УДК 532.546

БИБЛИОТЕКА ШАБЛОНОВ ИТЕРАЦИОННЫХ МЕТОДОВ ПОДПРОСТРАНСТВ КРЫЛОВА ДЛЯ ЧИСЛЕННОГО РЕШЕНИЯ ЗАДАЧ МЕХАНИКИ СПЛОШНЫХ СРЕД НА ГИБРИДНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Д. А. Губайдуллин[1], **А. И. Никифоров**[1], **Р. В. Садовников**[1]

Представлена библиотека шаблонов C++ итерационных методов подпространств Крылова с предобусловливанием для решения больших разреженных систем линейных алгебраических уравнений на современных графических процессорах NVIDIA. Методы могут использоваться как на отдельном, так и на нескольких графических устройствах одновременно, т.е. на так называемых гибридных вычислительных системах, в которых вычисления на центральных процессорах совмещаются с вычислениями на графических процессорах. Библиотека предназначена для пользователей, которым приходится иметь дело с большими разреженными системами линейных алгебраических уравнений. Работа выполнена в рамках программы Президиума РАН № 14 “Интеллектуальные информационные технологии, математическое моделирование, системный анализ и автоматизация”.

Ключевые слова: параллельные вычисления, графические процессоры, итерационные методы подпространств Крылова, метод контрольных объемов, фильтрация в пористых средах.

1. Введение. Методы подпространств Крылова с предобусловливанием являются наиболее эффективной группой итерационных методов, применяемой для решения разреженных систем линейных алгебраических уравнений. Такие системы уравнений возникают при численном решении широкого класса задач механики сплошных сред, описываемых дифференциальными уравнениями в частных производных: задачи теплопроводности, задачи гидродинамики многофазных сред, задачи геофизики и др. Для численного решения этих задач требуются значительные вычислительные ресурсы. Повышенные требования к производительности и памяти обусловлены пространственным характером и нестационарностью протекающих процессов, многофазностью сред, нелинейностью моделей сред и другими факторами. Применение параллельных вычислений значительно расширяет возможности исследователей, занимающихся компьютерным моделированием таких сложных физических процессов [1]. Для решения разреженных линейных систем большой размерности на различных архитектурах центральных процессоров, а также параллельных вычислительных кластерах, построенных на их основе, создано множество различных библиотек программ [2]. Как правило, библиотеки программ используют итерационные методы с предобусловливанием и отличаются форматами представления матриц, типами предобусловливателей, способами обмена сообщениями между процессорами и связаны с архитектурой параллельной вычислительной системы, типами поддерживаемых платформ и др.

В последнее время возрос интерес к параллельным вычислениям на графических процессорных устройствах (ГПУ) NVIDIA и AMD в связи с их высокой производительностью и низким энергопотреблением. При программировании с использованием ГПУ последнее рассматривается как вычислительное устройство, способное выполнять большое число одинаковых вычислений параллельно. Для программирования на ГПУ значительную популярность приобрела унифицированная архитектура компьютерных вычислений NVIDIA CUDA [3], основанная на расширении языка C. Эта технология дает возможность доступа к набору инструкций ГПУ и управления его памятью при организации параллельных вычислений.

Особенно перспективным является одновременное использование для расчетов нескольких графических устройств, установленных как на одном вычислительном узле, так и на нескольких вычислительных узлах, т.е. использование так называемых гибридных вычислительных систем, на которых вычисления на центральных процессорах (ЦПУ) совмещаются с вычислениями на графических процессорах. Такие вычислительные системы отличаются довольно высокой производительностью и невысокой стоимостью

[1] Институт механики и машиностроения Казанского научного центра РАН, ул. Лобачевского, 2/31, 420111, г. Казань; Д. А. Губайдуллин, директор, чл.-корр. РАН, e-mail: gubajdullin@mail.knc.ru; А. И. Никифоров, зав. лабораторией, e-mail: nikiforov@mail.knc.ru; Р. В. Садовников, ст. науч. сотр., e-mail: sadovnikov@mail.knc.ru

(собираются из компьютеров и графических видеокарт серийного производства) [4]. Для расчетов на таких системах требуется структурная перестройка алгоритма с явным выделением критических фрагментов, которые можно эффективно реализовать на графических процессорах.

Одной из первых работ, посвященных реализации алгоритмов решения линейных систем с разреженной матрицей на ГПУ с помощью CUDA, является работа [5], в которой предложена реализация метода сопряженных градиентов. И хотя метод нельзя назвать универсальным, так как он ограничен использованием только симметричных матриц, тем не менее, в работе была предложена концепция использования итерационных методов на ГПУ. В работах [6, 7] рассмотрены различные варианты форматов представления разреженных матриц и способы параллельной реализации операции умножения матрицы на вектор на ГПУ, а также вопросы их оптимизации. В [8] на основе работы [5] предложена реализация стабилизированного метода бисопряженных градиентов на ГПУ, который позволяет работать с несимметричными разреженными матрицами. В работе [9] была представлена библиотека gpu_sparse итерационных методов подпространств Крылова с предобусловливанием для решения на ГПУ разреженных линейных систем с нерегулярной структурой (как симметричных, так и несимметричных).

В настоящей статье представлено расширение библиотеки gpu_sparse итерационных методов для использования на гибридных вычислительных системах, на которых вычисления на центральном процессоре совмещаются с вычислениями на графических процессорах. Все методы записаны в виде шаблонов на языке программирования C++ таким образом, что поддерживаются вычисления как с одинарной, так и с двойной точностью. Библиотека итерационных методов предназначена для пользователей, которым приходится иметь дело с большими разреженными линейными системами. Представленные в библиотеке методы протестированы на примере численного решения задачи фильтрации. Проведено сравнение производительности вычислений на узле с несколькими ГПУ с расчетами на многопроцессорном кластере.

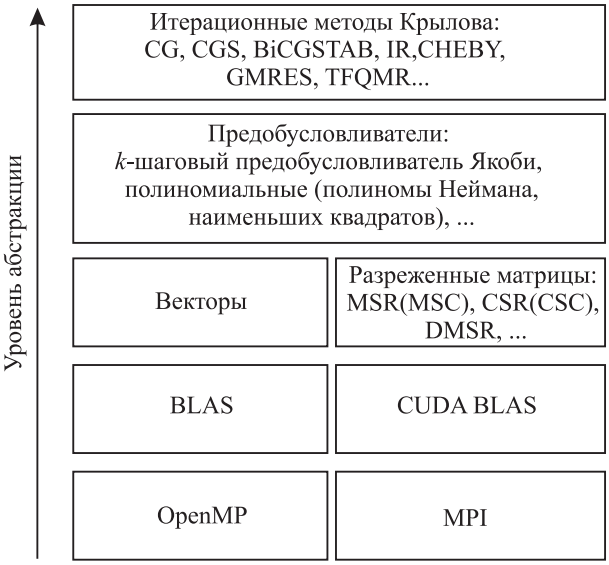

Рис. 1. Схема библиотеки gpu_sparse

2. Структура библиотеки. На рис. 1 представлена схема библиотеки gpu_sparse. На нижнем уровне абстракции представлены основные операции с векторами как на центральном, так и на одном или нескольких графических процессорах. Эти операции не зависят от структуры представления разреженной матрицы. Операции с векторами реализованы с помощью библиотеки CUDA BLAS [10]. Обмены между ядрами (процессорами) реализуются с помощью библиотек OpenMP [11] (MPI [12]). На следующем уровне абстракции представлены классы векторов, матриц и предобусловливателей, которые поддерживают операции умножения матрицы на вектор как на центральном процессоре, так и на одном или нескольких графических процессорах. Верхний уровень абстракции представлен функциями, реализующими итерационные методы подпространств Крылова. Все операции и методы реализованы таким образом, что возможна их дальнейшая модификация в связи, например, с изменением архитектуры графических процессоров или использованием другого формата матрицы.

Среди методов, которые были реализованы в составе библиотеки, следующие: IR — метод Ричардсона, Cheby — метод Чебышева, CG — метод сопряженных градиентов, CGS — квадратичный метод сопряженных градиентов, BiCGSTAB — стабилизированный метод бисопряженных градиентов, GMRES — обобщенный метод минимальных невязок с рестартами и TFQMR — метод квазиминимальных невязок без использования транспонирования. Среди предобусловливателей, которые реализованы в библиотеке, следующие: k-шаговый диагональный предобусловливатель Якоби (где $k > 0$ — показатель степени), предобусловливание полиномами Неймана и полиномами наименьших квадратов.

Основные этапы реализации методов на ГПУ следующие: загрузить вектора и матрицу (в одном из форматов) с ЦПУ на ГПУ; произвести операции умножения, сложения, вычитания с векторами, матрицами и предобусловливателями; загрузить результат из ГПУ на ЦПУ.

Детали реализации структуры данных в библиотеке отделены от математического алгоритма с помощью шаблонов и объектно-ориентированного программирования на языке C++. Векторы и матрицы записаны в виде шаблонов классов, так что они могут использоваться для расчетов как с одинарной, так

и с двойной точностью. Методы записаны в виде функций в формате шаблона, что позволяет использовать их с любой матрицей и вектором, обеспечивая необходимый уровень функциональности. Шаблоны классов библиотеки представлены в виде заголовочных файлов, а их код помещен в пространство имен gpu_sparse. Поэтому при использовании библиотеки не возникает конфликта имен переменных, классов и функций, а также не требуется сборка библиотеки под конкретную операционную систему, что способствует ее переносимости. Ниже представлен код шаблона класса на C++ итерационных методов подпространств Крылова. Шаблон класса параметризован типами переменных, а также операциями, используемыми над векторами. По умолчанию для операций линейной алгебры с векторами на ГПУ используются функции библиотеки CUDA BLAS [10], которые реализованы в шаблоне класса cuda_blas. Однако для них могут использоваться любые другие реализации операций над векторами, в том числе операции BLAS для центральных процессоров. Функции-члены класса параметризованы типом матрицы, вектора и предобусловливателя. Такая организация шаблонов классов библиотеки делает их универсальными в использовании как на центральных процессорах, так и на графических.

```
namespace gpu_sparse {
...
template<typename IndexType, typename ValueType, template<typename> class
Operations = cuda_blas>
class Krylov {
...
public:

//bicgstab
template<class Matrix, class Vector, class Preconditioner>
inline static IndexType bicgstab(Matrix &A, Vector &b, Vector &x,
Preconditioner &M, IndexType iter_max, ValueType epsilon, double &time);

//gmres
template<class Matrix1, class Vector, class Preconditioner, class Matrix2>
inline static IndexType gmres(Matrix1 &A, Vector &b, Vector &x,
Preconditioner &M, Matrix2 &H, IndexType restarts, IndexType iter_max,
ValueType epsilon, double &time);

//tfqmr
template<class Matrix, class Vector, class Preconditioner>
inline static IndexType tfqmr(Matrix &A, Vector &b, Vector &x,
Preconditioner &M, IndexType iter_max, ValueType epsilon, double &time);
...
};
...
};
```

3. Реализация операции умножения матрицы на вектор. Самой трудоемкой операцией в итерационных методах является операция умножения матрицы на вектор. Из-за непредсказуемого шаблона доступа к памяти и сложной структуры данных для представления разреженных матриц построение эффективных алгоритмов для этой операции затруднено. Способ ее вычисления на любой архитектуре вычислителя, в том числе на ГПУ, зависит от формата представления разреженной матрицы, который, в свою очередь, связан со спецификой рассматриваемой задачи, с типом используемой сетки (количество ненулевых элементов матрицы определяется связями узлов сетки) и способом аппроксимации. Известно довольно много схем хранения разреженных матриц, которые встречаются в расчетах. Применение определенной схемы заключается в увеличении эффективности как в использовании памяти, так и в уменьшении количества арифметических операций.

3.1. Реализация операции умножения матрицы на вектор на системе с одним ГПУ. Для матриц с произвольной нерегулярной структурой ненулевых элементов одним из самых распространенных форматов хранения разреженной матрицы является формат CSR (CSC) — формат сжатой разреженной строки (столбца). Другой формат MSR (MSC) — модифицированный формат разреженной строки (столбца) — отличается от CSR (CSC) тем, что главная диагональ матрицы хранится отдельно от внедиагональных членов в каждой строке (столбце). Выбор этих форматов представления разреженных матриц

для использования мотивирован несколькими факторами: их простотой, общностью представления, широким использованием и довольно легкой параллельной реализацией. Запись методов библиотеки в виде шаблонов на языке C++ позволяет использовать любые другие форматы матриц без изменения алгоритмов. Схема реализации операции умножения матрицы в MSR-формате на вектор на отдельном ГПУ была подробно описана в работе [9]. В данной работе рассматривается реализация этой операции и схема вычислений при использовании нескольких ГПУ одновременно.

3.2. Реализация операции умножения матрицы на вектор на системе с несколькими ГПУ. Для использования нескольких графических устройств на одном вычислительном узле требуется одновременно запустить несколько потоков ЦПУ. Поскольку данные одного графического устройства недоступны напрямую для других графических устройств, необходимо организовать обмен данными между устройствами через память ЦПУ. Для этого данные из памяти графического устройства сначала пересылаются в память ЦПУ. После обмена данными между потоками ЦПУ эти данные пересылаются в память другого графического устройства. Обмен данными между памятью ГПУ и ЦПУ осуществляется с помощью функций CUDA Runtime API [13] через PCI-Express x16, который поддерживает скорость передачи данных до 4 Гб/с. Обмен данными между потоками ЦПУ реализуется средствами OpenMP или MPI.

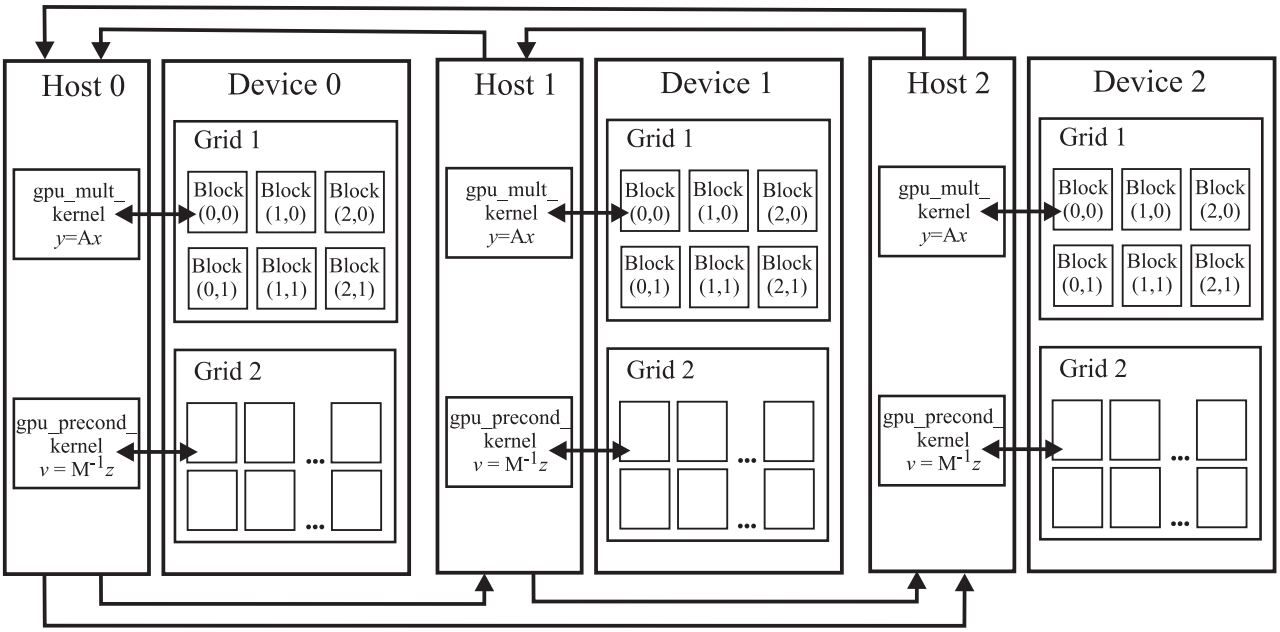

Рис. 2. Схема умножения матрицы на модуле гибридной вычислительной системы с тремя ГПУ

Следовательно, для использования нескольких графических устройств на одном вычислительном узле можно предложить следующую схему вычислений. Необходимо разделить область решения задачи на подобласти. Для этого применяются методы декомпозиции области (декомпозиция сетки расчетной области) с помощью методов теории графов, реализованных в специальных библиотеках программ таких, например, как Chaco [14] и Metis [15]. После сборки каждым потоком ЦПУ своей локальной подматрицы и соответствующего вектора правой части линейной системы, каждой подобласти будут соответствовать строки матрицы и компоненты векторов правой части и вектора неизвестных, соответствующие номерам узлов сетки подобласти. При умножении строк локальной подматрицы на вектор на каждом из ГПУ могут потребоваться части этого вектора, принадлежащие другим ГПУ (так называемые внешние для данного устройства переменные). Поэтому перед каждым умножением локальной подматрицы на вектор на ГПУ необходимо сначала переслать внешние для данной подобласти части этого вектора с соседних ГПУ. Такой обмен данными реализуется либо через общую память ЦПУ средствами OpenMP, либо с помощью межпроцессорных сообщений средствами MPI. На рис. 2 представлена схема умножения разреженной матрицы на одном узле гибридной вычислительной системы с тремя графическими устройствами. Стрелками указаны обмены данными между потоками ЦПУ и памятью ГПУ.

Для выполнения операции умножения разреженной матрицы на вектор на узле с несколькими графическими устройствами хорошо подходит формат DMSR — формат распределенной модифицированной разреженной строки. Формат DMSR — это обобщение формата MSR. Структура данных для хранения матрицы в формате DMSR состоит из двух векторов (целочисленного и вещественного), каждый длины

$nnz_i + 1$ $(i = 1, n_{\text{gpu}})$, где nnz_i — количество ненулевых элементов в локальной подматрице для данного ГПУ, n_{gpu} — количество ГПУ на данном вычислительном узле. Первые n_i позиций целочисленного вектора содержат значения главной диагонали, а последующие позиции содержат индексы столбцов элементов и указатели на начало каждой строки локальной подматрицы. Вещественный вектор содержит ненулевые значения локальной подматрицы, которые хранятся строка за строкой. Для более детального обсуждения формата DMSR и его использования необходимо обратиться к [16].

Операции с векторами, такие как сложение (вычитание) векторов и умножение вектора на скаляр, в случае использования нескольких ГПУ остаются без изменений за исключением вычисления нормы вектора и скалярного произведения векторов. После вычисления значений этих операций локально на каждом ГПУ требуется глобальная операция суммирования, которая выполняется средствами OpenMP или MPI.

Такая же схема вычислений может быть использована и в случае объединения нескольких вычислительных узлов, каждый из которых снабжен, как минимум, одним графическим устройством. В этом случае для межпроцессорных коммуникаций используется MPI. При объединении нескольких вычислительных узлов с несколькими графическими устройствами обмен данными между графическими устройствами локально в рамках одного вычислительного узла может выполняться с помощью функций библиотеки OpenMP, а обмен данными между отдельными узлами — с помощью библиотеки MPI.

Ниже представлен пример использования библиотеки шаблонов итерационных методов на модуле гибридной вычислительной системы с помощью OpenMP.

```
#include "gpu_vector.h"
#include "gpu_dmsr.h"
#include "gpu_msr_poly_pre.h"
#include "gpu_solver.h"
using namespace gpu_sparse;

int main(void) {
#pragma omp parallel num_threads(3) {
float tol= 0.000001f;
int maxit= 2500;
double solve_time;
GPU_DMSR_Mat<float> A(M,N,nz);
GPU_Vector<float> xi(N), rhs(N);
std::fstream rhs_file("rhs",std::ios::in);
rhs_file>>rhs;
rhs_file.close();
std::fstream matrix_file("matrix",std::ios::in);
matrix_file>>A;
matrix_file.close();
GPU_Poly_Precond<float> M(A);
int results = Krylov<int,float>::bicgstab(A, rhs, xi, M, maxit, tol,
&solve_time);
std::cout<<"solve_time="<<solve_time<<std::endl;
...
}
return 0;
};
```

4. Численные результаты. В этом разделе представлены результаты тестирования описанной выше библиотеки gpu_sparse итерационных методов подпространств Крылова с предобусловливанием на примере численного решения задачи фильтрации жидкости к скважинам в трехмерной области произвольной формы. Для тестирования библиотеки gpu_sparse на нескольких графических устройствах использовался вычислительный модуль, оснащенный центральным процессором Intel Core i7-920 (4 ядра, 2.66 ГГц, 6 Гб ОЗУ), а также двумя видеокартами NVIDIA: Palit GTS 250 (128 ядер с частотой 745 МГц, 16 потоковых мультипроцессоров, 1 Гб DRAM DDR3 с пропускной способностью 70.4 Гб/с), MSI GTS 250 (128 ядер с частотой 760 МГц, 16 потоковых мультипроцессоров, 1 Гб DRAM DDR3 с пропускной способностью 70.4 Гб/с) и высокопроизводительным вычислительным модулем Tesla C1060 (240 ядер с частотой 1296 МГц, 4 Гб DRAM DDR3 с пропускной способностью 102 Гб/с). Результаты расчетов срав-

ниваются с решением этой же задачи с помощью библиотеки подпрограмм Aztec [16] на вычислительном кластере с параллельной архитектурой. Кластер состоит из 4 модулей. Каждый модуль оснащен двумя процессорами AMD Opteron 246 (2.0 ГГц, 2 Гб ОЗУ). Выполнение параллельных задач обеспечивает система управления прохождением задач МВС–1000/7. Подробное описание расчетов на кластере и основные данные приведены в работе [17].

4.1. Постановка задачи и метод решения. Рассматривается нестационарная задача фильтрации однофазной жидкости к скважинам со сложной геометрией (вертикальным, наклонным, горизонтальным, искривленным и др.) в анизотропной среде с двойной пористостью в 3D-области произвольной формы.

Модель фильтрации однофазной жидкости в упругом трещиновато-пористом пласте, основанная на концепции взаимопроникающих континуумов (система трещин и блоков) с учетом обмена жидкостью между ними, была предложена Г. И. Баренблаттом и др. [18]. При учете анизотропии системы трещин и блоков матрицы породы модель описывается системой уравнений:

$$\begin{aligned} \beta_1^* \frac{\partial p_1}{\partial t} &= \nabla\left(\frac{\boldsymbol{k}_1}{\mu}\nabla p_1\right) + \frac{\alpha}{\mu}(p_2 - p_1), \\ \beta_2^* \frac{\partial p_2}{\partial t} &= \nabla\left(\frac{\boldsymbol{k}_2}{\mu}\nabla p_2\right) - \frac{\alpha}{\mu}(p_2 - p_1), \end{aligned} \tag{1}$$

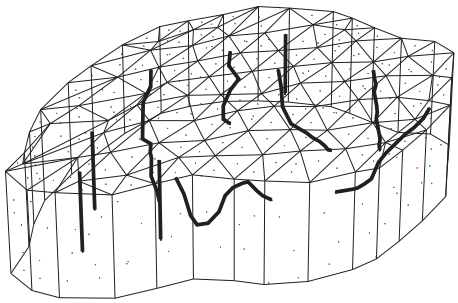

Рис. 3. Схема пласта и расположение скважин

где $(x, y, z) \in D$, $0 < t \leqslant T$, t — время, T — общее время исследований, $\beta_l^* = \beta_{\text{ск}l} + m_l\beta_{\text{ж}}$ — коэффициент упругоемкости пласта, $\beta_{\text{ск}l}$ — коэффициент сжимаемости скелета, m_l — коэффициент пористости, $\beta_{\text{ж}}$ — коэффициент сжимаемости жидкости, p_l — давление жидкости, μ — коэффициент динамической вязкости жидкости, α — параметр перетока жидкости между трещинами и блоками, $\boldsymbol{k}_l$ — тензор коэффициентов проницаемости. Индекс 1 относится к трещинам, 2 — к блокам матрицы породы. На рис. 3 представлена многосвязная область фильтрации D, внутренние поверхности которой образованы скважинами, представляющими собой цилиндрическую полость определенного радиуса и траектории. Рассматривались скважины с различной формой траектории ствола: вертикальные, наклонные, горизонтальные и др. На скважинах могут быть заданы граничные условия первого или второго рода:

$$p_1(x, y, z, t) = p_2(x, y, z, t) = p_{\text{wi}}(t), \quad 0 \leqslant t \leqslant T, \quad (x, y, z) \in \partial S_i^p, \quad i = 1, 2, \ldots, N_{\text{w}}^p, \tag{2}$$

$$\left(\frac{\boldsymbol{k}_1}{\mu}\nabla p_1, \boldsymbol{n}_j\right) + \left(\frac{\boldsymbol{k}_2}{\mu}\nabla p_2, \boldsymbol{n}_j\right) = q_j(x, y, z, t), \quad 0 \leqslant t \leqslant T, \quad (x, y, z) \in \partial S_j^q, \quad j = 1, 2, \ldots, N_{\text{w}}^q, \tag{2a}$$

где $p_{\text{wi}}(t)$ — давление на поверхности скважины ∂S_i^p, N_{w}^p — количество таких скважин, q_j — объемный расход жидкости, приходящийся на единицу поверхности скважины ∂S_j^q, $\boldsymbol{n}_j$ — вектор внешней нормали, N_{w}^q — количество таких скважин. Граничные условия на внешней поверхности пласта также могут быть заданы условиями первого или второго рода:

$$p_1(x, y, z, t) = p_2(x, y, z, t) = p_{\text{пл}}(x, y, z, t), \quad 0 \leqslant t \leqslant T, \quad (x, y, z) \in \partial D^p, \tag{3}$$

$$\left(\frac{\boldsymbol{k}_1}{\mu}\nabla p_1, \boldsymbol{n}_j\right) + \left(\frac{\boldsymbol{k}_2}{\mu}\nabla p_2, \boldsymbol{n}_j\right) = q^*(x, y, z, t), \quad 0 \leqslant t \leqslant T, \quad (x, y, z) \in \partial D^q, \tag{3a}$$

где $p_{\text{пл}}$ — давление жидкости на части внешней поверхности пласта ∂D^p, q^* — объемный расход жидкости, приходящийся на единицу внешней поверхности пласта ∂D^q, $\partial D = \partial D^p \bigcup \partial D^q$ — внешняя поверхность пласта. Начальные условия имеют вид

$$p_1(x, y, z, 0) = p_1(x, y, z), \quad p_2(x, y, z, 0) = p_2(x, y, z), \quad (x, y, z) \in D. \tag{4}$$

Объемный дебит скважины вычисляется по формуле

$$\int\limits_{\partial S_j^q} q_j(x, y, z, t)\, ds = Q_j(t), \quad j = 1, 2, \ldots, N_{\text{w}}^q. \tag{5}$$

Для получения значения забойного давления на скважине используются дополнительные условия: равенство давления в трещинах и блоках породы на поверхности скважины и постоянство давления на поверхности скважины:

$$p_1(x, y, z, t) = p_2(x, y, z, t), \quad (x, y, z) \in \partial S_j^q, \quad j = 1, 2, \ldots, N_{\text{w}}^q. \tag{6}$$

Эти два условия в сочетании с формулой (5) позволяют записать для скважины, на которой задан объемный расход, дополнительное уравнение для определения забойного давления.

Для решения задачи (1)–(6) использовался метод конечных элементов на неструктурированной сетке тетраэдров. Аппроксимация уравнений строилась методом взвешенных невязок в сочетании с методом Галеркина, а для аппроксимации производной по времени использовалась неявная схема [17].

Загрузка процессоров

Кол-во процессоров	Кол-во элементов (тыс.)	Кол-во неизвестных (тыс.)	Кол-во ненулевых элементов матрицы (тыс.)
1	4263,344	1523,460	46340,996
2	2077,584 2185,760	743,474 779,986	22573,737 23767,259
3	1400,508 1441,480 1421,356	502,120 512,738 508,602	15249,685 15591,495 15499,816

4.2. Результаты расчетов. Расчеты проводились на сетке с 761,730 тыс. узлами. Количество тетраэдров сетки составило 4263,344 тыс. элементов. Количество ненулевых элементов матрицы системы 46340,996 тыс. Поскольку каждый узел имеет две степени свободы (в каждом узле два давления жидкости), то количество неизвестных составляет 1523,460 тыс. Для разделения конечно-элементной сетки на непересекающиеся подобласти использовалась библиотека подпрограмм Metis [15], предназначенная для разделения графов. В таблице представлены данные загрузки процессоров.

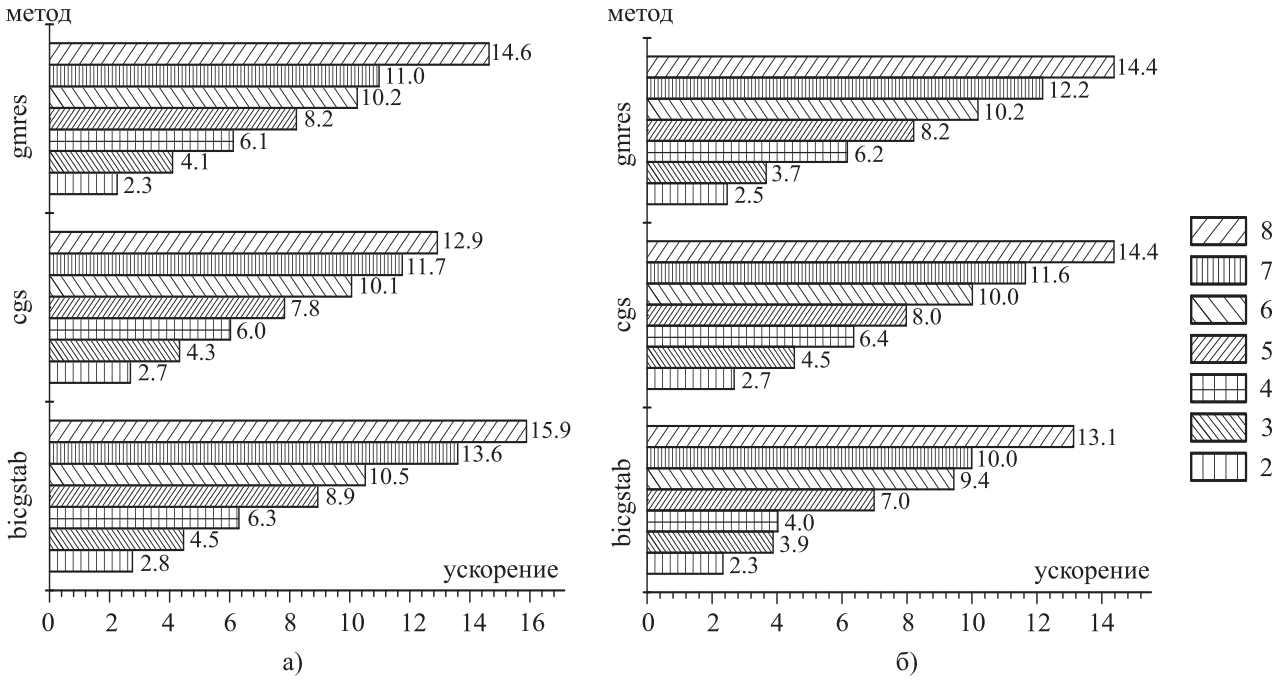

Рис. 4. Ускорение вычислений на кластере для полиномиального предобусловливания: а) рядами Неймана, б) методом наименьших квадратов

4.2.1. Решение задачи на кластере. На рис. 4 представлены результаты расчетов с помощью библиотеки Aztec на вычислительном кластере. Из представленных результатов видно, что применение параллельных вычислений на 8 процессорах позволяет ускорить вычисления почти в 13–16 раз (в зависимости от выбранного итерационного метода и типа предобусловливателя) по сравнению с последовательными вычислениями на одном процессоре. Ускорение вычислялось делением времени решения задачи на одном процессоре на время решения на n_p процессорах.

4.2.2. Решение задачи на модуле гибридной вычислительной системы. На рис. 5 и 6 представлены результаты расчетов задачи на вычислительном модуле с тремя графическими устройствами. На рис. 5 представлены результаты расчетов, в которых для обменов данными между графическими устройствами использовалась технология MPI. На рис. 6 — технология OpenMP. Ускорение вычислений подсчитывалось делением времени расчетов на восьми процессорах указанного выше кластера на время решения задачи на одном графическом устройстве GTS 250 (1 — на рис. 5 и 6), на двух графических устройствах GTS 250 (Palit и MSI; 2 — на рис. 5 и 6) и на трех графических устройствах 2xGTS 250

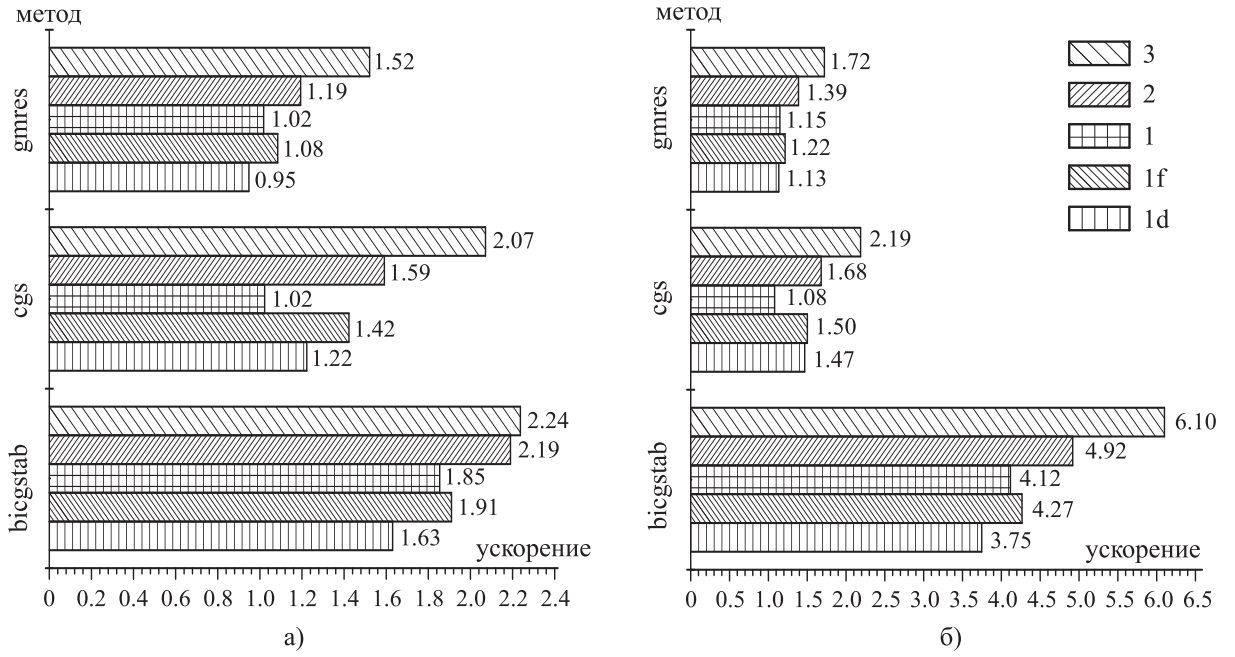

Рис. 5. Ускорение вычислений на модуле гибридной вычислительной системы с тремя ГПУ для полиномиального предобусловливания: а) рядами Неймана, б) методом наименьших квадратов. Использование технологии MPI

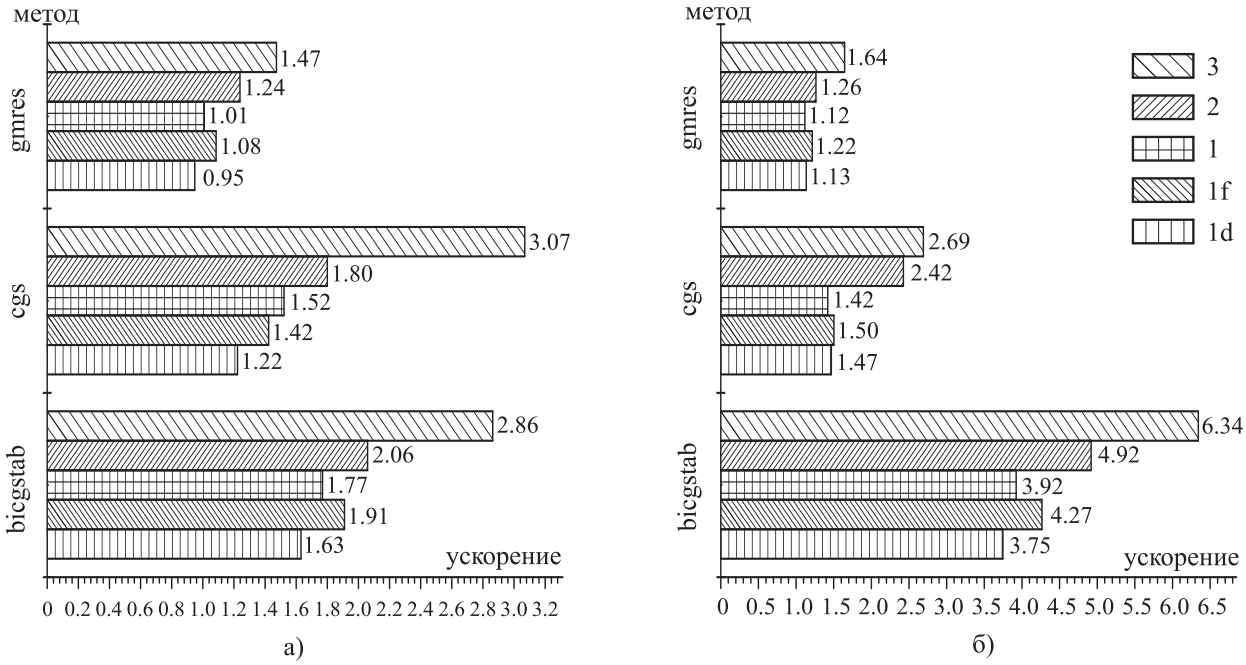

Рис. 6. Ускорение вычислений на модуле гибридной вычислительной системы с тремя ГПУ для полиномиального предобусловливания: а) рядами Неймана, б) методом наименьших квадратов. Использование технологии OpenMP

(Palit и MSI) и Tesla C1060 (3 — на рис. 5 и 6). Для сравнения приведены результаты для вычислений на Tesla C1060: с одинарной точностью (1f — на рис. 5 и 6) и с двойной точностью (1d — на рис. 5 и 6). Из представленных результатов видно, что в зависимости от метода и типа предобусловливателя максимальное ускорение для расчетов на трех графических устройствах достигает 6.3 раза по сравнению с расчетами на 8 процессорах Opteron 246. Если сравнивать расчеты, полученные с помощью MPI и OpenMP, то можно сказать, что они практически одинаковы, поскольку обмен данными происходит на одном узле и в случае MPI нет задержек, связанных с межпроцессорными обменами по сети. Таким обра-

зом, мы продемонстрировали некоторые эффективные применения библиотеки шаблонов итерационных методов подпространств Крылова с предобусловливанием на модуле гибридной вычислительной системы с графическими процессорами NVIDIA.

5. Заключение. Построена библиотека gpu_sparse итерационных методов подпространств Крылова с предобусловливанием для решения разреженных линейных систем с нерегулярной структурой (как симметричных, так и несимметричных) на гибридных вычислительных системах, на которых вычисления на центральном процессоре совмещаются с вычислениями на одном или нескольких графических процессорах NVIDIA с помощью CUDA, MPI и OpenMP. Библиотека записана в виде шаблонов классов на C++, что позволяет проводить вычисления с одинарной и с двойной точностью. Как видно из представленных расчетов, использование графических процессоров, а также построенных на их основе решений для высокопроизводительных вычислений, позволяет значительно повысить производительность расчетов при низкой себестоимости и низком энергопотреблении.

СПИСОК ЛИТЕРАТУРЫ

1. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. СПб: БХВ-Петербург, 2002.
2. http://www.netlib.org/utk/people/JackDongarra/la-sw.html.
3. NVIDIA Corporation. NVIDIA CUDA Programming Guide. February, 2010. Version 3.0.
4. *Андреев С.С., Давыдов А.А., Дбар С.А., Карагичев А.Б., Лацис А.О., Плоткина Е.А.* Макет гибридного суперкомпьютера МВС-экспресс // XVII Всероссийская конференция "Теоретические основы и конструирование численных алгоритмов для решения задач математической физики с приложением к многопроцессорным системам", посвященная памяти К. И. Бабенко. Абрау-Дюрсо, 2008.
5. *Buatois L., Cauman G., Levy B.* Concurrent number cruncher: an efficient sparse linear solver on the GPU // High Performance Computation Conference (HPCC). Springer Lecture Notes in Computer Sciences. Berlin: Springer, 2008. 358–371.
6. *Bell N., Garland M.* Efficient sparse matrix–vector multiplication on CUDA. NVIDIA Tech. Rep. 2008.
7. *Baskaran M., Bordawekar R.* Optimizing sparse matrix–vector multiplication on GPUs. IBM Tech. Rep. 2009.
8. *Чадов С.Н.* Реализация алгоритма решения несимметричных систем линейных уравнений на графических процессорах // Вычислительные методы и программирование. 2009. **10**, № 2. 135–140.
9. *Губайдуллин Д.А., Садовников Р.В., Никифоров А.И.* Использование графических процессоров для решения разреженных СЛАУ итерационными методами подпространств Крылова с предобусловливанием на примере задач теории фильтрации // Сб. трудов Международной научной конференции "Параллельные вычислительные технологии (ПаВТ-2010)". Уфа, 2010. 132–140.
10. NVIDIA Corporation. CUBLAS Library. February, 2010. Version 3.0.
11. OpenMP Architecture Review Board (http://www.openmp.org).
12. MPI: A Message-Passing Interface (MPI) Standard (http://www.mcs.anl.gov/research/projects/mpi/).
13. NVIDIA CUDA. Reference Manual. February, 2010. Version 3.0.
14. *Hendrickson B., Leland R.* The Chaco user's guide. Version 1.0. Technical Report Sand93-2339. Sandia National Laboratories. Albuquerque, 1993.
15. METIS: family of multilevel partitioning algorithms (http://glaros.dtc.umn.edu/gkhome/views/metis).
16. Aztec: a massively parallel iterative solver library for solving sparse linear systems (http://www.cs.sandia.gov/CRF/aztec1.html).
17. *Губайдуллин Д.А., Садовников Р.В.* Применение параллельных алгоритмов для решения задачи фильтрации жидкости в трещиновато-пористом пласте к скважинам со сложной траекторией // Вычислительные методы и программирование. 2007. **8**, № 2. 95–102.
18. *Баренблатт Г.И., Желтов Ю.П., Кочина И.М.* Об основных представлениях теории фильтрации однородных жидкостей в трещиноватых породах // Прикл. матем. и механ. 1960. **123**, № 3. 852–864.

Поступила в редакцию
11.10.2010