Эффективный предобуславливатель для решения систем линейных уравнений, возникающих в задаче фильтрации в деформируемой поровой среде

О.С. Борщук

ООО «РН-Технологии», ФГБОУ ВО «Уфимский университет науки и технологий»

В работе исследуется эффективность совместного применения предобуславливателей типа ILU и AIPS для решения систем линейных уравнений специального вида в задачах фильтрации флюидов в пористой среде, учитывающих изменение напряженно-деформированного состояния породы. Показано, что использование адаптированного под задачу предобуславливателя до 2.8 раз ускоряет решение системы по сравнению со стандартными методами решения, используемыми в пакетах общего назначения. Приведены численные результаты распараллеливания полученного метода решения на многоядерные системы.

Ключевые слова: итерационные методы, предобуславливатели на основе неполного LU разложения, предобуславливатели на основе степенного разложения обратной матрицы AIPS, безматричное умножение.

1. Введение

В процессе решения задач фильтрации флюидов в пористой среде с учетом изменения напряженно-деформированного состояния породы возникает необходимость решения большого количества систем линейных алгебраических уравнений (СЛАУ) специального вида:

$$\begin{bmatrix} F & E \\ C & B \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ b \end{bmatrix}, \tag{1}$$

где E – единичная матрица, F – заполненная матрица, связанная с уравнениями геомеханики, C и B – разряженные матрицы с одинаковым шаблоном разряженности, т.е. ненулевые элементы находятся в одних и тех же местах. C и B отвечают за уравнения гидродинамики. Блоки в системе (1) обладает следующими свойствами:

- F – строится один раз и не меняется в процессе эволюционного расчета,
- C и B – меняются на каждом шаге по времени и на каждой итерации нелинейного решателя, но шаблон разряженности остается постоянным,
- f_{ij} элементы матрицы F, представимы в следующем виде:

  $$f_{ij} = W\left(|i_x - j_x|, \, |i_y - j_y|\right) G\left(i_y, \, j_y\right) R\left(i_y\right), \tag{2}$$

  где W, G, R – некоторые известные функции, i_x, i_y – координаты узла i в расчетной сетке.

Из первой строки системы (1) следует

$$v = -Fu. \tag{3}$$

Подставляя (3) во второе уравнение системы (1) получаем СЛАУ на неизвестный вектор u:

$$\left(C - BF\right) u = b. \tag{4}$$

Отметим, что практически все итерационные методы решения систем линейных уравнений базируются на операции умножения матрицы на вектор. Поэтому можно не вычислять матрицу $C-BF$, а последовательно вычислять: 1. $q = Cu$; 2. $w = Fu$; 3. $(C-BF)u \equiv q-Bw$. Для удобства обозначим $A = C - BF$.

Далее везде для построения матрицы предобуславливателя на основе неполного LU разложения будем использовать матрицу:

$$\tilde{A} = C - B\tilde{F}, \tag{5}$$

где $\tilde{F}$ – разряженная матрица, полученная из F отбрасыванием элементов $f_{ij}, \quad i \neq j$, таких, что $|f_{ij}| < \varepsilon |f_{ii}|$, $\varepsilon \in (0,\,1)$ – параметр, выбираемый из условия минимизации времени расчета.

В разделе 2 описаны методы предобуславливания, наиболее часто используемые для систем общего вида, и аспекты их параллельной реализации. В разделе 3 описан метод уточнения предобуславливателя на основе степенного разложения обратной матрицы, описаны принципы совмещения предобуславливателей типа ILU и AIPS и возможные сценарии их использования. В разделе 4 описан метод безматричного умножения матрицы F на вектор, показаны пути оптимизации под современные процессоры с архитектурой x86 и ARM. В разделе 5 представлены результаты численных экспериментов и сравнение эффективности предобуславливателей, в том числе в зависимости от количества потоков при расчетах на многоядерных процессорах. Численные результаты показывают, что совместное использование предобуславливателей типа неполного LU разложения для диагональных блоков без перекрытия и степенного разложения обратной матрицы AIPS значительно улучшает сходимость итерационных линейных решателей и позволяет ускорить решение системы (1) до 2.8 раз по сравнению с наиболее часто используемыми универсальными подходами.

2. Предобуславливатели на основе неполного LU разложения для параллельных вычислений

В качестве одной из составных частей предобуславливателей будем использовать матрицы, основанные на неполном LU разложении:

- ILU0 (Incomplete LU factorization with zero fill) – неполное LU разложение без расширения «шаблона разряженности». Выполняется LU-разложение, но при этом отбрасываются все «порождаемые» ненулевые элементы (т.е. остаются только те ненулевые элементы, которые расположены в тех же позициях, где были ненулевые элементы исходной матрицы) [1, 2]. Самый простой и быстрый в построении предобуславливатель в классе ILU, но на сложных задачах может не обеспечивать нужной скорости сходимости либо не приводить к сходимости вообще.

- ILUT (Incomplete LU with Threshold) – факторизация строится на основе числового порога (threshold), который определяет какие ненулевые элементы можно отбросить после вычисления значений в элементах L и U [1, 3].

- BILU0/BILUT (Block ILU0/ILUT) – блочно диагональные варианты неполного LU разложения. Используются для параллельной реализации ILU0/ILUT (рис. 1a), при этом обмены между блоками отсутствуют либо минимальны, поэтому хорошо распараллеливается, но сходимость может значительно ухудшаться с ростом числа исполнительных потоков [1, 4].

- OBILU0/OBILUT (Overlaped Block ILU0/ILUT) – блочно диагональные варианты неполного LU разложения с перекрытием. Размер перекрытия выбирается из условия минимизации влияния элементов матрицы вне перекрытия на внутренние элементы блока (рис. 1б) [5]. Позволяет повысить качество предобуславливателя (меньше итераций требуется для получения решения), но чем больше зона перекрытия, тем сильнее растет сложность построения предобуславливателя.

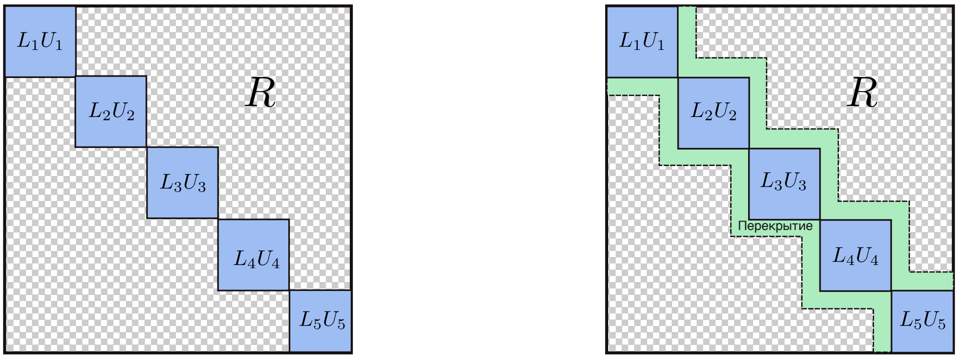

Рис. 1. Примеры декомпозиции матрицы для параллельной ILU факторизации: а) в ILU разложении участвуют диагональные блоки без перекрытия; б) в ILU разложении участвуют блоки большего размера с перекрытием, все блоки раскладываются независимо, но решение берется только по внутренним частям блоков

3. Уточнение предобуславливателя на основе степенного разложения обратной матрицы

Для улучшения сходимости итерационных методов решения СЛАУ (1) и уточнения предобуславливателей типа ILU можно использовать разложение обратной матрицы в ряд Неймана [1, 6]:

$$(E - B)^{-1} = E + B + B^2 + \cdots + B^n + \ldots, \quad \rho(B) < 1, \tag{6}$$

где E – единичная матрица, а $\rho(B)$ – спектральный радиус матрицы B.

Стандартным является представление матрицы A в виде [7]:

$$A = P + R = P(E + P^{-1}R) = P\left(E - (-P^{-1}R)\right), \tag{7}$$

где P – матрица предобуславливателя, R – матрица ошибки. При выполнении условия $\rho(P^{-1}R) < 1$ можно получить следующую формулу обращения предобуславливателя M_N порядка N:

$$M_N^{-1} = \left[E + \sum_{k=1}^{N} (-1)^k \left(P^{-1}R\right)^k\right] P^{-1}. \tag{8}$$

Вычисление M_N^{-1} тем сложнее, чем больше N, но с ростом N улучшается качество предобуславливания $A^{-1} \approx M_N^{-1}$ и снижается количество необходимых для решения линейной системы итераций. В общем случае выбор параметра N проводится эмпирически для каждого типа решаемых задач отдельно. Предобуславливатель M_N, определяемый формулой (8), принято называть AIPS (Approximation of Inverse by Power Series).

Формула (8) удобна, когда матрица R легко определима; если же R неизвестна, то может быть заменена $R = A - P$. Формула (8) в этом случае примет вид:

$$\tilde{M}_N^{-1} = \left[E + \sum_{k=1}^{N} Q^k\right] P^{-1}, \quad Q = E - P^{-1}A. \tag{9}$$

Если матрица предобуславливателя P построена на малой части ненулевых элементов матрицы A, например, используются три главные диагонали [7], то разница в сложности вычислений между формулами (8) и (9) не значительна, матрица R меньше матрицы A на 3 диагонали. С другой стороны, при неполном LU разложении [2] для получения матрицы R необходимо умножить матрицы $L \cdot U$, что трудоемко, при этом в матрице R может оказаться значительно больше ненулевых элементов, чем в матрице A; в этом случае выгоднее использовать формулу (9). Предобуславливатель, полученный по формуле (9), будем называть AIPS2.

Отметим, что алгоритмы построения предобуславливателей AIPS и AIPS2 хорошо подходят для использования на параллельных процессорах, все вычислительные потоки работают независимо, пересылки данных между потоками минимальны.

Большим потенциалом обладает совмещение неполного LU разложения с уточнением предобуславливателя по формулам (8), (9). Построение и обращение предобуславливателей типа ILU существенно последовательны, а их реализации для параллельных вычислений алгебраически не эквивалентны последовательным алгоритмам и требуют больше итераций линейного решателя для достижения сходимости. При этом сходимость линейного решателя ухудшается с ростом количества вычислительных потоков. Определим эффективность параллельной блочной реализации предобуславливателей типа ILU с уточнением AIPS для компенсации ухудшения свойств блочных предобуславливателей типа ILU при росте числа вычислительных потоков.

4. «Безматричное» умножение матрицы F на вектор

Поскольку блок-матрица F плотно заполненная, то при росте размерности сложность умножения матрицы F на вектор растет как $O(n^2)$, где n – размерность матрицы. Остальные блок-матрицы – разреженные с ограниченным количеством ненулевых элементов в строке, и сложность их умножения на вектор растет как $O(n)$. Соответственно, с ростом размерности операции с матрицей F становятся доминирующими и требуют особого внимания. В литературе и пакетах программ известны «безматричные» подходы к умножению матрицы на вектор [8, 9], в этом случае формирование матрицы как единого массива значений не происходит. Обычно это выгодно, когда матрица формируется из структурированных массивов меньшей размерности; в этом случае можно формировать матрицу «на лету» маленькими блоками. Такой подход обычно требует кратно больше арифметических операций для построения результата умножения матрицы на вектор, но значительно сокращает требования к объему оперативной памяти. Если же удается использовать векторные инструкции, например, AVX2/AVX512 [10, 11] для процессоров x86 или NEON/SVE [12, 13] для процессоров ARM, то общая скорость операции может значительно вырасти. Так же положительно на скорости вычислений сказывается локализация данных и использование КЭШ памяти процессора.

Рассмотрим матрицу F более подробно. Согласно формуле (2) на расчетной сетке размером $n = n_x \times n_y$ (рис. 2) для построения матрицы F надо вычислить W в $n_x \times n_y$ точках, G – в $n_y \times n_y$ точках, а R – в n_y точках.

Используем i, j, k в качестве индексов строк и столбцов в матрице F. Обозначим i_x, i_y – координаты в расчетной сетке по оси X и Y, соответствующие узлу i. Таким образом $i = i_x + i_y \times n_x$, аналогично для j и k: $j = j_x + j_y \times nx$, $k = k_x + k_y \times n_x$.

Введем следующие обозначения:

$$\Delta_{ij} = (|i_x - j_x|, \, |i_y - j_y|)$$

$$W_{\Delta_{ij}} = W\left(|i_x - j_x|\,, \, |i_y - j_y|\right),$$

$$G_{ij} = G\left(i_y, \, j_y\right),$$

$$R_i = R\left(i_y\right).$$

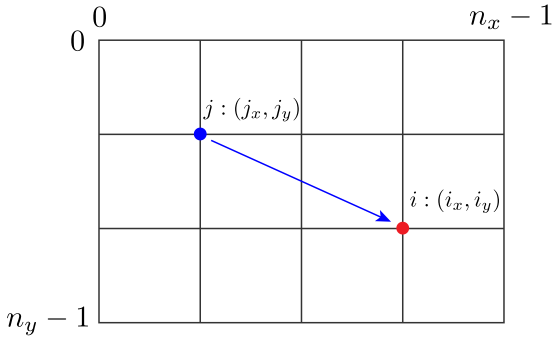

Рис. 2. Расчетная сетка размером n_x узлов вдоль горизонтальной оси X и n_y вдоль вертикальной оси Y. Расстояние между узлами по оси X равно ΔX, а по оси Y – ΔY

Внутренняя структура матрицы F по формуле (2) позволяет записать умножение i-строки матрицы на вектор в виде:

$$u_i = F_i w = \sum_{k=0}^{n-1} f_{ik} w_k = R_i \sum_{k=0}^{n-1} W_{\Delta_{ik}} G_{ik} w_k. \tag{10}$$

Использование формулы (10) позволяет сократить требования к оперативной памяти. Например, для размеров сетки $n_x = 100$, $n_y = 100$ для хранения матрицы F необходимо 800 МБайт оперативной памяти, с другой стороны при использовании формулы (10) для $W_{\Delta_{ij}}$ необходимо 80 КБайт, для G_{ij} – 80 КБайт, а для R_i – 800 Байт, т.е. приблизительно в 5 000 раз меньше. Снижение требований к оперативной памяти позволяет разместить данные в КЭШ памяти процессора, что значительно ускоряет доступ к этим данным. При этом требуемое количество арифметических операций возрастает приблизительно в 2 раза.

Результаты сравнения скорости расчета с использованием обычного алгоритма умножения плотной матрицы на вектор и по уравнению (10) показывают, что наблюдается замедление расчета в более чем 20 раз. Это не может быть связано только с возрастанием количества арифметических операций, формула (10) требует всего в 2 раза больше арифметических операций по сравнению с обычным умножением.

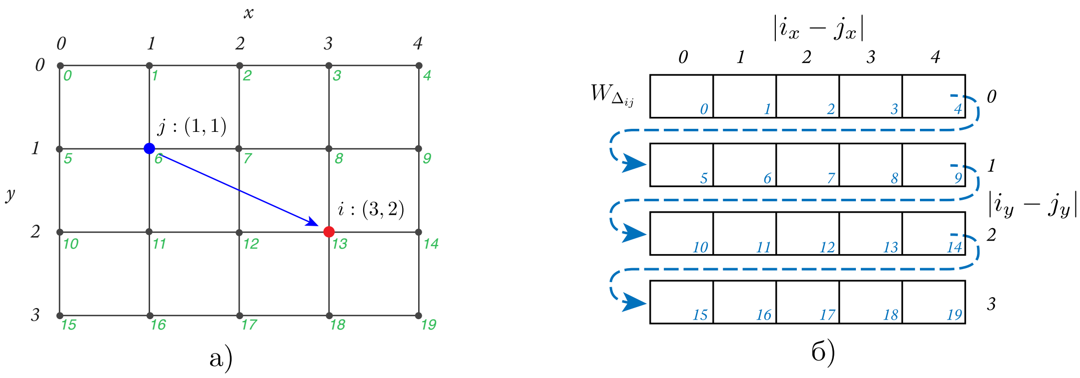

Рис. 3. Пример расчетной сетки ($n_x = 5$, $n_y = 4$). Зеленым выделены индексы узлов расчетной сетки в матрице F (узел $j = 6$, координаты $j_x = 1, j_y = 1$, узел $i = 13$, координаты $i_x = 3$, $i_y = 2$). На рисунке б представлено расположение элементов $W_{\Delta_{ij}}$ в оперативной памяти, синим цветом выделены индексы $l(i,j) = |i_x - j_x| + n_x \times |i_y - j_y|$ в линейном массиве

Причиной значительного замедления работы является непоследовательный доступ к данным, что приводит к большому количеству «КЭШ промахов» и затрудняет для процессора предсказание ветвлений.

Для примера рассмотрим сетку ($n_x = 5$, $n_y = 4$) (рис. 3а). Размерность матрицы F – $n = n_x \times n_y = 20$. Для хранения $W_{\Delta_{ij}}$ используется одномерный массив длиной $n_x \times n_y$ с индексированием согласно формуле $l(i,j) = |i_x - j_x| + nx \times |i_y - j_y|$ (рис. 3б). Рассмотрим умножение строки матрицы, соответствующей элементу сетки $i = 13$ с координатами $i_x = 3$, $i_y = 2$ (красная точка на рис. 3а) на вектор. Согласно формуле (10) последовательность доступа к массиву $W_{\Delta_{ik}}$ представлена в таблице 1.

Таблица 1. Пример изменения Δ_{ik} и линейного индекса $l(i,k)$ массива $W_{\Delta_{ik}}$ при расчете u_{13} согласно формуле (10) $i = 13$ на сетке $n_x = 5$, $n_y = 4$. Отметим, что k пробегает все значения от 0 до $n - 1 = 19$

k	0	1	2	3	4	5	6	7	8	9
(k_x, k_y)	(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)	(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)
Δ_{ik}	(3, 2)	(2, 2)	(1, 2)	(0, 2)	(1, 2)	(3, 1)	(2, 1)	(1, 1)	(0, 1)	(1, 1)
$l(i,k)$ в массиве $W_{\Delta_{ik}}$	**13**	**12**	**11**	**10**	**11**	**8**	**7**	**6**	**5**	**6**
k	10	11	12	13	14	15	16	17	18	19
(k_x, k_y)	(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)	(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)
Δ_{ik}	(3, 0)	(2, 0)	(1, 0)	(0, 0)	(1, 0)	(3, 1)	(2, 1)	(1, 1)	(0, 1)	(1, 1)
$l(i,k)$ в массиве $W_{\Delta_{ik}}$	**3**	**2**	**1**	**0**	**1**	**8**	**7**	**6**	**5**	**6**

Решением проблемы с непоследовательным доступом к памяти является расширение $W_{\Delta_{ij}}$ путем зеркального отображения по оси X и Y (рис. 4).

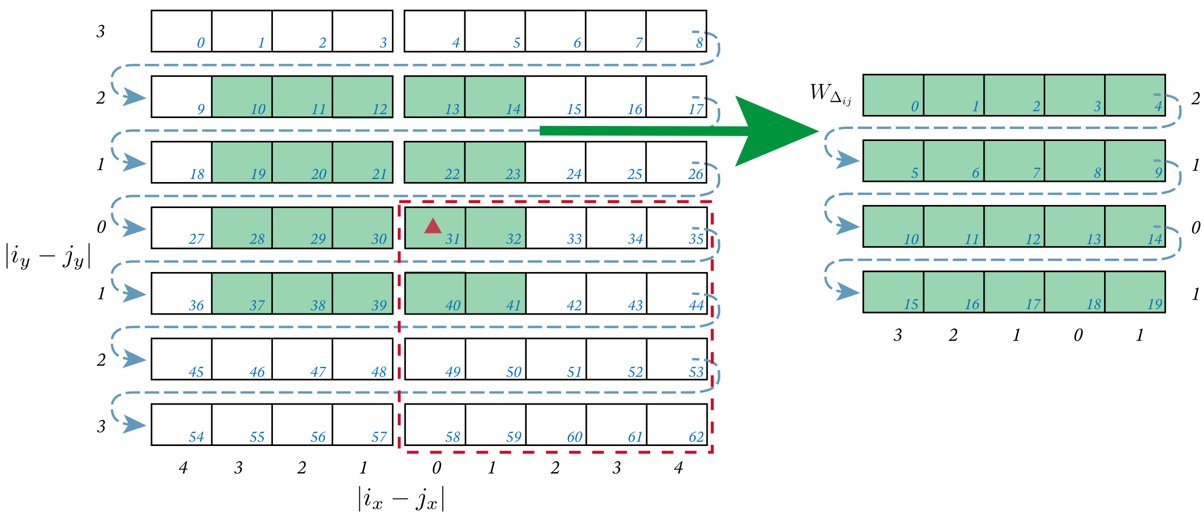

Рис. 4. Пример расширенного массива элементов $W_{\Delta_{ij}}$ в оперативной памяти, синим цветом выделены индексы l в линейном массиве. Зеленым выделена часть, необходимая для умножения строки матрицы, соответствующая узлу $i = 13$ с координатами $i_x = 3$, $i_y = 2$. Красным пунктиром обозначен исходный (до расширения) массив $W_{\Delta_{ij}}$. Элемент массива $W_{\Delta_{ij}}$ с красным треугольником – центр симметрии, т.е. элементы расширенного массива $W_{\Delta_{ij}}$ с индексами 30 и 32 совпадают, аналогично 51 и 11, 49 и 13, и т.д.

Ускорение достигается за счет использования векторных инструкций из расширения AVX2 [11] для архитектуры x86 и NEON [12] для архитектуры ARM. AVX2 позволяет обрабатывать блоки матрицы 8×8 для одинарной точности (FP32), и 4×4 для двойной точности (FP64). Реализация с использованием NEON оперирует с блоками матрицы 4×4 независимо от точности.

Итоговое ускорение относительно классического метода составило от 2 раз на матрицах размерности 500, до 3.5 раз на матрицах размерности 40 000. В дальнейшем во всех численных результатах используется «безматричное» умножение F.

5. Численные результаты

Для тестирования в качестве линейного решателя будем использовать предобусловленный метод BiCGStab [14, 15]. BiCGStab является одним из универсальных и наиболее часто используемым методом для решения разряженных несимметричных матриц.

Тестовый пример: решение задачи фильтрации флюида в деформируемой поровой среде. Всего решаемых матриц 5432, размерность матриц от 53 до 10 567, количество ненулевых элементов в матрице для предобуславливателя $\tilde{A}$ вида (5) максимум 501 011.

В качестве тестового стенда использован ноутбук Apple MacBook Pro с процессором Apple M4 Pro, архитектура ARM, 10 быстрых ядер 4.5 ГГц, пропускная способность оперативной памяти 273 ГБайт/секунду.

Будем рассматривать следующие варианты предобуславливателей:

- OBILU0 – блочный алгоритм неполного LU разложения с перекрытием. Матрица A делится на диагональные блоки по количеству параллельных потоков вычисления, блоки перекрываются (рис. 1б). Размер перекрытия выбирается так, чтобы элементы вне перекрытия не влияли на внутреннюю часть блоков;

- OBILUT – блочный алгоритм неполного LU разложения с перекрытиями (рис. 1б) и разложением в модификации ILUT [3], величина параметра $\tau = 1e^{-5}$ – при разложении матрица $\tilde{A}$ отбрасываются все элементы $\tilde{A}_{ij}$ такие, что $|\tilde{A}_{ij}| < \tau||\tilde{A}_i||_2$, где $\tilde{A}_i$ – i строка матрицы $\tilde{A}$;

- BILU0+AIPS2 – блочный алгоритм неполного LU разложения без перекрытия (рис. 1а). Для улучшения сходимости используем AIPS2 первого порядка $N = 1$ (9).

Результаты тестовых расчетов приведены в таблице 2. Как видно для системы (1) более выгоден подход BILU0+AIPS2, ускорение относительно стандартно используемых методов OBILU0 и OBILUT составляет 2.78 и 3.25 раза соответственно. Вместе с тем для подхода BILU0+AIPS2 ускорение на этапе построения предобуславливателя (Setup) составляет 6.41 на 8 вычислительных потоках, несколько хуже чем 7.65 раза для OBILU0. Данный результат удивителен, так как BILU0+AIPS2 не используется перекрытие блоков и должен ускоряться лучше OBILU0, этот факт требует дополнительных исследований. Видно, что для BILU0+AIPS2 фаза построения предобуславливателя занимает минимальное время, потому что AIPS2 не требует затрат на построение, а BILU0 проще строить, чем OBILU0, так как нет перекрытия блоков.

6. Выводы

Проведено исследование эффективности применения различных типов предобуславливателей для решения систем линейных алгебраических уравнений, возникающих в процессе решения задач фильтрации флюидов в пористой среде с учетом изменения напряженно-деформированного состояния породы. Показано, что совместное использование блочных предобуславливателей типа неполного LU разложения совместно с разложением обратной матрицы в степенной ряд позволяет ускорить поиск решения системы линейных уравнений до 2.8 раз по сравнению со стандартными, универсальными методами предобуславливания.

Дополнительно приведены подходы к использованию и оптимизации «безматричного» умножения плотной матрицы на вектор. Показано, что при наличии дополнительных знаний о построении плотной матрицы возможно значительно сократить до 50 раз требования к объему оперативной памяти, а при использовании векторных инструкций и локализации данных в КЭШ процессора – увеличить скорость операции умножения матрицы на вектор до 3.5 раза.

Таблица 2. Результаты численных расчетов. Всего матриц 5 432, размерность матриц от 53 до 10 567. **Type** – тип предобуславливателя, **NT** – количество вычислительных потоков, **Iters** – общее число итераций линейного решателя BiCGStab на решение всех матриц, **Setup** – время построения предобуславливателя в секундах, **Solve** – время решения систем в секундах, **Total** – общее время в секундах (**Setup** + **Solve**), **Speedup setup** – ускорение построения предобуславливателя, **Speedup solve** – ускорение решения систем

Type	NT	Iters	Setup (c.)	Solve (c.)	Total (c.)	Speedup setup	Speedup solve
OBILU0	1	227435	99.49	238.34	337.83	1.00	1.00
OBILU0	2	249323	49.69	145.58	195.27	2.00	1.64
OBILU0	4	258346	25.27	97.06	122.33	3.94	2.46
OBILU0	8	273876	13.01	75.91	88.92	7.65	3.14
OBILUT	1	93766	248.31	186.93	435.23	1.00	1.00
OBILUT	2	128316	120.62	117.39	238.01	2.06	1.59
OBILUT	4	160392	59.31	81.76	141.07	4.19	2.29
OBILUT	8	168459	39.39	64.42	103.81	6.30	2.90
BILU0+AIPS2	1	43004	52.06	98.77	**150.83**	1.00	1.00
BILU0+AIPS2	2	44324	26.36	56.18	**82.54**	1.97	1.76
BILU0+AIPS2	4	44891	13.90	34.58	**48.48**	3.75	2.86
BILU0+AIPS2	8	45206	8.12	23.82	**31.94**	6.41	4.15

Литература

1. Saad Y. Iterative Methods for Sparse Linear Systems, Second Edition. SIAM, Philadelphia, USA, 2003. 567 p. DOI: 10.1137/1.9780898718003.

2. Meijerink J.A., van der Vorst H.A. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix // Math. Comp. 1977. Vol. 31. P. 148–162. DOI: 10.2307/2005786.

3. Saad Y. ILUT: A Dual Threshold Incomplete LU Factorization // Numerical Linear Algebra with Applications. 1994. Vol. 1, no. 4. P. 387–402. DOI: 10.1002/nla.1680010405.

4. Smith B., Bjorstad P., Gropp W. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University Press, 2004. 237 p.

5. de Sturler E. Incomplete block LU preconditioners on slightly overlapping subdomains for a massively parallel computer // Applied Numerical Mathematics. 1995. Vol. 19, no. 1-2. P. 129–146. DOI: 10.1016/0168-9274(95)00077-8.

6. Chen K. Matrix Preconditioning Techniques and Applications. Cambridge Univ. Press, Cambridge, 2005. 564 p. DOI: 10.1017/CBO9780511543258.

7. Юлдашев А.В., Репин Н.В., Спеле В.В. Параллельный предобуслаливатель на основе степенного разложения обратной матрицы для решения разреженных линейных систем на графических процессорах // Вычислительные методы и программирование. 2019. Т. 20, № 4. С. 444–456. DOI: 10.26089/NumMet.v20r439.

8. Kronbichler M., Kormann K. A generic interface for parallel cell-based finite element operator application// Computers Fluids. 2012. Vol. 63. P. 135–147. DOI: 10.1016/j.compfluid.2012.04.012.

9. Arndt D., Bangerth W., Davydov D. et al. The deal.II Library, Version 9.3 // Journal of Numerical Mathematics. 2021. Vol. 29, no. 3. DOI: 10.1515/jnma-2021-0081.

10. Xie B., Zhan J., Liu X. et al. CVR: Efficient Vectorization of SpMV on X86 Processors // Proceedings of 2018 IEEE/ACM International Symposium on Code Generation and Optimization (CGO'18). ACM, New York, NY, USA, 2018. DOI: 10.1145/3168818.

11. Intel Advanced Vector Extensions 10.2, Architecture Specification. 2025. URL: https://cdrdv2.intel.com/v1/dl/getContent/828965 (дата обращения: 18.02.2025).

12. NEON Programmer's Guide. Version: 1.0. 2013. URL: https://developer.arm.com/documentation/den0018/a/?lang=en (дата обращения: 23.02.2025).

13. SVE Programming Examples. 2024. URL: https://developer.arm.com/documentation/dai0548/latest/ (дата обращения: 23.02.2025).

14. Barrett R., Berry M., Chan T.F. et al. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods // Mathematics of Computation. 1996. Vol. 64, no. 211. DOI: 10.2307/2153507.

15. van der Vorst H.A. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems // SIAM Journal on Scientific and Statistical Computing. 1992. Vol. 13, no. 2. DOI: 10.1137/0913035.