций и времена счета задачи с матрицей **msdoor** методом IIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=3934 5.05,25.55,30.60	1.93 0.01 14.7 16.65	1.17 0.01 12.19 13.37	0.94 0.01 13.6 14.56
16	It=3889 2.73,12.74,15.48	0.98 0.005 8.53 9.52	0.62 0.004 7.36 7.98	0.50 0.006 7.69 8.21
32	It=3854 1.27,4.93, 6.20	0.63 0.003 3.97 4.61	0.37 0.003 3.27 3.65	0.46 0.007 3.98 4.45
64	It=3881 0.89,2.92, 3.81	0.42 0.002 2.48 2.90	0.32 0.002 2.18 2.50	0.48 0.004 3.96 4.45
100	It=3800 0.77,1.9, 2.68	0.45 0.002 2.01 2.46	0.37 0.002 1.77 2.14	0.61 0.01 4.15 4.77

Таблица 9

Числа итераций и времена счета задачи с матрицей **x104** методом IIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=8355 4.03,15.75,19.78	1.64,14.37,16.01	1.06,12.76, 13.83	0.79,12.61,13.41
16	It=8326 2.11,7.19, 9.31	0.83, 6.39, 7.23	0.55, 5.52, 6.08	0.53, 6.02, 6.55
32	It=8057 1.32,4.05, 5.37	0.49, 4.77, 5.26	0.35, 4.32, 4.67	0.32, 5.59, 5.51
64	It=7928 0.98,2.75 3.74	0.49, 3.77, 4.27	0.40, 3.57, 3.97	0.56, 5.24, 5.81
100	It=7775 0.86,1.81, 2.67	0.57, 3.41, 3.98	0.49, 3.35, 3.85	0.67, 4.96, 5.64

В таблице 10 приведены числа итераций и времена счета методом IIC-CG без использования OpenMP технологии, минимальные времена счета при использовании OpenMP технологий, как описано выше, ускорения счета Sp_th благодаря использованию нитей OpenMP, Sp_pr – ускорения счета для разного числа процессоров по сравнению с 8 процессорами в случае расчетов без использования OpenMP технологии, Speedup – ускорения счета благодаря использованию MPI и OpenMP технологии по сравнению с расчетами на 8 процессорах без использования OpenMP технологии.

Таблица 10

Числа итераций и времена счета методом IIC-CG различных задач на разном числе процессоров без использования и с использованием OpenMP технологии, ускорения счета благодаря использованию OpenMP технологии, ускорения счета по сравнению с расчетом на 8 процессорах

p p/8	8 1	16 2	32 4	64 8	100 12.5
5_1048576	It=1211 5.36 Th=8 3.60	It=1175 2.53 Th=16 2.11	It=1201 1.34 Th=8 0.96	It=1199 0.70 Th=8 0.70	It=1161 0.56 Th=8 0.64
Sp_th	1.48	1.2	1.39	1	Sp_th<1
Speedup	1.48	2.54	5.58	7.65	
Sp_pr	1	2.12	4	7.65	9.57
m_t1	It==3053 14.71	It=3069 7.27	It=3044 3.85	It=2984 2.46	It=2938 2.08
	Th=16 6.81	Th=8 3.21	Th=8 2.30	Th=4 2.17	Th=8 1.76
Sp_th	2.16	2.26	1.67	1.13	1.18
Speedup	2.16	4.58	6.39	6.77	8.35
Sp_pr	1	2.02	3.82	5.97	7.07
hood	It=424 3.58 Th=8 1.42	It=409 1.90 Th=16 0.69	It=415 1.01 Th=8 0.45	It=421 0.70 Th=8 0.43	It=415 0.72 Th=8 0.50
Sp_th	2.52	2.75	2.24	1.63	1.44
Speedup	2.52	5.18	7.96	8.32	7.16
Sp_pr	1	1.88	3.54	5.11	4.97
pwtk	It=3696 16.03	It=3571 6.19	It=3600 3.18	It=3581 1.95	It=3543 1.66
	Th=8 8.54	Th=8 3.23	Th=8 2.19	Th=8 1.88	Th=8 1.82
Sp_th	1.87	1.91	1.45	1.04	Sp_th<1
Speedup	1.87	4.96	7.31	8.52	
Sp_pr	1	2.58	5.04	8.22	9.65
msdoor	It=3934 30.60	It=3889 15.48	It=3854 6.20	It=3881 3.81	It=3800 2.68
	Th=8 13.37	Th=8 7.98	Th=8 3.65	Th=8 2.50	Th=8 2.14
Sp_th	2.28	1.93	1.7	1.52	1.25
Speedup	2.28	3.83	8.38	12.2	14.3
Sp_pr	1	1.98	4.93	8.03	11.4105
x104	It=8355 19.78	It=8326 9.31	It==8057 5.37	It=7928 3.74	It=7775 2.67
	Th1=16 13.4	Th=8 6.08	Th=8 4.67	Th=8 3.97	Th=8 3.85
Sp_th	1.48	1.53	1.14	Sp_t <1	Sp_t <1
Speedup	1.48	3.25	4.23		
Sp_pr	1	2.12	3.68	5.29	7.4

В таблицах 11-16 приведены числа итераций и времена счета различных задач методом BJIIC-CG на разных числах процессоров p без использования OpenMP технологии (первый столбец), времена их счета при использовании Th=4, Th=8, Th=16 OpenMP нитей. В первом столбце приведены числа итераций, затем времена вычисления матрицы $\breve{G}_s$, времена итерационного процесса решения СЛАУ методом BJIIC-CG и общее время. В остальных столбцах – времена вычисления матриц $\breve{G}_s$, времена вычисления $\breve{G}_s^T$, времена счета итерационного процесса в методе BJIIC-CG и общее время.

Таблица 11

Числа итераций и времена счета модельной задачи методом BJIIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=1824 0.09,5.27, 5.36	0.07 0.004 3.33 3.40	0.6 0.004 2.46 2.52	0.09 0.005 2.79 2.89
16	It=1777 0.05,2.92, 2.97	0.04 0.002 1.84 1.89	0.4 0.002 1.46 1.51	0.17 0.003 1.57 1.76
32	It=1953 0.02,1.55 1.57	0.03 0.001 1.23 1.26	0.21 0.001 1.19 1.41	0.20 0.001 1.43 1.64
64	It=1959 0.018,0.90, 0.92	0.02 0.001 1.01 1.04	0.16 0.001 0.89 1.05	0.02 0.001 1.68 1.89
100	It=1926 0.02,0.66, 0.68	0.03 0.001 0.86 0.90	0.21 0.001 1.4 1.61	0.21 0.002 2.24 2.46

Таблица 12

Числа итераций, времена счета задачи с матрицей **m_t1** методом BJIIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=3335 0.79, 8.45, 1.25	2.17 0.004 4.87 7.05	1.27 0.007 4.28 5.56	1.1 0.004 4.65 5.77
16	It=3296 3.12, 3.36, 6.49	1.11 0.002 2.11 3.32	0.68 0.002 1.94 2.63	0.53 0.002 1.92 2.45
32	It=3439 1.64, 1.93, 3.58	0.59 0.001 1.97 2.56	0.40 0.001 1.67 2.07	0.20 0.003 3.22 3.57
64	It=3649 0.93, 1.17, 2.11	0.34 0.002 1.85 2.19	0.23 0.002 2.26 2.50	0.34 0.002 3.43 3.78
100	It=3769 0.54,0.89, 1.44	0.23 0.001 1.49 1.72	0.17 0.001 1.53 1.70	0.42 0.002 3.57 4.00

Таблица 13

Числа итераций и времена счета задачи с матрицей **hood** методом BJIIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=426 2.2,1.22, 3.42	0.86 0.004 0.74 1.61	0.49 0.004 0.57 1.07	0.39 0.005 0.57 0.97
16	It=449 1.1, 0.59, 1.69	0.31 0.003 0.48 0.80	0.27 0.003 0.28 0.56	0.25 0.003 0.45 0.71
32	It=460 0.56, 0.32, 0.89	0.17 0.002 0.32 0.49	0.14 0.001 0.22 0.36	0.26 0.001 0.27 0.53
64	It=463 0.29,0.22, 0.51	0.18 0.002 0.32 0.51	0.07 0.001 0.21 0.28	0.28 0.003 0.45 0.74
100	It=454 0.19, 0.17, 0.37	0.25 0.001 0.27 0.53	0.07 0.001 0.23 0.30	0.26 0.002 0.46 0.73

Таблица 14

Числа итераций и времена счета задачи с матрицей **pwtk** методом BJIIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=3784 2.91, 13.24, 16.2	0.95 0.004 6.91 7.87	0.44 0.005 5.76 6.21	0.45 0.005 6.06 6.51
16	It=3776 1.2, 4.79, 6.04	0.43 0.003 2.77 3.21	0.30 0.003 2.28 2.59	0.29 0.003 2.15 2.45
32	It=3964 0.62, 2.64, 3.26	0.12 0.001 1.73 1.86	0.36 0.002 1.91 2.28	0.39 0.003 2.73 3.12
64	It=4120 0.33, 1.61, 1.94	0.08 0.001 1.67 1.76	0.25 0.002 2.32 2.58	0.3 0.002 3.48 3.79
100	It=4158 0.19, 1.22, 1.41	0.05 0.003 1.60 1.69	0.26 0.001 1.93 2.19	0.12 0.001 3.27 3.39

Как видно из таблиц 11-16, при использовании метода BJIIC-CG для решения модельной задачи и задач из коллекции университета Флориды наблюдается некоторый (не всегда монотонный) рост числа итераций с ростом числа процессоров. Это соответствует приведенным выше результатам теоретического исследования. Однако, благодаря значительному уменьшению числа пересылок, время счета методом BJIIC-CG обычно не очень сильно отличается от времени счета методом IIC-CG.

В таблице 17 приведены числа итераций и времена счета методом BJIIC-CG без использования OpenMP технологии, минимальные времена счета при

Таблица 15

Числа итераций и времена счета задачи с матрицей **msdoor** методом BJIIC-CG на p процессорах без использования и с использованием OpenMP технологии

P		4 нити	8 нитей	16 нитей
8	It=4135	1.7 0.008	0.75 0.01	0.99 0.01
	4.92, 26.8, 31.74	13.68 15.4	10.65 11.42	10.67 11.68
16	It=4427	0.78 0.005	0.38 0.005	0.73 0.005
	2.2, 14.27, 16.48	8.13 8.92	6.44 6.83	7.12 7.86
32	It=5638	0.52 0.002	0.45 0.002	0.30 0.005
	1.14, 7.2, 8.35	4.91 5.43	4.5 4.96	3.85 4.16
64	It=5609	0.22 0.001	0.39 0.003	0.35 0.004
	0.63, 4.2, 4.83	3.44 3.67	3.36 3.76	5.57 5.93
100	It=5216	0.15 0.003	0.28 0.003	0.35 0.003
	0.39, 3.02, 3.41	2.56 2.71	2.68 2.97	5.07 5.42

Таблица 16

Числа итераций и времена счета задачи с матрицей **x104** методом BJIIC-CG на p процессорах без использования и с использованием OpenMP технологии

P	1 нить	4 нити	8 нитей	16 нитей
8	It=8240	1.3 0.003	0.6 0.003	0.57 0.003
	3.72, 15.78, 9.51	9.49 10.8	8.09 8.70	8.90 9.48
16	It=8633	0.63 0.002	0.46 0.002	0.3 0.002
	1.79, 7.45, 9.25	5.17 5.80	4.41 4.87	4.45 4.76
32	It=9465	0.36 0.001	0.31 0.001	0.31 0.001
	0.9, 4.47, 5.37	4.05 4.42	3.99 4.31	4.97 5.29
64	It=9355	0.21 0.002	0.33 0.002	0.36 0.001
	0.6, 3.09, 3.69	3.82 4.04	3.98 4.32	5.44 5.81
100	It=9522	0.13 0.001	0.36 0.001	0.23 0.001
	0.31, 2.1, 2.41	3.54 3.68	4.11 4.48	5.09 5.33

использовании OpenMP технологии, как описано выше, ускорения счета Sp_th благодаря использованию нитей, Sp_pr – ускорения счета для разного числа процессоров по сравнению с временем счета на 8 процессорах в случае расчета без использования OpenMP, Speedup – ускорения счета благодаря использованию MPI+OpenMP подхода по сравнению со счетом на 8 процессорах без использования OpenMP технологии. На рис. 1-7 представлены графики зависимости времен счета этих задач от числа процессоров в логарифмическом масштабе для различных методов без использования (сплошная линия) и с использованием (пунктирная линия) OpenMP технологии.

Таблица 17

Числа итераций и времена счета методом BJIIC-CG различных задач на разном числе процессоров без использования и с использованием OpenMP технологии, ускорения счета благодаря использованию OpenMP технологии, ускорения счета по сравнению с расчетом на 8 процессорах

P p/8	8 1	16 2	32 4	64 8	100 12.5
5_1048576 Sp_th Speedup Sp_pr	It=1824 5.36 Th=8 2.52 2.12 2.12 1	It=1777 2.97 Th=8 1.51 1.97 3.54 1.8	It=1953 1.57 Th=4 1.26 1.25 4.25 3.41	It=1959 0.92 Th=4 1.04 Sp_th < 1 5.82	It=1161 0.56 Th=8 0.64 Sp_th<1 9.57
m_t1 Sp_th Speedup Sp_pr	It=3335 15.25 Th=8 5.56 2.74 2.74 1	It=3296 6.49 Th=16 2.45 2.64 6.22 2.34	It=3439 3.58 Th=8 2.07 1.73 7.63 4.25	It=3649 2.11 Th=4 2.19 Sp_th < 1 7.23	It=3769 1.44 Th=8 1.70 Sp_th < 1 10.6
hood Sp_th Speedup Sp_pr	It=426 3.42 Th=16 0.97 3.52 3.52 1	It=449 1.69 Th=8 0.56 3.01 6.12 2.02	It=460 0.89 Th=8 0.36 2.47 9.5 3.84	It=463 0.51 Th=8 0.28 1.63 1.82 6.7	It=454 0.72 Th=8 0.37 1.95 9.24 4.75
pwtk Sp_th Speedup Sp_pr	It=3784 16.16 Th=8 6.21 2.6 2.6 1	It=3776 6.04 Th=16 2.45 2.46 6.59 2.67	It=3964 3.26 Th=4 1.86 1.75 8.68 4.95	It=4120 1.94 Th=4 1.76 1.1 9.18 8.32	It=4158 1.41 Th=4 1.69 Sp_th < 1 11.77
msdoor Sp_th Speedup Sp_pr	It=4135 31.74 Th=8 11.42 2.77 2.77 1	It=4427 16.48 Th=8 6.83 2.41 4.64 1.52	It=5638 8.35 Th=16 4.16 2.0 7.62 3.8	It=5609 4.83 Th=4 3.67 1.31 8.64 6.57	It=5216 3.41 Th=4 2.71 1.26 11.67 9.3
x104 Sp_th Speedup Sp_pr	It=8240 19.51 Th=8 8.70 2.24 2.24 1	It=8633 9.25 Th=16 4.76 1.94 4.1 2.1	It=9465 5.37 Th=8 4.31 1.25 4.56 3.63	It=9355 3.69 Th=4 4.04 Sp_th <1 5.28	It=9522 2.41 Th=4 3.68 Sp_th <1 8.09

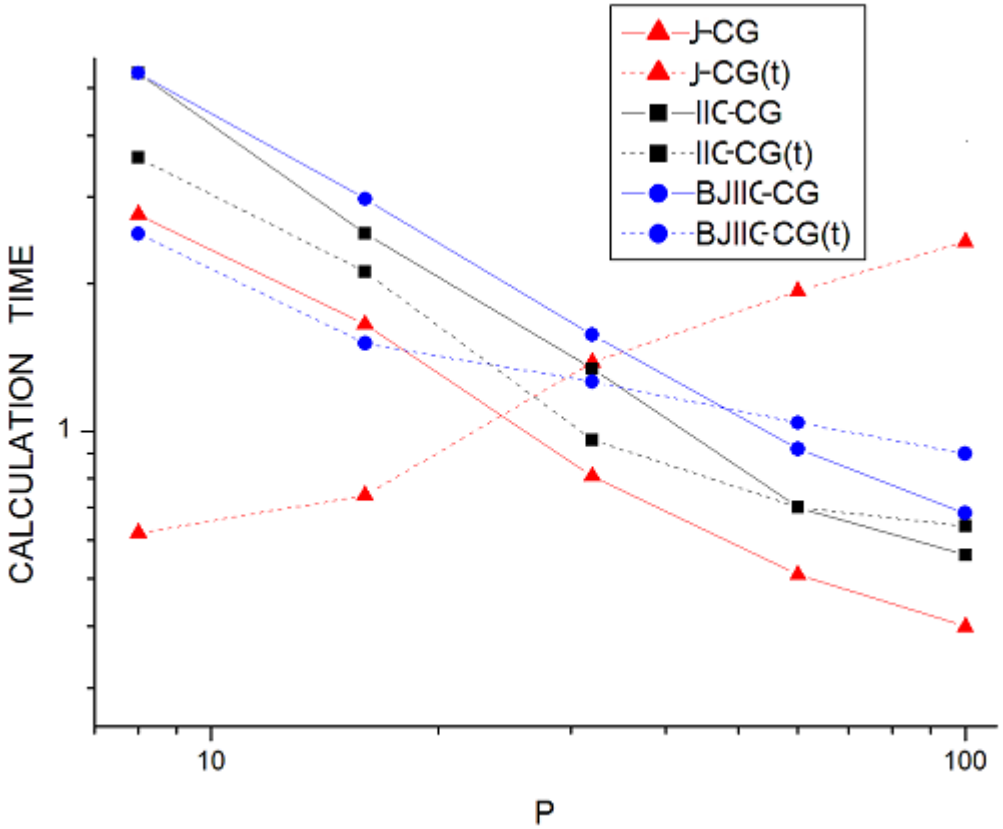

Рис. 1. Времена счета задачи с матрицей 5_1048576 различными методами

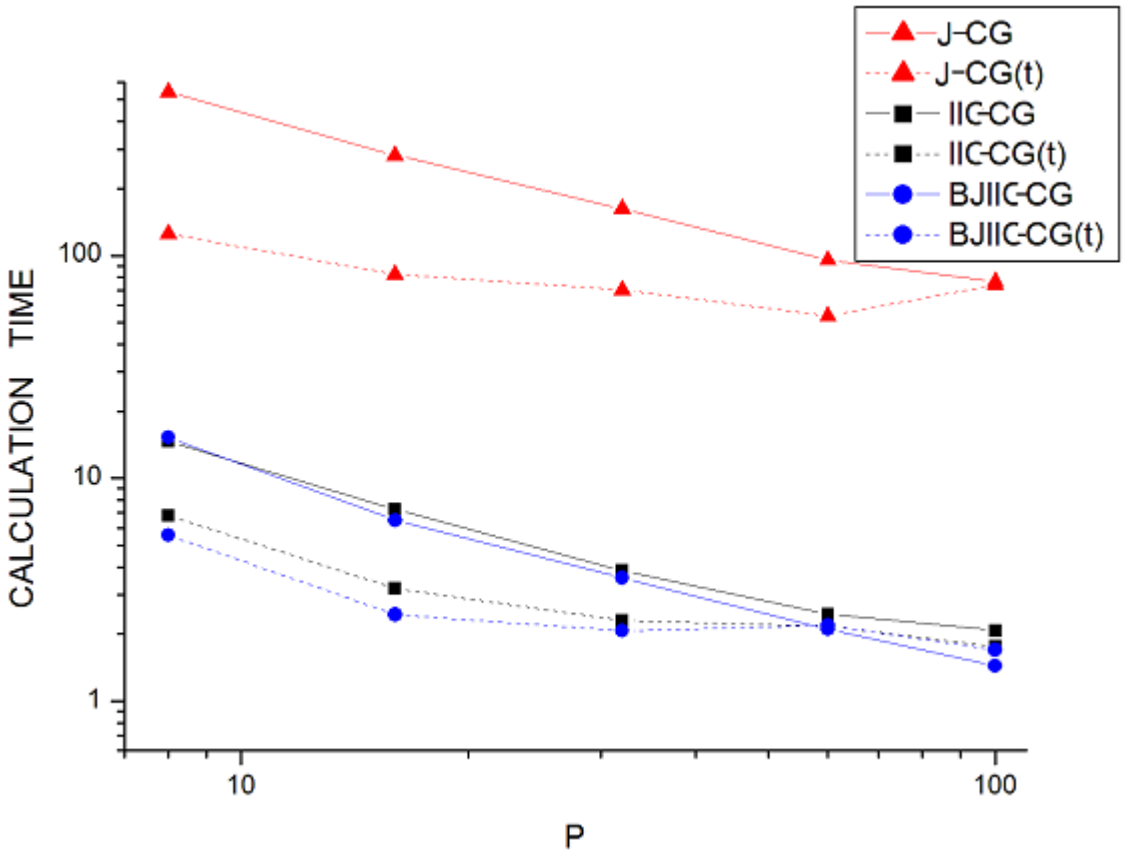

Рис. 2. Времена счета задачи с матрицей **m_t1** различными методами

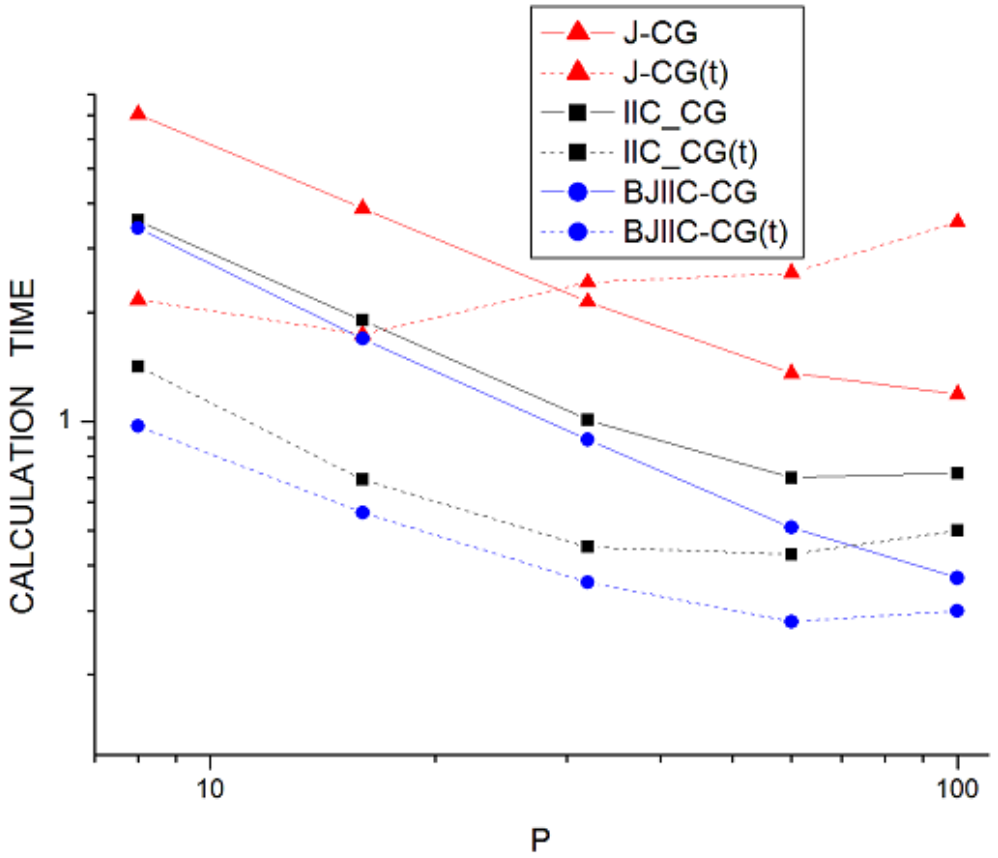

Рис. 3. Времена счета задачи с матрицей **hood** различными методами

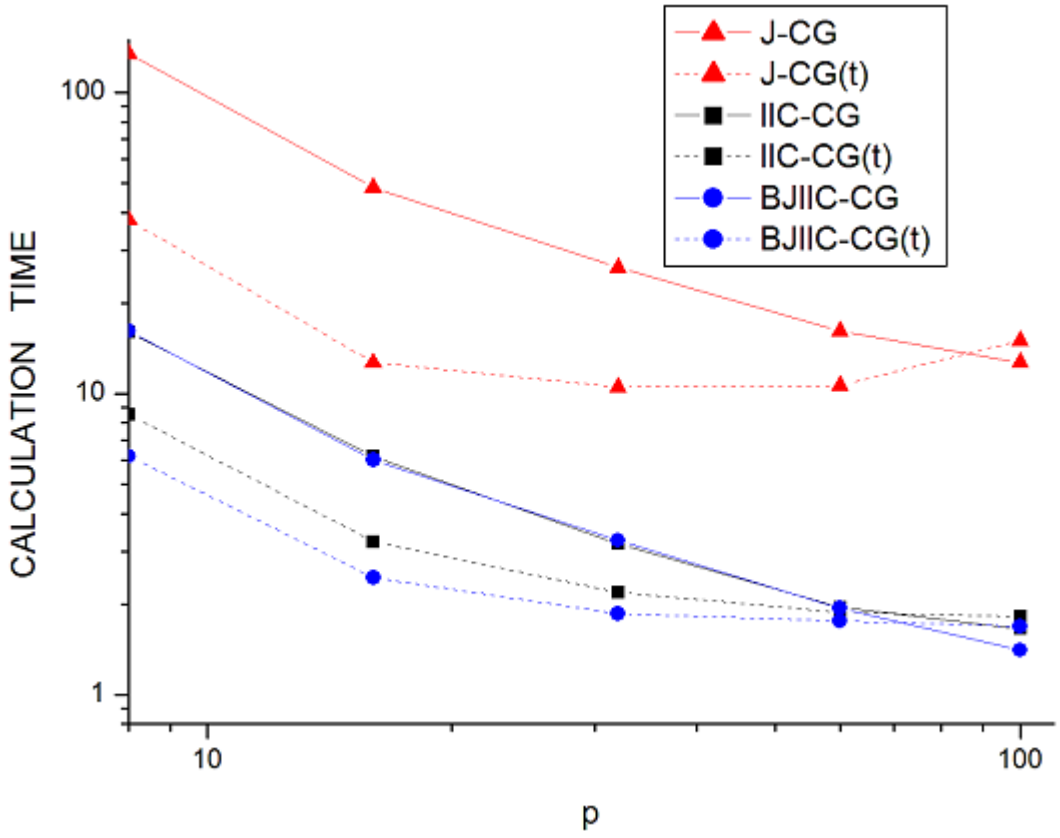

Рис. 4. Времена счета задачи с матрицей **pwtk** различными методами

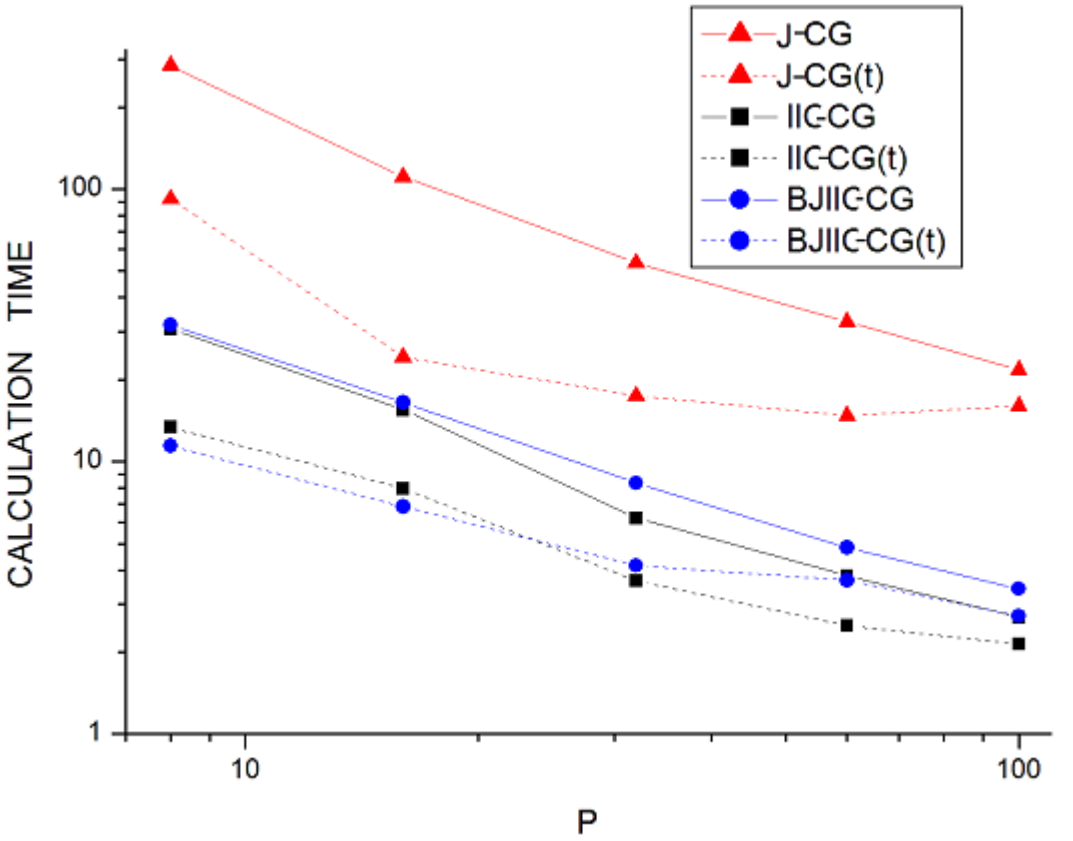

Рис. 5. Времена счета задачи с матрицей **msdoor** различными методами

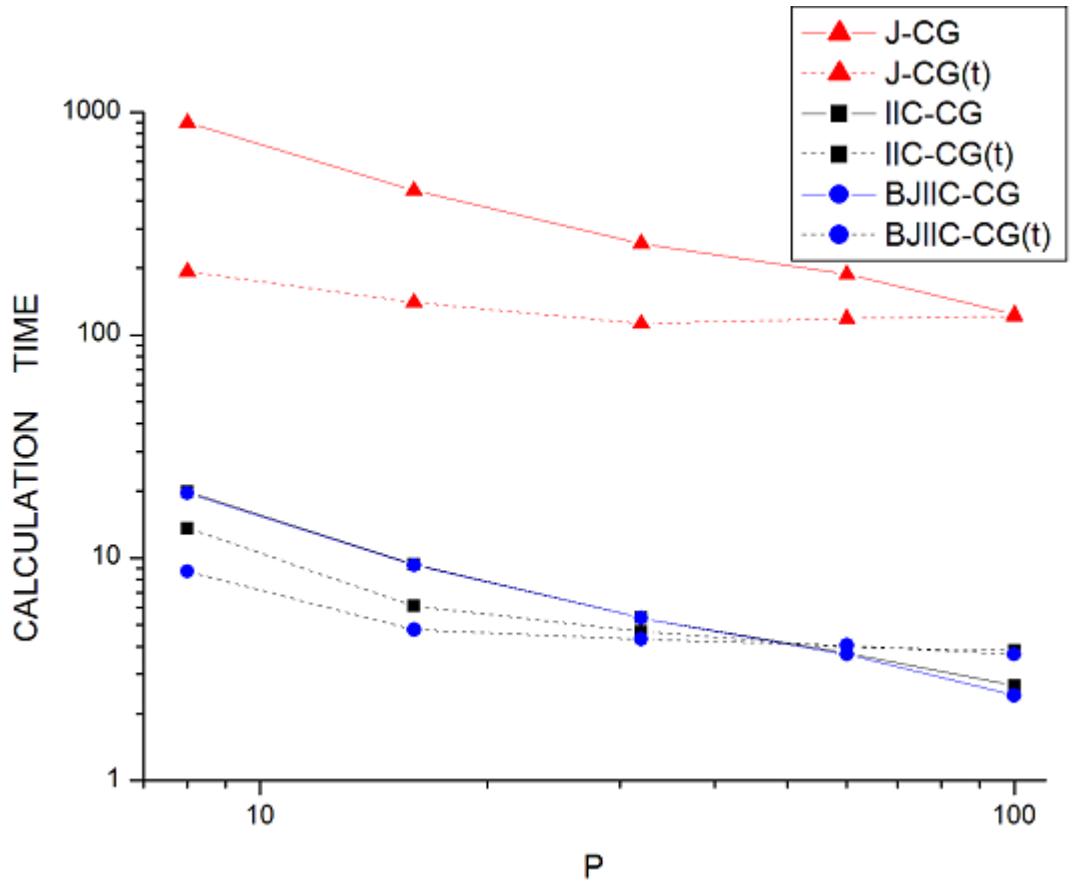

Рис. 6. Времена счета задачи с матрицей **x104** различными методами

Как видно из таблиц 2,3 и рисунков 1-7, 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 очень сильно разреженных матриц использование OpenMP технологии позволяет ускорить расчет лишь для небольшого числа процессов (см. модельная задача, задача с матрицей **hood**). В случае не очень больших, но плотных матриц (**m_t1, x104**) применение OpenMP технологии тоже позволяет существенно ускорить вычисления.

Как видно из таблиц 4-10 и рисунков 1-7, 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 модельной задачи использование OpenMP технологии позволяет ускорить расчет при использовании до 64 процессов. В случае не очень больших, но плотных матриц (**m_t1**) применение OpenMP технологии тоже позволяет существенно ускорить вычисления. Недостаточно по сравнению с J-CG хороший эффект при применении OpenMP при решении задачи с матрицей **x104** связан с недостаточным при этом ускорением при обращении предобусловливателя.

Как видно из таблиц 11-17 и рисунков 1-7, при решении задач методом BJIIC-CG применение OpenMP технологии позволяет существенно ускорить вычисления в случае достаточно больших и не очень сильно разреженных матриц. В случае модельной задачи использование OpenMP технологии позволяет ускорить расчет при использовании до 64 процессов. В случае задачи с матрицей **m_t1** применение OpenMP технологии позволяет ускорить вычисления до 64 процессоров. Недостаточно, по сравнению с J-CG, хороший эффект при применении OpenMP технологии при решении задачи с матрицей **x104** связан с недостаточным ускорением при обращении предобусловливателя.

Как видно из рисунков 2-7, при небольшом числе процессоров быстрее всего происходят расчеты рассматриваемых задач методом BJIIC-CG с применением MPI+OpenMP. Расчет задачи с матрицей **hood** для любого рассмотренного числа процессоров быстрее всего происходит методом BJIIC-CG с применением MPI+OpenMP. В случае задачи с матрицей **msdoor** начиная с p=32 быстрее всего происходят расчеты методом IIC-CG с применением MPI+OpenMP. При решении задач с матрицами **m_t1**, **x104** начиная с p=64, а в случае задачи с матрицей **pwtk** начиная с p=100, быстрее всего расчет происходит методом BJIIC-CG без применения OpenMP технологии.

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, заметно больше. Следует ожидать, что потеря эффективности от

использования OpenMP технологии наступит при значительно большем числе процессоров.

Заметим, что применение OpenMP технологии позволяет в ряде случаев уменьшить количество использованных процессоров при параллельной реализации расчетов с помощью MPI+OpenMP подхода для получения того же времени счета. Еще раз отметим, что решение задач с большими и сильно разреженными матрицами с применением OpenMP технологии методом J-CG начиная с некоторого числа процессоров становится неэффективно.

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

Время счета задач методом IIC-CG, как правило, близко к времени их счета методом BJIIC-CG. С точки зрения программной реализации метод BJIIC-CG проще, чем метод IIC-CG.

Список литературы

1. Davis T., Hu Y.F. University of Florida sparse matrix collection.//ACM Trans. on Math.~Software. 2011. V.38, N.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//Presented at 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 // Препринты ИПМ им. М.В.Келдыша. 2017. № 37. 28 с. 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.: Из-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. V.1. - P.179-210.
7. Kolotilina L.Yu., Yeremin A.Yu. Factorized sparse approximate inverse preconditionings. I.~Theory // SIAM J. Matrix Anal. Appl. 1993. V.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 (Yu.G.Evtushenko, V.A.Garanzha, M.K.Kerimov, eds.), Procs. Int. Conf. NUMGRID2010, Moscow, 11-13 Oct. 2010, P. 110-116.
9. Chow E. A priori sparsity patterns for parallel sparse approximate inverse preconditioners // SIAM J. Sci. Comput. 2000. V.21. N.5. - P.1804-1822.
10. Chow E. Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns // Internat. J. High Performance Comput. Appl. 2001. V.15. N.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. Linear Algebra Appl. 1999. V.6. - P.515-531.
12. Anzt H., Huckle T.K., Brackle J., Dongarra J. Incomplete Sparse Approximate Inverses for Parallel Preconditioning //Parallel Computing. 2018. V.71 –P.1–22.
13. Axelsson O. Iterative solution methods. New York: Cambridge Univ. Press, 1994.
14. Капорин И.Е. Предобусловливание итерационных методов решения систем линейных алгебраических уравнений. Дисс. на соиск. ученой степени д.ф.-м.н. 2011. Москва. 216 С.
15. Капорин И.Е. Использование полиномов Чебышева и приближенного обратного треугольного разложения для предобусловливания метода сопряженных градиентов // Ж. вычисл. матем. и матем. физики. 2012. Т.52. № 2. – С.1-26.

Оглавление