ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ РАСПАРАЛЛЕЛИВАНИЯ РЕШЕНИЯ НЕКОТОРЫХ РАЗРЕЖЕННЫХ СЛАУ НА МНОГОЯДЕРНЫХ ПРОЦЕССОРАХ

И.И. Газизов, А.В. Юлдашев

Введение

Решение систем линейных алгебраических уравнений с разреженной матрицей (разреженных СЛАУ) является вычислительным ядром множества задач математического моделирования. В нашей работе исследуется эффективность распараллеливания решения разреженных СЛАУ, возникающих в задаче численного моделирования многофазных фильтрационных потоков в нефтегазовых пластах. Исследуются возможности ускорения предобусловленного итерационного метода бисопряженных градиентов со стабилизацией (BiCGStab) с блочным предобуславливателем на основе неполного LU-разложения без заполнения (ILU0). Рассмотрена эффективность распараллеливания наиболее ресурсоемких операций данного метода на многоядерных процессорах Intel Xeon Sandy Bridge и графических процессорах NVIDIA Tesla Kepler. Вычислительные эксперименты проведены средствами специализированного решателя разреженных СЛАУ собственной разработки, который основан на использовании библиотеки Intel Math Kernel Library, а также библиотек NVIDIA cuSPARSE и cuBLAS, реализующих базовые операции линейной алгебры.

Методы решения разреженных СЛАУ

Наиболее популярными и быстрыми методами решения разреженных систем большой размерности считаются предобусловленные методы подпространства Крылова. Предобуславливание предполагает переход от решения исходной системы $A\,x = b$ к решению эквивалентной системы с матрицей, обладающей улучшенными спектральными характеристиками. Так при левом предобуславливании исходная СЛАУ сводится к эквивалентной системе $M^{-1}A\,x = M^{-1}b$, где M - матрица предобуславливателя. При этом известно, что скорость сходимости метода напрямую зависит от выбора предобуславливателя.

В нашей работе для решения разреженных СЛАУ применяется предобусловленный метод бисопряженных градиентов со стабилизацией, пригодный для решения СЛАУ с несимметричной матрицей. В качестве предобуславливателя используется неполное LU-разложение без заполнения: $M = LU = A - R$, где L и U – нижне- и верхнетреугольные матрицы соответственно, R – невязка или ошибка неполного LU-разложения [1]. Необходимо отметить, что использование неполного LU-разложения является одной из популярных техник предобуславливания. В частности, при решении СЛАУ, возникающих в задаче гидродинамического моделирования процессов нефтегазодобычи, ILU0 используется либо как самостоятельный предобуславливатель либо в составе специализированного двухступенчатого предобуславливателя CPR [2,3].

Рассмотрим алгоритм предобусловленного метода BiCGStab [4]:

1. Φ_0 – вектор начального приближения решения;
2. $r_0 = b - A\Phi_0$;
3. $\overline{r}_0$ – произвольный вектор, такой, что $(\overline{r}_0, r_0) \neq 0$;
4. $\rho_0 = \alpha = \omega_0 = 1$;
5. $v_0 = p_0 = 0$;
6. для $k = 1,2,3,\ldots$
7. $\quad \rho_k = \left(\overline{r}_0, r_{k-1}\right)$; $\beta = \left(\frac{\rho_k}{\rho_{k-1}}\right)\left(\frac{\alpha}{\omega_{k-1}}\right)$;
8. $\quad p_k = r_{k-1} + \beta\left(p_{k-1} - \omega_{k-1}v_{k-1}\right)$;
9. $\quad$ определение вектора y из решения системы $My = p_k$;
10. $\quad v_k = Ay$;
11. $\quad \alpha = \frac{\rho_k}{\left(\overline{r}_0, v\right)}$;
12. $\quad s = r_{k-1} - \alpha v_k$;
13. $\quad$ определение вектора z из решения системы $Mz = s$;
14. $\quad t = Az$;
15. $\quad w_k = \frac{(t,s)}{(t,t)}$
16. $\quad \Phi_k = \Phi_{k-1} + \alpha y + \omega_k z$;
17. $\quad$ если Φ_k достигло требуемой точности – выход из цикла;
18. $\quad r_k = s - \omega_k t$;
19. конец цикла по k

Как видно из представленного алгоритма при использовании предобусловленного метода BiCGStab наиболее трудоемкими операциями являются умножение разреженной матрицы на вектор, а также решение систем с матрицей предобуславливателя. С учетом того, что в качестве предобуславливателя у нас используется ILU0, решение системы с матрицей предобуславливателя сводится к решению двух систем с нижне- и верхнетреугольными матрицами.

Таким образом, основными операциями, требующими ускорения, являются:

- построение предобуславливателя ILU0;
- решение треугольных систем;
- умножение разреженной матрицы на вектор.

В основу разработанных нами программных реализаций для CPU и GPU были положены существующие оптимизированные библиотеки от производителей процессоров Intel и NVIDIA: Intel Math Kernel Library (MKL), ориентированная на многоядерные системы с общей памятью, для CPU и NVIDIA cuSPARSE и cuBLAS из состава CUDA Toolkit для GPU. В составе данных библиотек имеются все необходимые функции, для реализации предобусловленного метода BiCGStab, в том числе функции, реализующие перечисленные выше операции, а также различные векторные операции: вычисление нормы вектора, скалярное произведение векторов, линейная комбинация векторов.

Подходы к распараллеливанию

Операция умножения разреженной матрицы на вектор, которая является базовой операцией множества итерационных методов, алгоритмически достаточно легко распараллеливается. Однако основным препятствием к достижению высокой производительности при выполнении данной операции для разреженных матриц на современных вычислительных системах является интенсивный и как, правило, нерегулярный доступ к памяти. В связи с этим производительность данной операции лимитируется скоростными характеристиками доступа к памяти. Для центральных процессоров архитектуры x86 производительность умножения разреженной матрицы на вектор может быть существенно выше при условии, что вектор, на который умножается разреженная матрица, помещается в кэш-память. Графические процессоры справляются с данной операцией значительно быстрее благодаря относительно высокой пропускной способности оперативной памяти GPU, а также его массивно-параллельной архитектуре, при условии достаточного ресурса параллелизма, который определяется размерностью разреженной матрицы и плотностью ее заполнения.

Отметим, что в используемых нами библиотеках имеются параллельные реализации данной операции с поддержкой различных форматов хранения разреженной матрицы в памяти. Причем в нашей работе использован популярный формат хранения со сжатием по строкам – CSR.

В отличие от умножения разреженной матрицы на вектор, построение предобуславливателя ILU0, а также решение возникающих в результате неполного LU-разложения систем с разреженными нижне- и верхнетреугольными матрицами в классической постановке обладают значительно меньшим ресурсом параллелизма. Вероятно, в связи с этим в библиотеке MKL данные операции реализованы последовательно. Очевидно, что исполнение программы на массивно-параллельном графическом процессоре в последовательном режиме неприемлемо, т.к. производительность одного ядра графического процессора крайне низкая. Поэтому библиотека cuSPARSE содержит реализации математически эквивалентных алгоритмов построения предобуславливателя ILU0 и решения треугольных систем уравнений на основе разделения матрицы на уровни, в рамках которых возможна параллельная обработка [5,6]. Использование данного подхода предполагает проведение рассматриваемых операций в два этапа:

- этап анализа;
- этап факторизации или решения треугольных систем.

На этапе анализа формируются уровни, содержащие строки матрицы, которые могут обрабатываться независимо между собой. На втором этапе происходит последовательная обработка по сформированным уровням. Несмотря на то, что уровни матриц обрабатываются последовательно друг за другом, на каждом уровне открываются возможности параллельной обработки множества строк. Таким образом, чем меньше выявлено уровней на этапе анализа, тем более эффективно могут быть распараллелены рассматриваемые операции. Необходимо отметить, что т.к. портрет матрицы, на основе которой строится предобуславливатель ILU0, и полученных нижне- и верхнетреугольных матриц совпадают в силу природы ILU0, этап анализа проводится однократно и позволяет провести подготовку к выполнению обеих операций.

Матрицы СЛАУ, используемые в нашем исследовании получены в ходе численного моделирования многофазных фильтрационных потоков в нефтегазовых пластах на основе неявной разностной схемы (Fully Implicit). Здесь отметим их некоторые ключевые особенности. При использовании полностью неявной схемы в зависимости от количества фаз на одну ячейку компьютерной модели приходится до 3 неизвестных параметров. Структура матрицы, получаемой в результате дискретизации, близка к блочно-ленточной с размерностью блока равной количеству фаз. Основная (регулярная) часть матрицы является несимметричной и сильно разреженной: для трехфазной фильтрации при использовании 7-точечного шаблона дискретизации в строках имеем не более 21 ненулевого элемента. Однако учет скважин в реальных моделях нефтегазовых месторождений приводит к

наличию дополнительных зависимостей между параметрами в ячейках, поэтому в матрице A появляется, так называемая, нерегулярная часть.

Необходимо отметить, что построение матрицы предобуславливателя в нашей работе проводится не на основе матрицы A, соответствующей решаемой СЛАУ, а полученной из нее путем дополнительного разреживания, которое сводится к отбрасыванию ненулевых элементов, не содержащихся в портрете регулярной части. Данный шаг несколько ухудшает качество предобуславливания, но в то же время ускоряет операции построения матрицы предобуславливателя и решения треугольных систем, особенно на графических процессорах, т.к. количество уровней матрицы, получаемых на этапе анализа оказывается значительно меньше.

Несмотря на приведенные подходы к распараллеливанию базовых операций метода BiCGStab с предобуславливателем ILU0 проведенные эксперименты показали, что для получения хорошо масштабируемого на современных многоядерных архитектурах параллельного алгоритма решения СЛАУ необходимо повышение ресурса параллелизма в алгоритме. Для этого предлагаются различные блочные модификации предобуславливателя ILU0 [1,7].

В нашей работе рассмотрен простейший подход для построения блочного предобуславливателя – матрица A', на основе которой строится предобуславливатель, разбивается на диагональные квадратные блоки размерности близкой к $n=\frac{N(A')}{nb}$, где $N(A')$ – размерность матрицы, nb – количество блоков. Элементы матрицы, которые не попали в блоки, отбрасываются. В результате неполное LU-разложение можно проводить не для всей матрицы в целом, а независимо для каждого блока, что дает возможность крупноблочного распараллеливания операций построения предобуславливателя, а также решения треугольных систем. Минусы данного подхода очевидны: вследствие отбрасывания части элементов может ухудшиться сходимость метода, однако, как правило, при правильном разбиении матрицы на блоки, некоторое ухудшение сходимости компенсируется ускорением, которое может быть получено в результате параллельной обработки матричных блоков.

На многоядерных системах параллельная обработка блоков при выполнении операций предобуславливания и решения треугольных систем позволяет использовать все вычислительные ядра вычислительной системы на всех этапах решения СЛАУ, что реализовано у нас средствами технологии OpenMP. С точки зрения использования GPU рассмотренный подход позволяет задействовать несколько графических процессоров, что в свою очередь позволяет проводить решение СЛАУ большей размерности.

Экспериментальная часть

В данном разделе представлены результаты исследования эффективности распараллеливания двух разработанных программ, реализующих итерационный метод BiCGStab с блочным предобуславливателем на основе ILU0, описанным выше. Первая программа предназначена для работы на многоядерных CPU Intel и опирается на функции библиотеки MKL для выполнения базовых операций метода решения СЛАУ. Вторая – предназначена для гибридных систем с GPU NVIDIA и основывается на функциях библиотеки cuSPARSE. Параллельная обработка блоков предобуславливателя в обеих программах проводится в многопоточном режиме благодаря использованию интерфейса OpenMP, причем число потоков выбирается равным числу блоков при построении предобуславливателя. При тестировании программы для CPU число блоков варьировалось от 1 до 12, а при работе на GPU число блоков выбиралось равным 1 или 2. Все расчеты проводились с двойной точностью на гибридном вычислительном узле с двумя CPU Intel Xeon E5-2680 и двумя GPU NVIDIA Tesla K20X, графические процессоры работали в режиме ECC Off.

Таблица 1 содержит основные характеристики матриц, задействованных в ходе проведенного исследования. Матрицы в таблице упорядочены по возрастанию размерности, а также, как следствие, количеству ненулевых элементов в регулярной части матриц.

Таблица 1. Основные характеристики матриц.

Матрица	Размерность	Кол-во ненулевых элементов во всей матрице	Кол-во ненулевых элементов в регулярной части матрицы	Среднее кол-во элементов в строке во всей матрице	Среднее кол-во элементов в строке в регулярной части матрицы	Размер вектора правой части, МБайт
1	1 330 503	25 261 821	25 245 027	18,99	18,97	10,15
2	1 330 503	25 333 677	25 245 027	19,04	18,97	10,15
3	1 350 000	27 922 950	27 837 000	20,68	20,62	10,30
4	1 500 000	55 815 624	30 519 000	37,21	20,35	11,44
5	2 304 102	42 859 314	42 775 866	18,60	18,57	17,58
6	4 320 921	85 471 137	85 471 137	19,78	19,78	32,97
7	6 610 263	118 221 633	118 219 743	17,88	17,88	50,43

Ниже представлены таблицы, содержащие времена выполнения базовых операций на CPU Intel.

Таблица 2. Построение предобуславливателя ILU0 на CPU.

Матрица	Время, с					Максимальное ускорение
	1 ядро	2 ядра	4 ядра	8 ядер	12 ядер	
1	1,355	0,636	0,317	0,159	0,113	11,991
2	1,301	0,639	0,315	0,159	0,113	11,513
3	1,549	0,756	0,37	0,192	0,139	11,144
4	1,668	0,808	0,398	0,207	0,148	11,270
5	2,13	1,057	0,542	0,28	0,204	10,480
6	4,817	2,37	1,199	0,608	0,415	11,607
7	13,673	6,785	3,454	1,753	1,179	11,597

Таблица 2 демонстрирует устойчивый рост времени построения предобуславливателя с увеличением размерности матриц и соответственно количества ненулевых элементов в регулярной части матриц (рисунок 1). Необходимо отметить заметное замедление выполнения данной операции на матрице 7 с максимальной размерностью. Максимальное ускорение, достигнутое на 12 ядрах, близко к идеальному благодаря полному отсутствию зависимостей по данным при построении неполных LU-разложений в каждом из блоков.

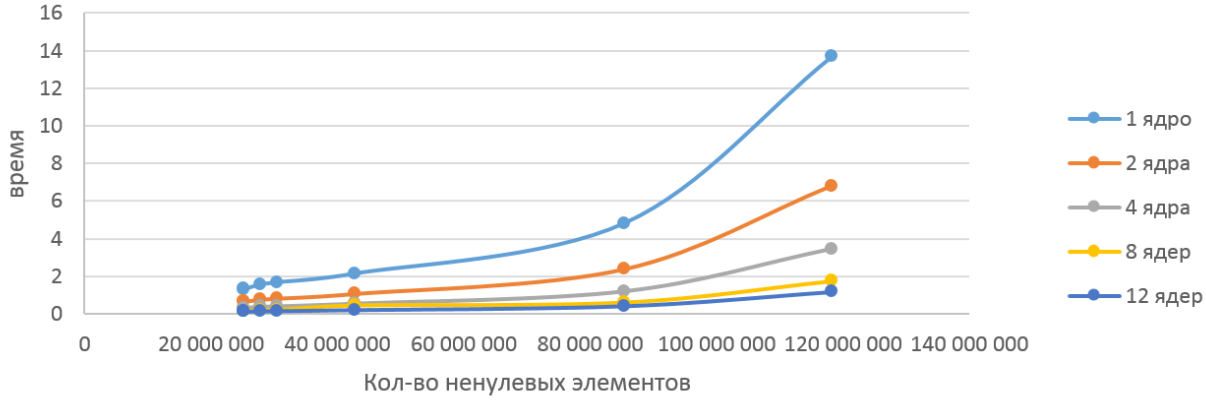

Рис. 1. Зависимость времени построения предобуславливателя на CPU от количества ненулевых элементов в регулярной части матрицы.

В таблице 3 приведены времена решения нижне- и верхнетреугольных систем, которые решаются независимо для каждого из блоков, что позволяет получить достаточно высокое ускорение на многоядерных CPU. Максимальное достигнутое ускорение оказывается несколько ниже, чем для предыдущей операции, что, скорее всего, связано, с более высокой интенсивностью обращений в память при решении треугольных систем относительно операции построения предобуславливателя.

Таблица 3. Решение треугольных систем на CPU.

Матрица	Время, с					Максимальное ускорение
	1 ядро	2 ядра	4 ядра	8 ядер	12 ядер	
1	0,099	0,046	0,021	0,012	0,010	9,552
2	0,099	0,046	0,021	0,012	0,010	9,478
3	0,106	0,050	0,022	0,013	0,011	9,484
4	0,118	0,055	0,025	0,014	0,014	8,429
5	0,166	0,071	0,038	0,021	0,018	9,222
6	0,325	0,138	0,073	0,039	0,032	10,156
7	0,845	0,442	0,228	0,113	0,090	9,389

На рисунке 2 видно, что время выполнения рассматриваемой операции монотонно растет с увеличением количества ненулевых элементов в регулярной части исходной матрицы СЛАУ.

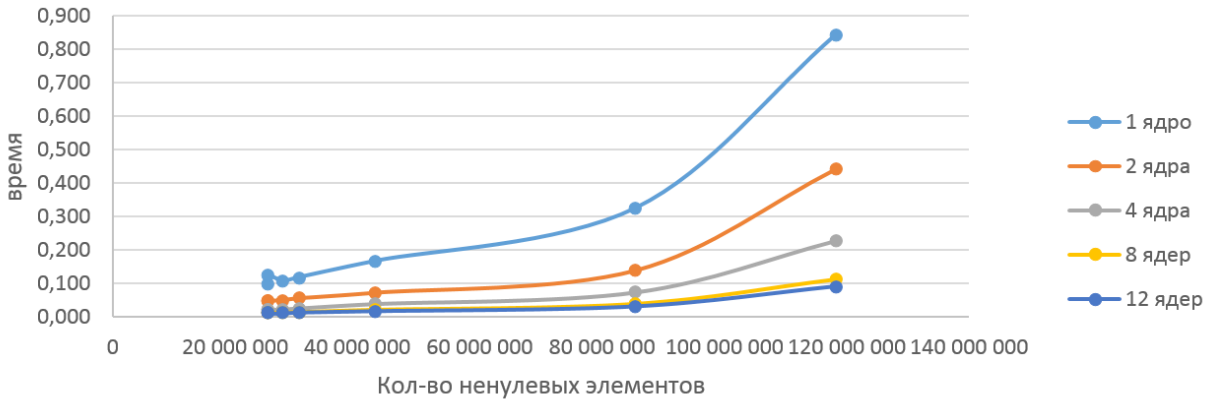

Рис. 2. Зависимость времени решения треугольных систем от количества ненулевых элементов в регулярной части матрицы на CPU

Операция умножения разреженной матрицы на вектор является наиболее легковесной из всех рассмотренных базовых операций, однако, показывает наименьшую эффективность распараллеливания в силу ограничений пропускной способности памяти многоядерной вычислительной системы. Время выполнения операции монотонно возрастает с увеличением размерности матриц (см. таблицу 4 и рисунок 3), но не количества ненулевых элементов в рассмотренных матрицах (см., например, характеристики матриц 4 и 5).

Таблица 4. Умножение разреженной матрицы на вектор на CPU.

Матрица	Время, с					Максимальное ускорение
	1 ядро	2 ядра	4 ядра	8 ядер	12 ядер	
1	0,044	0,021	0,012	0,010	0,010	4,179
2	0,044	0,021	0,012	0,010	0,010	4,315
3	0,047	0,022	0,013	0,011	0,011	4,161
4	0,074	0,036	0,020	0,018	0,018	4,238
5	0,087	0,041	0,025	0,023	0,023	3,752
6	0,145	0,069	0,043	0,038	0,033	4,393
7	0,395	0,207	0,107	0,072	0,071	5,573

Заметно, что также как и для предыдущих операций наблюдается непропорциональное замедление при работе с матрицей 7. Это, вероятно, связано с тем, что вектор, на который умножается матрица не помещается полностью в кэш-память последнего уровня ни одного ни двух CPU Intel Xeon E5-2680, в связи с чем повышается вероятность промахов при обращении в кэш-память и увеличивается количество обращений в более медленную оперативную память.

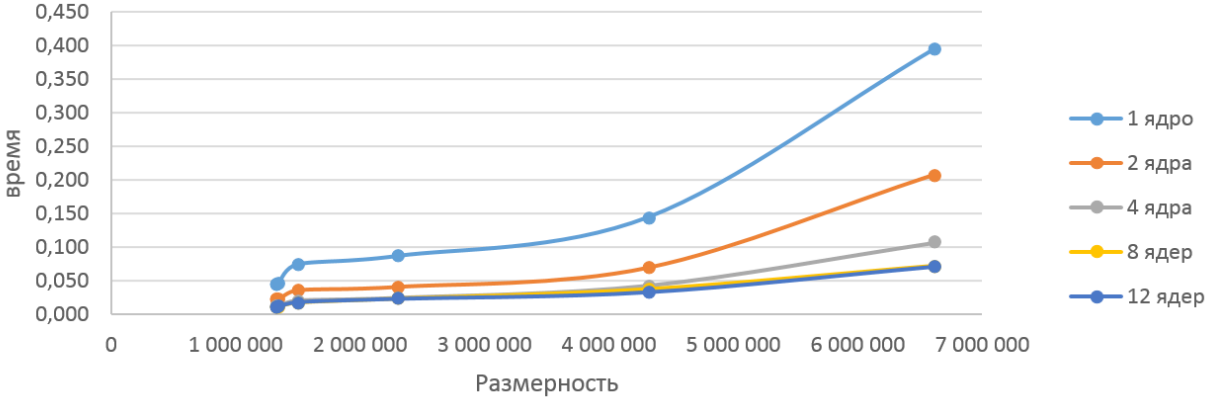

Рис. 3. Зависимость времени умножения разреженной матрицы на вектор от размерности на CPU.

Ниже представлены таблицы, содержащие времена выполнения базовых операций на GPU NVIDIA. Времена приведены без учета копирования входных-выходных данных в память графических процессоров.

Таблица 5. Построение предобуславливателя ILU0 на GPU.

Матрица	Время, с	Ускорение

	1 GPU	2 GPU	
1	0,134	0,075	1,787
2	0,133	0,074	1,797
3	0,151	0,085	1,776
4	0,177	0,101	1,752
5	0,232	0,131	1,771
6	0,414	0,217	1,908
7	0,601	0,333	1,805

Из таблицы 5 и рисунка 4 видно, что в отличие от CPU время построения предобуславливателя на GPU растет практически линейно в зависимости от количества ненулевых элементов в регулярной части матрицы. Достигается хорошее ускорение при крупноблочном распараллеливании построения ILU0 на двух GPU. Необходимо отметить, что время на проведение этапа анализа может быть сопоставимо с временем построения ILU0, но здесь не приводится и не учитывается, т.к. данная процедура является разовой.

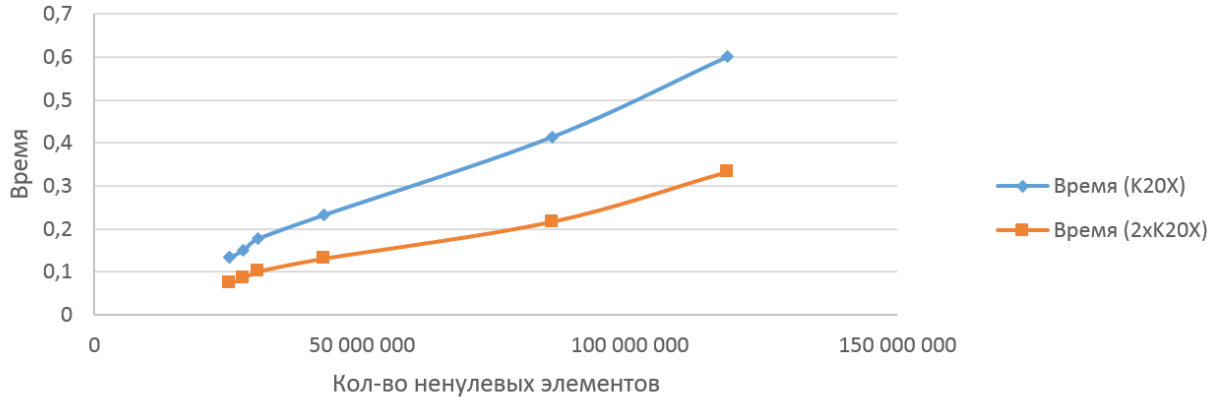

Рис. 4. Зависимость времени выполнения предобуславливания от количества ненулевых элементов в регулярной части матрицы на GPU.

Ситуация с эффективностью распараллеливания при решении треугольных систем на двух графических процессорах обстоит гораздо хуже (см. таблицу 6 и рисунок 6). Заметное ускорение удается получить лишь на двух матрицах наибольшей размерности (6 и 7), а на матрице 4 никакого ускорения получить не удается. На наш взгляд, это связано с нелинейным снижением количества уровней, возникающих в блоках предобуславливателя: снижение размерности решаемых треугольных систем в два раза не приводит к сопутствующему снижению числа уровней, соответственно обработка каждого из блоков на выделенном GPU распараллеливается менее эффективно.

Таблица 6. Решение треугольных систем на GPU.

Матрица	Время, с		Ускорение
	1 GPU	2 GPU	
1	0,028	0,027	1,037
2	0,028	0,027	1,037
3	0,033	0,031	1,065
4	0,041	0,043	0,953
5	0,051	0,047	1,085
6	0,079	0,065	1,215
7	0,142	0,126	1,127

Операция умножения разреженной матрицы на вектор, напротив, ускоряется практически идеально при использовании двух GPU, что можно объяснить увеличением пропускной способности памяти за счет подключения к расчету второго графического процессора с независимой памятью, а также достаточным ресурсом параллелизма при разделении матрицы на 2 части.

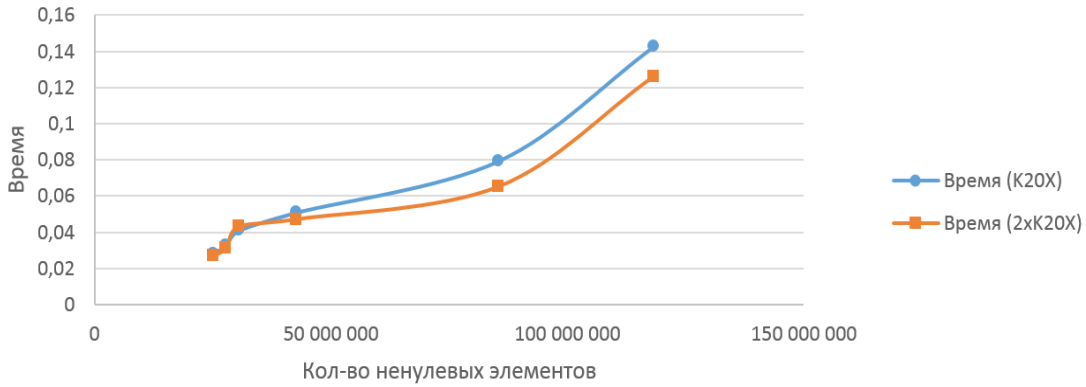

Рис. 5. Зависимость времени решения треугольных систем от количества ненулевых элементов в регулярной части матрицы на GPU.

На рисунке 6 показан график зависимости времени выполнения данной операции от размерности матриц для 1 и 2 GPU. Время обработки растет практически линейно с увеличением размерности матриц. Однако необходимо отметить относительно высокую производительность при обработке матрицы 4, что можно объяснить меньшей разреженностью данной матрицы относительно остальных, т.к. чем плотнее заполнены строки в матрице, тем выше локальность доступа к памяти, а также ресурс SIMD-параллелизма.

Таблица 7. Умножение разреженной матрицы на вектор на GPU.

Матрица	Время, с		Ускорение
	1 GPU	2 GPU	
1	0,005	0,002	1,881
2	0,005	0,003	1,864
3	0,005	0,003	1,928
4	0,007	0,004	1,954
5	0,008	0,004	1,982
6	0,016	0,008	1,946
7	0,021	0,011	1,909

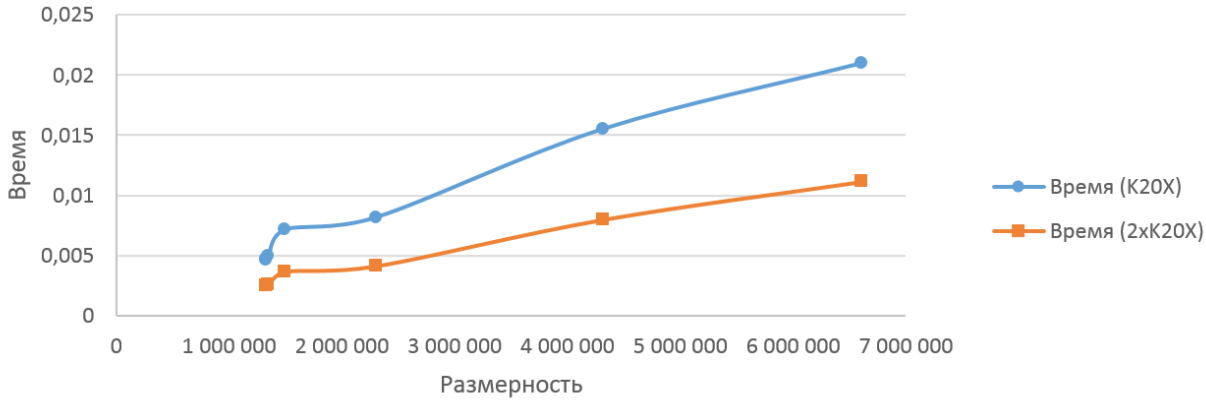

Рис. 6. Зависимость времени умножения разреженной матрицы на вектор от размерности на GPU.

Подытоживая вышесказанное, отметим, что при сравнении минимальных времен выполнения базовых операций с учетом крупноблочного распараллеливания на CPU и GPU имеем следующее:

- построение матрицы предобуславливателя проводится на GPU в 1,5-3,5 раза;
- решение треугольных систем осуществляется быстрее на CPU в 1,4-3,1 раза;
- умножение разреженной матрицы на вектор проводится быстрее на GPU в 5-16,5 раза.

Причем наибольшее ускорение на GPU относительно CPU получается для матриц большей размерности, при обработке которых невозможно эффективное использование кэш-памяти CPU.

Таблица 8. Сравнение общего времени решения разреженных СЛАУ на

CPU и GPU.

Матрица	Процессор	Кол-во ядер CPU или кол-во GPU	Кол-во итераций	Общее время решения	Ускорение относительно 1 ядра CPU
4	CPU	1	3	3,038	1,00
	CPU	8	4	0,631	4,81
	GPU	2	3	0,412	7,37
5	CPU	1	36	21,876	1,00
	CPU	8	59	7,606	2,88
	GPU	2	36	4,179	5,23
6	CPU	1	74	84,187	1,00
	CPU	12	88	19,767	4,26
	GPU	2	79	13,199	6,38
7	CPU	1	9	39,963	1,00
	CPU	12	15	10,496	3,81
	GPU	1	9	3,844	10,40

Рассмотрим таблицу 8, содержащую сведения о времени выполнения и сходимости предобусловленного метода решения СЛАУ, включающего все описанные выше базовые операции. В таблице для каждой матрицы приведены лучшие (по времени выполнения) результаты, полученные на CPU и GPU, а также результат последовательного расчета на одном ядре CPU. Приведены ускорения параллельных реализаций относительно времени последовательного расчета. Основными результатом является ускорение решения СЛАУ на многоядерной системе с общей памятью в 2,8-4,8 раза, а также возможность получить дополнительное ускорение в 1,5-2,7 раза за счет использования GPU.

Заключение

В данной работе была исследована эффективность распараллеливания на современных многоядерных процессорах решения разреженных СЛАУ итерационным методом BiCGStab с блочным предобуславливателем на основе ILU0. Показана возможность ускорения работы метода на некоторых СЛАУ, возникающих в задаче численного моделирования многофазных фильтрационных потоков в нефтегазовых пластах, до 10 раз относительно времени последовательного расчета на одном ядре CPU за счет использования ряда модификаций алгоритма построения и применения предобуславливателя ILU0. Полученное ускорение работы метода сопровождается снижением качества предобуславливания, что выливается в увеличение количества итераций, требуемых для сходимости метода. Поэтому дальнейшие работы планируется проводить как в направлении оптимизации разработанных программ, так и в исследовании возможностей улучшения сходимости.

ЛИТЕРАТУРА:

1. Saad Y. Iterative methods for sparse linear systems. – 2nd edition. – SIAM Society for Industrial & Applied Mathematics, 2003. – 477 p.
2. Богачев К.Ю., Жабицкий Я.В. Блочные предобусловливатели класса ILU для задач фильтрации многокомпонентной смеси в пористой среде // Вестник МГУ. Серия 1: Математика. Механика. 2009. №5. С. 19-24.
3. Борщук О.С. О модификации двухступенчатого метода предобуславливания при численном решении задачи многофазной фильтрации вязкой сжимаемой жидкости в пористой среде // Вестник УГАТУ. 2009. Т. 12. № 1. С. 146-150.
4. Фомина Л.Н. О повышении эффективности полинейного рекуррентного метода решения разностных эллиптических уравнений // Труды международной конференции «Современные проблемы прикладной математики и механики: теория, эксперимент и практика» (Новосибирск, Россия, 30 мая – 4 июня 2011 г.). URL: http://conf.nsc.ru/niknik-90/ru/reportview/37289/
5. Naumov M. Parallel Solution of Sparse Triangular Linear Systems in the Preconditioned Iterative Methods on the GPU. Technical Report NVR-2011-001, NVIDIA, 2011.
6. Naumov M. Incomplete-LU and Cholesky Preconditioned Iterative Methods Using CUSPARSE and CUBLAS. NVIDIA Technical Report NVR-2012-003, 2012.
7. Song Yu, Hui Liu, Zhangxin Chen, Ben Hsieh, and Lei Shao. GPU-based Parallel Reservoir Simulation for Large-scale Simulation Problems SPE Europec/EAGE Annual Conference, 4-7 June 2012, Copenhagen, Denmark.