Записки научных
семинаров ПОМИ
Том 405, 2012 г.

Р. Р. Ахунов, С. П. Куксенко, В. К. Салов, Т. Р. Газизов

ФОРМАТЫ ХРАНЕНИЯ РАЗРЕЖЕННЫХ МАТРИЦ И УСКОРЕНИЕ РЕШЕНИЯ СЛАУ С ПЛОТНОЙ МАТРИЦЕЙ ИТЕРАЦИОННЫМИ МЕТОДАМИ

§1. Введение

При решении задач математического моделирования основные вычислительные затраты часто приходятся на решение СЛАУ с плотной матрицей. Поэтому актуально совершенствование математических методов и алгоритмов для решения таких СЛАУ. Существуют два класса методов решения СЛАУ: точные методы (например, метод Гаусса) и итерационные методы. Для точных методов основные вычислительные затраты пропорциональны N^3 (N – порядок матрицы), что существенно ограничивает использование таких методов в задачах математического моделирования, особенно, при большом порядке матрицы СЛАУ. Для обычных итерационных методов вычислительные затраты пропорциональны $N_{it} \cdot N^2$ (N_{it} – количество итераций). Из этого следует, что при $N_{it} < N$ (а это часто имеет место) использовать итерационные методы выгоднее. Для уменьшения вычислительных затрат в итерационных методах используют предобусловливание [1]. Однако это требует хранения дополнительной матрицы, что увеличивает требуемую память компьютера. Особенностью дополнительной матрицы является то, что она, как правило, является разреженной [1]. Следовательно, естественным является применение форматов разреженных матриц для экономии требуемой памяти [2]. Между тем, можно предположить, что их использование может не только дать экономию памяти, но и ускорить решение СЛАУ [3, 4]. В

Ключевые слова: система линейных алгебраических уравнений, разреженная матрица, итерационные методы, предобусловливание, предфильтрация.

Работа выполнена в порядке реализации постановления No. 218 Правительства РФ от 09.04.2010 г. “О мерах государственной поддержки развития кооперации российских высших учебных заведений и организаций, реализующих комплексные проекты по созданию высокотехнологичного производства” и договора No. 13.G25.31.0017 от 07.09.2010 между ОАО “ИСС” им. акад. М. Ф. Решетнева” и Минобрнауки РФ.

работе [5] рассмотрено решение СЛАУ с плотной матрицей итерационными методами с предобусловливанием. Предложено использование разреженных матриц в предобусловливании [6]. Однако использование форматов разреженных матриц для ускорения решения СЛАУ с плотными матрицами остаётся неисследованным.

Цель данной работы – исследование возможностей ускорения решения СЛАУ с плотной матрицей итерационными методами за счет использования форматов хранения разреженных матриц. Для достижения поставленной цели необходимо: выполнить обзор и сравнение форматов хранения разреженных матриц, выбрать приемлемый формат, по возможности усовершенствовать его, а также выполнить вычислительный эксперимент.

§2. Обзор распространённых форматов хранения разреженных матриц

Первый формат хранения был предложен Д. Э. Кнутом в 1968 г. При использовании данного формата матрица представляется в виде пяти одномерных массивов. Первый массив содержит ненулевые элементы матрицы. Второй и третий массивы содержат информацию о положении элемента в матрице. Для ускорения работы с матрицей использованы два дополнительных массива, в которых хранятся указатели на следующий ненулевой элемент в строке и в столбце соответственно. Также необходимо хранить указатели входа в одномерный массив для строк и столбцов.

Данный формат для хранения требует пять векторов (не считая указателей на строки), что неэкономно. Достоинством же этого метода является быстрый доступ к элементу матрицы, а также легкость добавления нового элемента. Существуют другие форматы, которые являются модификациями метода Кнута. Например, формат Рейнболдта и Местеньи (1973 г.), сохраняющий ценные свойства формата Кнута, но использующий значительно меньше памяти. Еще один вариант формата Кнута использовался Ларкумом (1971 г.) для хранения симметричных матриц, элементами которых могут быть как числа, так и подматрицы.

Разреженный строчный формат, предложенный Чангом (1969 г.) и Густавсоном (1972 г.) – это один из наиболее широко используемых форматов хранения разреженных матриц. Этот формат предъявляет минимальные требования к памяти и, в то же время, оказывается

очень удобным для нескольких важных операций над разреженными матрицами [2]: сложение, умножение, перестановка строк и столбцов, транспонирование, решение СЛАУ с разреженными матрицами как прямыми, так и итерационными методами и т.д. [7]. Значения ненулевых элементов матрицы и соответствующие индексы столбцов хранятся в этом формате по строкам в двух массивах. Используется также массив указателей на ненулевые элементы, с которых начинается очередная строка. Существуют два типа этого формата: разреженный строчный формат и разреженный столбцовый формат. Они отличаются лишь порядком заполнения: в столбцовом формате элементы записываются по столбцам.

§3. Аналитические оценки коэффициента сжатия и сравнение форматов

В данной работе для сравнительной оценки перечисленных форматов используется коэффициент сжатия, определяемый отношением N^2 к количеству хранимых данных. При этом тип данных во внимание не принимается (хранятся как сами ненулевые элементы, так и указатели, которые требуют меньше бит). Таким образом, получается нижняя граница коэффициента сжатия, используемая для сравнения форматов между собой.

Для определения коэффициента сжатия можно использовать подсчет числа данных конкретной матрицы, хранящихся в сжатом виде, но удобнее использовать аналитические выражения (формулы), позволяющие получить простую оценку коэффициента сжатия без подсчета. В известной авторам литературе такие формулы не приводятся, но их легко получить, исходя из структуры формата хранения разреженных матриц. В табл. 1 представлены полученные формулы для коэффициента сжатия некоторых форматов хранения разреженной матрицы.

Примечание. k – коэффициент сжатия; N – порядок матрицы; $N_{\text{н}}$ – количество ненулевых элементов матрицы; q – плотность матрицы.

Очевидно, что коэффициент сжатия зависит от порядка матрицы и количества ненулевых элементов в ней. Оценить количество ненулевых элементов можно с помощью показателя плотности матрицы, представляющего собой отношение числа её ненулевых элементов к N^2. Для примера на рис. 1 представлены рассчитанные по полученным

Таблица 1. Формулы для определения параметров форматов хранения

Разреженный формат	Коэффициент сжатия	Граница эффективности	Максимальный коэффициент сжатия
Кнута	$k = \frac{N^2}{5N_н+2N}$	$q = \frac{(N^2-2N)\cdot 100\%}{5N^2}$	$k = \frac{100\%}{5q}$
Рейнболдта–Местеньи	$k = \frac{N^2}{3N_н+2N}$	$q = \frac{(N^2-2N)\cdot 100\%}{3N^2}$	$k = \frac{100\%}{3q}$
Разреженный строчный	$k = \frac{N^2}{2N_н+N}$	$q = \frac{(N^2-N)\cdot 100\%}{2N^2}$	$k = \frac{100\%}{2q}$

формулам зависимости коэффициента сжатия различных форматов от плотности матрицы при $N = 1000$. Видно, что формат хранения разреженной матрицы не всегда эффективен, т.е. может не уменьшать, а увеличивать число хранимых данных. Очевидно, что для каждого формата существует граница эффективности (значение плотности матрицы, при которой коэффициент сжатия равен 1), значение которой можно оценить аналитически. В табл. 1 приведены формулы для оценки плотности матрицы, ниже которой использование того или иного формата неэффективно.

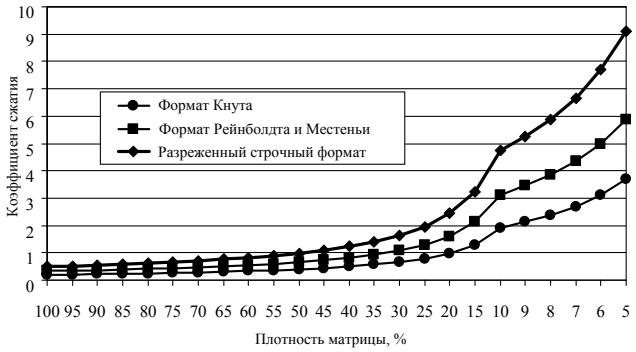

Рис. 1. Зависимость коэффициентов сжатия от плотности матрицы при использовании различных форматов

Коэффициент сжатия зависит от порядка матрицы. На рис. 2 показана зависимость коэффициента сжатия от порядка матрицы при плотности матрицы 10%. Видно, что коэффициент сжатия с ростом N стремится к максимально возможному значению. Предел выражения для коэффициента сжатия при порядке матрицы N, стремящемся к бесконечности, даёт аналитическую формулу для определения данного максимального значения (табл. 1).

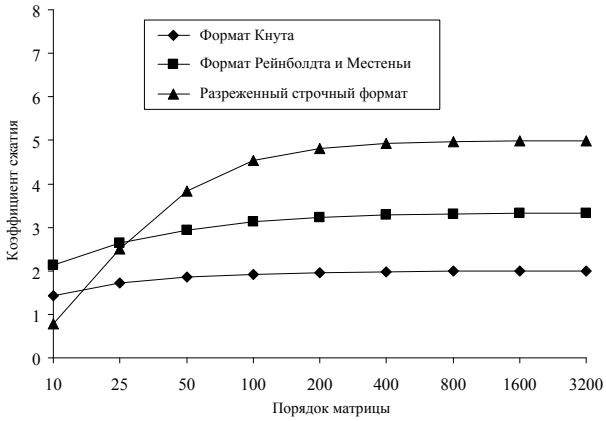

Рис. 2. Зависимость коэффициентов сжатия от порядка матрицы при использовании различных форматов

Ясно, что самым эффективным по степени сжатия является разреженный строчный формат. Поэтому для реализации и дальнейшего использования был выбран именно он.

§4. Использование разреженного строчного формата при формировании предобусловливания

Методы предобусловивания делятся на две группы: явные и неявные. В данной работе используется неявное предобусловливание, поскольку оно хорошо зарекомендовало себя при решении СЛАУ с плотной матрицей [5]. При неявном предобусловливании исходную СЛАУ можно представить в виде

$$\mathbf{MAx} = \mathbf{Mb}, \tag{1}$$

где $\mathbf{M}$ – невырожденная матрица.

Наиболее работоспособные методы построения неявного преобусловливания основаны на LU-разложении [5]. Однако данный метод предполагает неконтролируемое добавление новых элементов, что при использовании разреженных форматов недопустимо, т.к. данные форматы могут стать неэффективными. Более того, разреженный строчный формат имеет тот недостаток, что добавление новых элементов сложно. Это связано со структурой формата: элементы должны быть отсортированы, следовательно, добавление нового элемента требует операции сдвига всех оставшихся элементов. Это требует дополнительных операций над матрицей и, следовательно, замедляет работу алгоритма. Однако существует метод, который не требует добавления новых элементов – это ILU(0)-разложение. Один из алгоритмов (ikj-версия) данного метода выглядит следующим образом:

1 Для $i = 2, ..., N$
2 Для $k = 1, ..., i-1$
3 Если $a^S_{i,k} \neq 0$
4 $a^S_{i,k} = a^S_{ik} / a^S_{k,k}$
5 Для $j = k+1, ..., N$
6 Если $a^S_{i,j} \neq 0$
7 $a^S_{i,j} = a^S_{i,j} - a^S_{i,k} \times a^S_{kj}$
8 Увеличить j
9 Увеличить k
10 Увеличить i

Здесь a^S – элемент матрицы $\mathbf{A}_S$, которая получается из матрицы $\mathbf{A}$ после предфильтрации.

После того как разложение получено, полагается $\mathbf{M}$=$\mathbf{A}_S$. Данный алгоритм позволяет пересчитать i-ю строку матрицы $\mathbf{A}_S$ в i-ю строку матриц $\mathbf{L}$ и $\mathbf{U}$. Первые $j-1$ строк матрицы $\mathbf{A}_S$ участвуют в определении j-х строк матриц $\mathbf{L}$ и $\mathbf{U}$, но сами больше не модифицируются, что позволяет не использовать при вычислениях дополнительную память для хранения отдельно матриц $\mathbf{L}$ и $\mathbf{U}$.

В разреженном строчном формате получить прямой доступ к ненулевому элементу невозможно. Самый простой вариант доступа к элементу – использование функции его поиска, которая и была реализована в алгоритме. Алгоритм получился простым, но медленным. Так,

время работы двух указанных алгоритмов (первый алгоритм ILU(0)-разложения без использования разреженного формата матрицы, второй с использованием) сравнивалось на матрице порядка $N = 1000$. В результате оказалось, что алгоритм, использующий разреженный формат хранения, работает примерно в 500 раз дольше обычного алгоритма без сжатия матрицы. Следовательно, алгоритм в таком виде использовать нельзя, и его необходимо усовершенствовать.

§5. Усовершенствование алгоритма ILU(0)-разложения матрицы, хранящейся в разреженном строчном формате

Если отметить элементы матрицы, используемые в полном цикле ikj-версии (строки 5–7), то получится картина, изображенная на рис. 3. Здесь возможно использование цикла по ненулевым элементам в разреженном строчном формате, тогда поиск элементов не нужен, операция (строка 7 в алгоритме) будет производиться только с теми элементами строки, у которых равны индексы столбцов. Алгоритм в этом случае будет выглядеть следующим образом:

1 Для $i = 2, ..., N$
2 Для $k = 1, ..., i-1$
3 Найти S – номер элемента $a^S_{i,k}$ в векторе **aelem**;
4 Если **aelem**$(S) \neq 0$
5 **aelem**$(S) =$ **aelem**$(S)/$**Poisk**(k, k);
6 Найти T – номер элемента $a^S_{i,k}$ в векторе **aelem**;
7 $TekElem1 = T + 1$;
8 Найти T – номер элемента $a^S_{i,k+1}$ в векторе **aelem**;
9 $TekElem2 = T$;
10 $Pr =$ Истина;
11 Пока $Pr =$ Истина
12 Если **jptr**$(TekElem1) =$ **jptr**$(TekElem2)$
13 **aelem**$(TekElem1) =$ **aelem**$(TekElem1) -$ **aelem**$(T) \times$ **aelem**$(TekElem2)$;
14 Увеличить $TekElem1$ и $TekElem2$;
15 Если **jptr**$(TekElem1) >$ **jptr**$(TekElem2)$
16 Увеличить $TekElem2$;
17 Если **jptr**$(TekElem1) <$ **jptr**$(TekElem2)$
18 Увеличить $TekElem1$;

19 Если **iptr**$(k{+}1) < TekElem2$ или **iptr**$(i{+}1) < TekElem2$
20 $Pr =$ Ложь;
21 Увеличить k
22 Увеличить i

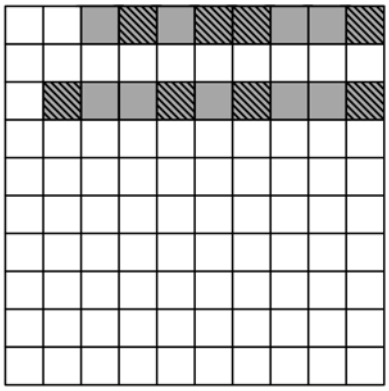

Рис. 3. Схематичное расположение элементов, используемых за один цикл алгоритма ILU(0)-разложения (ikj-версия). Штриховкой отмечены ненулевые элементы, а серым цветом – нулевые

В приведенном алгоритме сначала происходит поиск первого элемента (на рис. 3 – это второй элемент третьей строки), который используется при нахождении очередного элемента (строка 13), поэтому позиция элемента запоминается (переменная S). В строке 5 присутствует функция $Poisk(k,k)$, которая предназначена для поиска элемента в матрице и возвращает его значение. Переменная Pr сигнализирует, когда достигнут конец строки. Переменные $TekElem1$ и $TekElem2$ определяют текущий элемент в первой и второй строках соответственно. Переход по элементам стоки в матрице происходит в цикле по следующему принципу: если значение какой-либо из

двух переменных больше другого, то инкрементируют значение второй (строки 15–18); если же значения переменных равны (т.е. существуют ненулевые элементы в двух строках текущего столбца), то выполняется операция и значения обеих переменных инкрементируются (строки 12–14).

Поиск диагонального элемента в строке 5 данного алгоритма существенно увеличивает время его работы. Поэтому целесообразно ввести в формат хранения дополнительный вектор **Diag**, в котором будут храниться указатели на диагональные элементы. (Выигрыш, полученный за счет этого, будет представлен ниже.)

Окончательная версия алгоритма примет вид:

1 Для $i = 2, ..., N$
2 Для $k = 1, ..., i-1$
3 Найти S – номер элемента $a^S_{i,k}$ в векторе **aelem**;
4 Если $\mathbf{aelem}(S) \neq 0$
5 $\mathbf{aelem}(S) = \mathbf{aelem}(S)/\mathbf{aelem}(\mathbf{Diag}(k))$;
6 Найти T – номер элемента $a^S_{i,k}$ в векторе **aelem**;
7 $TekElem1 = T + 1$;
8 Найти T – номер элемента $a^S_{i,k+1}$ в векторе **aelem**;
9 $TekElem2 = T$;
10 $Pr =$ Истина;
11 Пока $Pr =$ Истина
12 Если $\mathbf{jptr}(TekElem1) = \mathbf{jptr}(TekElem2)$
13 $\mathbf{aelem}(TekElem1) = \mathbf{aelem}(TekElem1) - \mathbf{aelem}(T) \times \mathbf{aelem}(TekElem2)$;
14 Увеличить $TekElem1$ и $TekElem2$;
15 Если $\mathbf{jptr}(TekElem1) > \mathbf{jptr}(TekElem2)$
16 Увеличить $TekElem2$;
17 Если $\mathbf{jptr}(TekElem1) < \mathbf{jptr}(TekElem2)$
18 Увеличить $TekElem1$;
19 Если $\mathbf{iptr}(k+1) < TekElem2$ или $\mathbf{iptr}(i+1) < TekElem2$
20 $Pr =$ Ложь;
21 Увеличить k
22 Увеличить i

При использовании неявного предобусловливания на каждой итерации необходимо решать СЛАУ вида

$$\mathbf{My} = \mathbf{z}. \tag{2}$$

С учетом того, что матрица $\mathbf{M}$ хранится с помощью разреженного строчного формата, при реализации алгоритма для решения СЛАУ использовался тот же принцип, что и в ILU(0)-разложении, рассмотренном выше. Разработанный алгоритм выглядит следующим образом:

1 Для $i = 1, ..., N$
2 Найти $S = jptr(i)$, что соответствует элементу a_{i1}^S в векторе **aelem**;
3 $y_i = b_i$
4 Для $k = 1, ..., N$
5 Если $k < i$
6 $y_i = y_i - \mathbf{aelem}(S) y_k$;
7 Увеличить k
8 Увеличить i
9 Для $i = N, ..., 1$
10 $x_i = y_i$
12 Найти $S = \mathbf{jptr}(i)$, что соответствует элементу a_{i1}^S в векторе **aelem**;
13 Для $k = 1, ..., N$
14 Если $k > i$
15 $x_i = x_i - \mathbf{aelem}(S) x_k$;
16 Увеличить k
17 $x(i) = x(i)/\mathbf{aelem}(\mathbf{Diag}(i))$
18 Уменьшить i

Здесь строки 1–8 – это прямой ход, 9–18 – обратный ход.

§6. Вычислительный эксперимент

Для вычислительного эксперимента использовался персональный компьютер (при вычислениях не использовалось распараллеливание, т.е. работало одно ядро процессора) с параметрами: платформа – AMD Athlon(tm) 64 X2 Dual; частота процессора – 2100 МГц; объем ОЗУ – 2 Гбайт; число ядер – 2; операционная система – Windows XP.

Сравнение времени работы алгоритмов ILU(0)-разложения для различной плотности матрицы $\mathbf{A}_S$ порядка $N = 1000$ без использования (T) дополнительного вектора **Diag** и с его использованием (T_D) представлено в табл. 2. Как видно, использование вектора **Diag** ускоряет ILU(0)-разложение в 1,14–1,23 раза.

Таблица 2. Сравнение времени выполнения ILU(0)-разложения разными алгоритмами

Допуск обнуления	Плотность, %	T, с	T_D, с	$\frac{T}{T_D}$
0,012	59	285	258	1,14
0,014	56	261	234	1,13
0,016	53	242	213	1,16
0,018	50	224	195	1,15
0,02	47	207	178	1,16
0,022	45	192	162	1,17
0,024	42	175	146	1,17
0,026	40	163	133	1,17
0,028	38	152	123	1,18
0,03	36	142	112	1,19
0,032	35	132	103	1,19
0,034	33	124	95	1,20
0,036	31	116	88	1,20
0,038	30	109	81	1,20
0,04	29	103	75	1,21
0,042	27	97	70	1,21
0,044	26	91	65	1,23
0,046	25	86	60	1,22
0,048	24	81	56	1,23
0,05	23	76	52	1,23

Далее приведены результаты вычислительного эксперимента на полной задаче решения СЛАУ итерационным методом бисопряженных градиентов (BiCGStab). Измерялось время трёх составляющих решения: предфильтрации (в алгоритме, использующем разреженный формат, в ходе предфильтрации производится и конвертация матрицы в

разреженный строчный формат); ILU(0)-разложения; итерационного процесса.

Оценка временных затрат сделана на трёх матрицах (полученных последовательным учащением сегментации границ проводник-диэлектрик и диэлектрик-диэлектрик из задачи вычисления электрической ёмкости двух полосок на диэлектрическом слое над идеально проводящей плоскостью) для двух вариантов алгоритма BiCGStab: без использования формата хранения разреженных матриц (T) и с использованием формата хранения разреженных матриц с дополнительным вектором **Diag** ($Tcsrd$).

В экспериментах изменялся допуск обнуления, меняющий плотность матрицы $\mathbf{A}_S$. Итерации продолжались до тех пор, пока относительная норма вектора невязки не становилась меньше 10^{-6}.

Сравнение времени двух алгоритмов на различных матрицах приведено на рис. 4 (а, в, д). Использовались матрицы следующих порядков: $N = 4800$ (рис. 4а), $N = 6000$ (рис. 4в), $N = 8000$ (рис. 4д). Видно, что ускорение алгоритма с использованием строчного формата хранения разреженной матрицы получается не при всех значениях плотности, а только для значений меньших, чем определенное пороговое значение. При этом максимальное ускорение от использования формата хранения разреженной матрицы достигается приблизительно при оптимальном значении допуска обнуления, соответствующем минимальному времени решения СЛАУ. В табл. 3 приведены данные сравнения алгоритмов для различных порядков матриц. Из нее следует, что ускорение имеет место на различных матрицах, изменяясь в пределах от 1,5 до 1,6 раза. Дополнительно приведено время решения методом Гаусса (Tge) и ускорение относительно него, изменяющееся в пределах от 1,2 до 4,4 раза. В данной задаче использовался алгоритм решения методом Гаусса, основанный на LU-разложении и последующем решении. (Данный алгоритм реализован авторами самостоятельно, без использования специализированных библиотек и методов, для проведения объективного сравнения алгоритмов в равных условиях. Библиотечные программы решения СЛАУ не применялись, поскольку в них используются различные методы ускорения алгоритма: распараллеливание, разбиение матрицы на блоки с целью использования кэш процессора и др.)

Таблица 3. Сравнение времени решения СЛАУ разными алгоритмами

N	T, с	$Tcsrd$, с	$\frac{T}{Tcsrd}$	Tge, с	$\frac{Tge}{Tcsrd}$
4800	164	101	1,6	129	1,3
6000	335	206	1,6	254	1,2
8000	216	140	1,5	616	4,4

Если проанализировать время работы алгоритма решения СЛАУ по составляющим, то выясняется, что ускорение получается именно за счёт вычисления ILU(0)-разложения. На рис. 4 (б, г, е) приведено время работы алгоритма ILU(0)-разложения для матриц порядка $N = 4800$ (рис. 4б), $N = 6000$ (рис. 4г) и $N = 8000$ (рис. 4е). Вычисление ILU(0)-разложения занимает большую часть всего времени решения СЛАУ, следовательно, его сокращением достигается общее ускорение всего алгоритма. Ускорение вычисления ILU(0)-разложения достигается за счет использования разреженного строчного формата, т.к. в данном формате нулевые элементы матрицы не хранятся, и, следовательно, они не участвуют в алгоритме, в отличие от алгоритма, который работает с полной матрицей.

§7. Заключение

Проведен обзор разреженных форматов хранения матриц, получены аналитические выражения, позволяющие сравнить между собой разреженные форматы хранения по коэффициенту сжатия. Показано, что коэффициент сжатия матрицы установленной плотности имеет конечное значение при больших матрицах. Из рассмотренных форматов выбран разреженный строчный формат. Разработан алгоритм с использованием разреженного строчного формата для предобусловливания при решении СЛАУ итерационным методом. Для ускорения работы алгоритма усовершенствован разреженный строчный формат: для быстрого доступа к диагональным элементам добавлен вектор указателей на эти элементы. Данное усовершенствование ускорило работу алгоритма в 1,14–1,23 раза. Проведен вычислительный эксперимент (на матрицах порядка $N = 4800, 6000, 8000$), показавший, что использование разреженного строчного формата позволяет ускорить

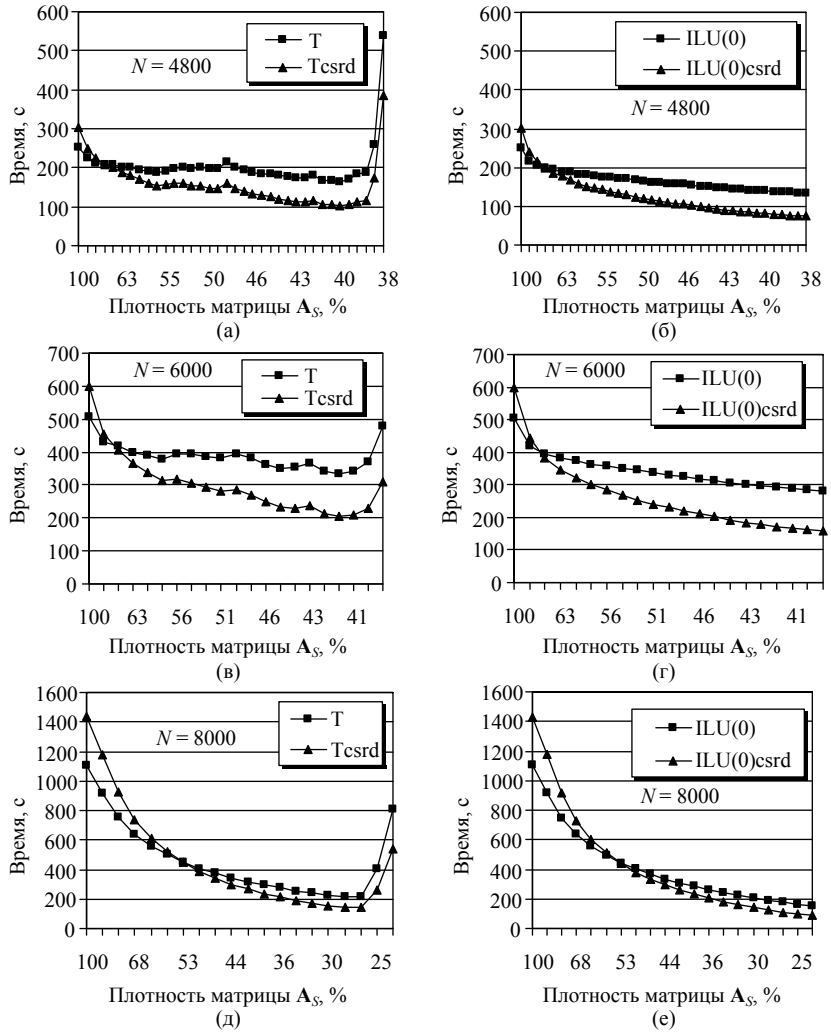

Рис. 4. Сравнение времени алгоритмов: а, в, д – полная задача решения СЛАУ; б, г, е – только вычисление ILU(0)-разложения

работу алгоритма решения СЛАУ в 1,5–1,6 раз по сравнению с алгоритмом, работающим с полной матрицей, и в 1,2–4,4 раза по сравнению с методом Гаусса.

По мнению авторов, полученные результаты могут дать новый импульс использованию итерационных методов при решении СЛАУ с плотной матрицей. Действительно, с недавнего времени они стали гораздо более эффективной альтернативой прямым методам, поскольку позволили получить ускорение в десятки раз. Однако необходимость хранения ещё одной матрицы существенно ограничивает их использование для матриц больших порядков, когда они не помещаются в оперативную память компьютера, хотя именно для таких матриц более всего и актуально ускорение. Между тем, за счет выбора допуска обнуления предфильтрации можно значительно снизить плотность матрицы предобусловливателя, а использование при этом формата хранения разреженной матрицы позволяет значительно снизить дополнительные затраты памяти на её хранение. Кроме того, решение СЛАУ предложенным алгоритмом даст, кроме ускорения за счет перехода от прямых методов к итерационным, дополнительное ускорение. Отметим, что рост N предполагает рост выигрыша как по затратам памяти, так и по ускорению, но это требует более детального исследования. Таким образом, использование результатов работы позволит уменьшить затраты как памяти компьютера, так и времени вычисления при решении задач большой размерности, а комплексный характер результатов (аналитические формулы, усовершенствованные алгоритмы, результаты вычислительных экспериментов) даёт всё необходимое для такого использования.

Литература

1. Т. Р. Газизов, С. П. Куксенко, *Оптимизация допуска обнуления при решении СЛАУ итерационными методами с предобусловливанием в задачах вычислительной электродинамики.* — Электромагн. волны электр. системы No. 8 (2004), 26–28.
2. С. Писсанецки, *Технология разреженных матриц.* Мир, М., 1988.
3. J. R. Gilbert, C. B. Moler, R. Schreiber, *Sparse matrices in MATLAB: design and implementation.* — SIAM J. Matrix Anal. Appl. **13** (1) (1992), 333–356.
4. M. Lujan, L. Freeman, J. Gurd, *Performance evaluation of storage formats for sparse matrices in fortran.* — In: High Performance Computing and Communications (Munich, Germany, September 13–15, 2006), Springer (2006), pp. 160–169.

5. С. П. Куксенко, Т. Р. Газизов, *Итерационные методы решения системы линейных алгебраических уравнений с плотной матрицей.* — Томский государственный университет, Томск, 2007.
6. М. Ю. Баландин, Э. П. Шурина, *Методы решения СЛАУ большой размерности.* Изд-во НГТУ, Новосибирск, 2000.
7. Y. Saad, *Iterative Methods for Sparse Linear Systems.* SIAM, 2003.

Akhunov R. R., Kuksenko S. P., Salov V. K., Gazizov T. R. Sparse matrix storage formats and acceleration of iterative solution of linear algebraic systems with dense matrices.

In the paper, formulas for comparing sparse matrix storage formats are derived. An iterative algorithm for solving linear algebraic systems using the sparse row format for storing prefiltered preconditioners is designed. A modification of the sparse row format leading to 1.14–1.23 times speed-up for matrices of order 1000 is suggested. It is demonstrated that as opposed to the usual storage format, the sparse row format provides for 1.5–1.6 times speed-up in solving linear systems of orders 4800, 6000, and 8000. The use of the results obtained allows one to reduce both memory and time requirements in solving large-scale problems with dense matrices.

Томский государственный университет
систем управления и радиоэлектроники,
кафедра телевидения и управления,
пр. Ленина, 40, г. Томск 634050, Россия

E-mail: arr@pop3.ru

E-mail: ksergp@sibmail.com

E-mail: catred@mail2000.ru

E-mail: talgat@tu.tusur.ru

Поступило 2 февраля 2012 г.