УДК 532.546

ОБ ОСОБЕННОСТЯХ ИСПОЛЬЗОВАНИЯ АРХИТЕКТУРЫ ГЕТЕРОГЕННОГО КЛАСТЕРА ДЛЯ РЕШЕНИЯ ЗАДАЧ МЕХАНИКИ СПЛОШНЫХ СРЕД

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

Обсуждаются вопросы применения высокопроизводительных вычислительных систем, построенных на базе графических процессоров NVIDIA. Сравнивается несколько моделей параллелизма для применения в итерационных методах подпространств Крылова, предназначенных для решения больших разреженных систем линейных алгебраических уравнений, возникающих при численном решении широкого класса задач механики сплошных сред, описываемых дифференциальными уравнениями в частных производных. Вычисления построены на использовании таких технологий как MPI, OpenMP, Boost и CUDA. Предложенные методы протестированы на суперкомпьютере "ГрафИТ!/GraphIT!" НИВЦ МГУ им. М. В. Ломоносова на решении нестационарной задачи фильтрации жидкости к скважинам со сложной траекторией в трехмерной области.

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

1. Введение. В настоящее время для моделирования сложных процессов механики сплошных сред стало доступно массовое применение параллельных вычислительных технологий: кластерные многопроцессорные системы с общей и распределенной памятью, многоядерные процессоры, гетерогенные суперкомпьютеры, построенные на основе современных графических процессоров и др. Для широкого использования суперкомпьютеров и эффективного использования особенностей их архитектуры при решении реальных задач необходимо постоянно адаптироваться к новым параллельным компьютерным технологиям, что требует развития новых подходов в методах решения задач, создания новых алгоритмов и программ.

Настоящая работа является продолжением работ [1–3], в которых предложена библиотека gpu_sparse шаблонов на языке C++ итерационных методов подпространств Крылова с предобусловливанием для решения разреженных симметричных и несимметричных систем линейных алгебраических уравнений с нерегулярной структурой. В этих работах подробно описан способ реализации методов библиотеки для использования как на одном, так и на нескольких графических устройствах NVIDIA одновременно, т.е. на так называемых гибридных вычислительных системах, на которых вычисления на центральном процессоре совмещаются с вычислениями на графических процессорах.

В настоящей статье представлено расширение методов библиотеки для использования на гетерогенных архитектурах суперкомпьютеров нового поколения, которые используют наряду с многоядерными центральными процессорами графические ускорители. Такие кластеры получают широкое распространение. Для использования архитектуры гетерогенных кластеров необходима структурная перестройка алгоритмов с явным выделением критических фрагментов, которые можно эффективно реализовать на графических ускорителях.

Архитектура гетерогенного вычислительного кластера объединяет в себе несколько вычислительных узлов, построенных на базе центральных процессоров с несколькими ядрами (ЦПУ) и нескольких графических процессоров (ГПУ). Для использования ресурсов такой вычислительной системы необходимо распределить вычислительную нагрузку на центральные и графические процессоры. Такое распараллеливание вычислений предполагает использование сразу нескольких технологий программирования: технологии MPI для организации обменов данными между узлами кластера, технологии OpenMP (или другой) для организации нескольких потоков процессора локально в рамках одного узла вычислительной системы, а также технологии CUDA для организации вычислений на графических ускорителях. По требованиям CUDA для использования графического устройства каждому потоку ЦПУ необходимо явно назначить контекст устройства, а это означает, что каждый поток может иметь доступ только к одному устройству

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

в любой момент времени. К тому же так как данные одного графического устройства недоступны для другого графического устройства, необходимо организовать обмен данными между ними через общую память ЦПУ.

Такая модель взаимодействия потоков ЦПУ с графическими устройствами использовалась до выхода новой версии CUDA 4.0. В новой версии каждый из потоков имеет доступ к любому из графических устройств, а память всех графических устройств объединена в единое виртуальное пространство, так что каждое графическое устройство может напрямую обратиться в память другого устройства, минуя память ЦПУ [4].

Результаты данной работы получены при использовании версии CUDA 3.2, в которой вся работа по организации обменов данными между графическими устройствами целиком ложится на программиста. Однако ничто не ограничивает применения описываемых здесь подходов в более поздних версиях CUDA.

Для эффективного использования гетерогенного кластера необходимо выделить наиболее критические фрагменты алгоритма, которые можно эффективно реализовать на графических ускорителях, поэтому на архитектуре такого кластера возможна реализация сразу нескольких шаблонов параллелизма. Мы рассмотрим здесь особенности использования архитектуры гетерогенного кластера, позволяющие применить новые способы организации данных в сложной иерархии памяти за счет использования графических ускорителей NVIDIA, а также различные способы организации взаимодействия между ними. Для достижения масштабируемости результатов на многопроцессорной платформе с графическими ускорителями предложена также новая стратегия декомпозиции области.

2. Организация библиотеки итерационных методов. Библиотека записана в виде шаблонов классов на языке C++, что позволяет проводить вычисления с одинарной и двойной точностью. Среди методов, которые 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, TFQMR — метод квазиминимальных невязок без использования транспонирования. На рис. 1 представлена схема библиотеки.

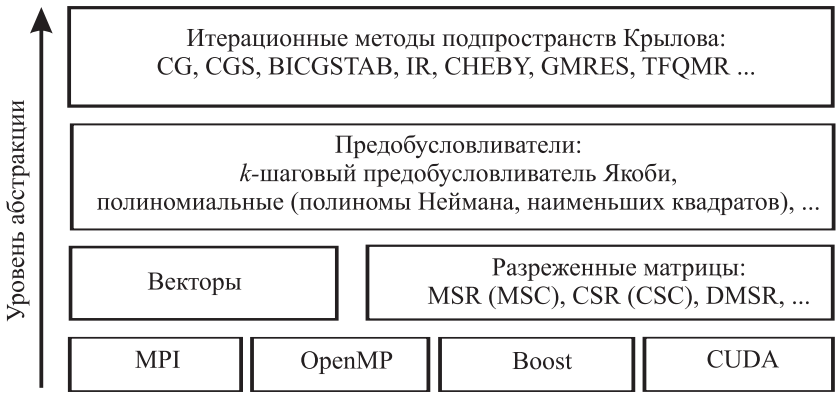

Рис. 1. Схема библиотеки

Детали реализации структуры данных в библиотеке отделены от математического алгоритма с помощью шаблонов и объектно-ориентированного программирования на языке C++. 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. Организация библиотеки нацелена на то, чтобы скрыть от пользователя коммуникации как между вычислительными узлами, так и между процессорными ядрами и графическими ускорителями. Шаблоны классов матриц и векторов в библиотеке имеют одно и то же название для каждой из моделей параллелизма, а выбор осуществляется с помощью пространства имен C++. Это позволяет использовать одни и те же имена независимо от того, какую модель выбирает пользователь для вычислений. Такая организация библиотеки позволяет пользователю сосредоточиться на распределении исходных данных задачи, чтобы эффективно использовать ресурсы вычислительной системы для решения конкретной задачи, а также на получении и анализе конечного результата.

3. Реализация умножения матрицы на вектор на гетерогенном кластере. Одной из самых основных операций в итерационных методах решения является операция умножения матрицы на вектор. Для параллельного умножения матрицы на вектор необходимо сначала распределить матрицу и вектор по ЦПУ узлов кластера. При вычислении каких-либо элементов вектора ЦПУ использует только те элементы вектора, которые были ему присвоены. Эти элементы вектора явно хранятся в памяти ЦПУ и определяются множеством индексов узлов сетки, назначенных для данного ЦПУ. Такое распределение сетки проводится на основе специальных методов теории графов [5, 6].

Аналогично векторам подмножество ненулевых элементов матрицы хранит на каждом ЦПУ только те строки, которые соответствуют переменным, назначенным для этого процессора. При этом элементы вектора, модифицируемые на данном ЦПУ, делятся на три группы: *внутренние* элементы, модифицируемые только на этом ЦПУ без использования коммуникаций между процессорами; *граничные* элементы, для модификации которых используются значения, хранящиеся на других процессорах (используются коммуникации); и *внешние* элементы, которые не модифицируются этим ЦПУ, но используются для модификации граничных элементов. Объединение множеств внутренних и граничных элементов вектора составляет множество элементов вектора, модифицируемых на этом ЦПУ. Такое представление данных способствует обмену только внешними элементами вектора между ЦПУ, что минимизирует коммуникации между узлами кластера и снижает нагрузку на коммуникационную сеть.

Элементы вектора на каждом ЦПУ хранятся в следующем порядке: сначала хранятся внутренние элементы, за которыми следуют граничные и в конце внешние элементы. Такая схема представления данных была предложена в библиотеке итерационных методов Aztec [7], которая предназначена для использования на кластерах с распределенной памятью и использует технологию MPI для обмена сообщениями между процессорами. Библиотека Aztec предоставляет набор инструментов для преобразования данных, который позволяет создавать распределенные неструктурированные разреженные матрицы для параллельного решения, подобно тому как это делается для разреженных матриц при последовательном решении. В результате сборки матрицы системы уравнений локально на каждом ЦПУ получается матрица, распределенная по процессорам кластера. Для хранения такой матрицы используется формат DMSR, который позволяет хранить ненулевые элементы каждой из подматриц в памяти ЦПУ строка за строкой.

Для реализации операции умножения матрицы на вектор на гетерогенном кластере необходимо распределить матрицу и вектор не только по центральным, но и по графическим процессорам. В настоящей статье предлагаются различные схемы, основанные на нескольких шаблонах параллелизма, учитывающих особенности гетерогенной архитектуры кластера, что позволяет применить несколько способов организации данных в сложной структуре памяти гетерогенного кластера. Процедура умножения матрицы на вектор происходит в несколько этапов. Для организации вычислений на ГПУ-кластере используются множественные интерфейсы программирования, такие как MPI, OpenMP, Boost, CUDA, а также новая стратегия декомпозиции области для параллелизма данных задачи, необходимого для достижения высокой пропускной способности и масштабируемых результатов на многопроцессорной графической платформе.

3.1. Двухуровневая декомпозиция области. Использование сразу нескольких ГПУ на каждом из узлов гетерогенного кластера позволяет произвести двухуровневую декомпозицию расчетной области. На первом уровне область разбивается на части по вычислительным узлам, аналогично тому, как это обычно делается при использовании многопроцессорного кластера с распределенной памятью. На втором уровне подобласть каждого вычислительного узла распределяется по доступным для данного узла графическим ускорителям. Для этого нами предлагается следующая процедура, которая использует преимущества описанной выше схемы организации данных в пакете Aztec, представляющей интерес с точки зрения ее использования на гетерогенном кластере.

Прежде всего мы используем тот факт, что для модификации значений из множества внутренних элементов вектора не требуются коммуникации между узлами вычислительного кластера. Следовательно, это множество можно эффективно распределить между графическими устройствами узла кластера. Для того чтобы распределить вычислительную нагрузку между несколькими графическими устройствами, необходимо распределить между ними данные (матрицу и вектор), соответствующие внутренним элементам. Для этого нами предлагается довольно простая процедура, которая заключается в том, что примерно одинаковое количество ненулевых элементов строк матрицы для каждого вычислительного узла распределяется между всеми ГПУ, начиная с первой и до последней строки по порядку. Выбор такого разбиения позволяет нам совершать одинаковое количество операций умножения и сложения при умножении строк матрицы на вектор на каждом ГПУ.

Далее, чтобы исключить обмен данными между графическими устройствами локально в рамках одного узла кластера, необходимо предоставить каждому ГПУ все элементы вектора, соответствующие номерам элементов, модифицируемым на данном ЦПУ. Это, конечно, немного перегружает память ГПУ, но избавляет нас от организации обменов данными между ГПУ при умножении матрицы на вектор одновременно на нескольких графических устройствах в рамках одного узла кластера. Поскольку матрица в ходе итерационного процесса не модифицируется, то ее нужно разместить в памяти ЦПУ и ГПУ только один раз перед основным стартом работы итерационного процесса того или иного метода. Это исключает дополнительные расходы, связанные с копированием матрицы на графические устройства при каждом умножении матрицы на вектор. Следует отметить очень важное преимущество такой декомпози-

ции области — она позволяет получить дополнительное измельчение области без увеличения нагрузки на коммуникационную сеть.

Оставшаяся часть матрицы, соответствующая множеству граничных элементов, которая предполагает использование данных с соседних процессоров, умножается на ЦПУ. Следует заметить, что количество таких граничных элементов значительно меньше количества внутренних элементов вектора, что позволяет эффективно использовать ресурсы ГПУ и ЦПУ. Процедура умножения матрицы на вектор происходит в несколько этапов и зависит от используемого шаблона параллелизма. Рассмотрим каждый из них отдельно.

3.2. MPI+CUDA. На каждом процессоре узла запускается один MPI-процесс, который использует только одно процессорное ядро и обращается только к одному ГПУ. В этой модели все векторы и матрицы на каждом ЦПУ перед стартом итерационного процесса копируются в память графического устройства и итерационный процесс организуется таким образом, что основной MPI-процесс управляет только вызовом процедур CUDA, замаскированных с помощью шаблонов языка C++ под шаблонные функции и методы шаблонов классов, а все вычисления с матрицей и векторами производятся на ГПУ.

Процедура умножения матрицы на вектор включает в себя следующие этапы. На первом этапе происходит копирование вектора с графического устройства в память ЦПУ. На втором этапе происходит обмен внешними элементами вектора между узлами кластера посредством библиотеки MPI. На третьем этапе обновленный вектор копируется обратно в память графического устройства. На четвертом этапе локальная матрица умножается на вектор на ГПУ. Таким образом, в этой модели ЦПУ узлов отвечают за обновление данных и коммуникации между ними посредством MPI, а основные вычисления реализуются на графических ускорителях с помощью CUDA. Такая модель использует ту же самую декомпозицию расчетной области, которая используется на многопроцессорном кластере с распределенной памятью. Такую декомпозицию области можно совершить с помощью специальных библиотек программ, построенных на методах теории графов [5, 6], в результате чего достигается равномерная загрузка вычислениями используемых графических процессоров и минимизируется количество сообщений между процессорами [8].

3.3. MPI+OpenMP+CUDA. Для использования на одном вычислительном узле одновременно нескольких графических ускорителей необходимо запустить несколько потоков ЦПУ, каждый из которых имеет доступ только к одному графическому устройству (только для CUDA версии не выше 3.2). Для этого на каждом узле запускается один MPI-процесс, который затем разветвляется на несколько процессорных потоков с помощью технологии OpenMP [9]. Технология OpenMP реализует параллельные вычисления с помощью многопоточности, в которой “главный” (master) поток создает набор “подчиненных” (slave) потоков и между ними распределяются вычисления. OpenMP предоставляет интерфейс взаимодействия с потоками посредством директив препроцессора.

Для эффективного использования потоков ЦПУ, участвующих в программе, необходимо, чтобы все потоки были равномерно загружены работой. Поскольку в рассматриваемой схеме умножение подматрицы каждого ЦПУ может производиться сразу на нескольких ГПУ, то эта схема предполагает, чтобы подматрица каждого ЦПУ была распределена между ГПУ равномерно. Мы предлагаем разделение матрицы между графическими устройствами, которое подробно было описано выше (см. п. 3.1). Это разделение матрицы будет соответствовать декомпозиции второго уровня расчетной области, поскольку каждому ГПУ будет назначено соответствующее множество узлов расчетной сетки подобласти ЦПУ. Заметим, что поскольку матрица системы не изменяется в процессе итерационного метода, то ее разделение и загрузку в память графических устройств необходимо произвести только один раз при старте итерационного метода. Поэтому в этой модели процедура умножения матрицы на вектор будет включать следующие этапы.

На первом этапе производится обмен внешними элементами вектора между ЦПУ узлов кластера посредством библиотеки MPI. На втором этапе каждый поток ЦПУ копирует этот вектор в память назначенного ему графического устройства. На третьем этапе каждым потоком ЦПУ вызывается функция умножения матрицы на вектор на ГПУ. На четвертом этапе каждым потоком ЦПУ копируется результат из памяти ГПУ в соответствующие элементы вектора на ЦПУ. На последнем этапе оставшаяся часть матрицы (соответствующая множеству граничных элементов) умножается на вектор на ЦПУ.

Следует заметить, что в этом подходе назначение контекста устройства конкретному потоку ЦПУ происходит один раз, во время распределения матрицы по графическим устройствам. Для этого с помощью директив препроцессора OpenMP порождается параллельная область с заданным количеством потоков ЦПУ, равным необходимому количеству ГПУ, и для каждого потока назначается контекст устройства с помощью функции cudaSetDevice(ndev), где ndev — номер устройства. После этого на каждом графическом устройстве размещается соответствующая часть матрицы необходимого размера и параллельная

область закрывается.

Далее, при умножении матрицы на вектор на ГПУ открывается параллельная область, при этом каждым потоком используется контекст того устройства, которое было назначено ему в предыдущей параллельной области. Такое использование возможно благодаря тому, что OpenMP использует целое множество потоков, которое не уничтожается между параллельными и последовательными областями, а управление передается либо заданному количеству потоков для параллельной области, либо главному потоку для последовательной области. Постоянная поддержка множества потоков дает выигрыш в производительности, так как создание потоков в каждой параллельной области является очень медленной операцией [9].

3.4. MPI+Boost+CUDA. Эта модель аналогична предыдущей, только вместо OpenMP потоки запускаются средствами библиотеки Boost [10], которая позволяет использовать множество потоков выполнения с разделяемыми данными в памяти в коде на C++.

Библиотека Boost имеет классы и функции для управления потоками, а также для синхронизации данных между потоками и для доступа к различным копиям данных в разных потоках. Такой подход впервые был использован в молекулярно-динамическом моделировании [11] на основе эффективного использования нескольких ГПУ в режиме master/slave, когда рабочие потоки ЦПУ содержат CUDA-контекст, а поток-мастер посылает сообщения рабочим потокам процессора. Были предложены синхронная и асинхронная процедуры для передачи вызова функций CUDA графическим устройствам. Синхронная процедура ожидает завершения всех процедур, запущенных на графических устройствах, асинхронная же лишь инициирует выполнение процедур на ГПУ, после чего сразу возвращает управление основному потоку ЦПУ, что позволяет, не дожидаясь завершения выполнения процедур на ГПУ, продолжить вычисления на ЦПУ. Такая организация вычислений позволяет одновременно использовать как ресурсы ЦПУ, так и ресурсы нескольких ГПУ. В нашем случае асинхронный вызов процедур возможен потому, что мы избавились от необходимости обменов между ГПУ.

В этой модели матрично-векторное произведение реализуется следующим образом. На первом этапе происходит обмен внешними элементами вектора между ЦПУ узлов кластера посредством библиотеки MPI. На втором этапе с помощью асинхронной передачи вызова функций CUDA инициируются следующие операции: копирование вектора из памяти ЦПУ в память каждого ГПУ, процедура умножения матрицы на вектор на каждом ГПУ, а также копирования результатов из памяти ГПУ в память ЦПУ. Не дожидаясь завершения этих операций, выполняется третий этап умножения оставшейся части матрицы (соответствующей множеству граничных элементов) на вектор на ЦПУ. После завершения последнего этапа на ЦПУ выставляется запрос на завершение операций на ГПУ. При такой организации вычислений мы добиваемся одновременного использования ресурсов сразу нескольких ГПУ и ресурсов ЦПУ.

3.5. Организация данных. Архитектура гетерогенного кластера имеет сложную иерархию памяти: во первых, это память ЦПУ, а во вторых — память ГПУ. К данным в памяти графического устройства невозможно обратиться напрямую из памяти ЦПУ. Для расчетов на ГПУ требуется предварительно выделить память необходимого размера на ГПУ, а затем скопировать данные из памяти ЦПУ в память ГПУ. Поэтому для расчетов на гетерогенном кластере, в соответствии с описанными шаблонами параллелизма, требуется распределить разреженную матрицу и векторы как в памяти ЦПУ, так и в памяти ГПУ. Для выполнения операции умножения разреженной матрицы на вектор на узле с несколькими графическими устройствами нами модифицирован формат DMSR — формат распределенной модифицированной разреженной строки, который является обобщением формата MSR [2, 3].

Структура данных для хранения матрицы в формате DMSR на каждом ГПУ узла гетерогенного кластера 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. Поскольку количество ненулевых элементов в строках матрицы неодинаково (это типично для матриц нерегулярной структуры), то ГПУ будут иметь различное количество ненулевых элементов, но близкое к их среднему арифметическому. Для более детального обсуждения формата DMSR и его использования для нескольких графических устройств необходимо обратиться к [3].

Следует заметить, что декомпозиция второго уровня производится только на этапе формирования матрицы системы уравнений, т.е. перед стартом итерационного процесса, и происходит автоматически в зависимости от количества ГПУ, которые пользователь намеревается использовать для решения своей

задачи.

4. Численные результаты. Представленные в библиотеке методы протестированы на примере численного решения задачи нестационарной фильтрации жидкости к скважинам в трехмерном пласте сложной конфигурации. Проведено сравнение производительности вычислений для различных моделей параллелизма на гетерогенном кластере "ГрафИТ!/GraphIT!" (НИВЦ МГУ им. М.В. Ломоносова) [12]. Суперкомпьютер занимает две стойки по 16 вычислительных узлов. Каждый из узлов имеет по 2 процессора Intel®Xeon X5650, 24 ГБ, 3 ГПУ NVIDIA "Fermi" Tesla M2050 (515 ГФлоп/с, 3 ГБ ОЗУ, ESS). Узлы объединены в сеть посредством QDR Infiniband 4x через коммутатор QDR Infiniband на 36 портов. В качестве управляющей сети используется Gigabit Ethernet. Система имеет 32 процессора, 192 ядра, 48 ГПУ, 768 ГБ (ОЗУ ЦПУ), 144 ГБ (ОЗУ ГПУ). В качестве программного обеспечения используется ОС RedHat Enterprise Linux, версия 5, библиотека OpenMPI 1.4.2, система управления заданиями Cleo, компилятор gcc 4.1.2, NVIDIA CUDA Toolkit и CUDA SDK версии 3.2, включая компилятор nvcc, а также другие средства программирования [12].

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 и проницаемостью в трехмерной 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. [13]. При учете анизотропии системы трещин и блоков матрицы породы модель описывается следующей системой уравнений:

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

Здесь $(x, y, z) \in D$, $0 < t < T$, t — время, T — общее время исследований, $\beta_l^* = \beta_{\text{скл}} + m_l\beta_{\text{ж}}$ — коэффициент упругоемкости пласта, $\beta_{\text{скл}}$ — 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. 2 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:

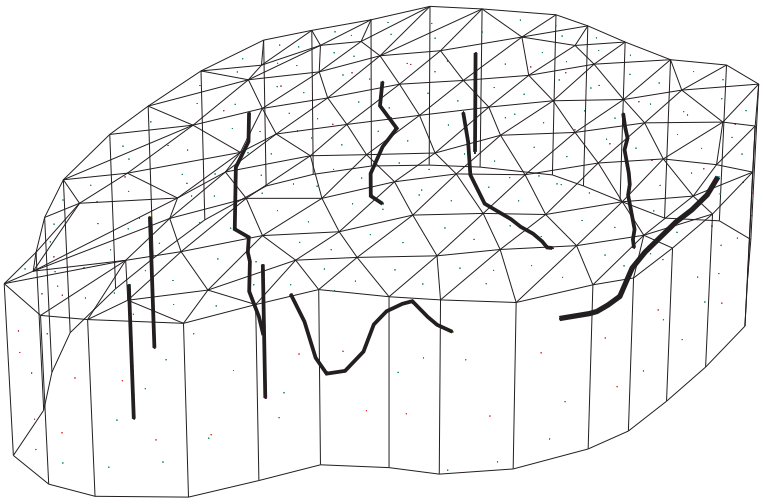

Рис. 2. Схема пласта и расположение скважин

$$\begin{aligned} &p_1(x, y, z, t) = p_2(x, y, z, t) = p_{wi}(t), \quad 0 \leqslant t \leqslant T, \quad (x, y, z) \in \partial S_i^p, \quad i = 1, 2, \ldots, N_w^p, \\ &\left(\frac{\boldsymbol{k}_1}{\mu}\nabla p_1,\, \boldsymbol{n}_j\right) + \left(\frac{\boldsymbol{k}_2}{\mu}\nabla p_2,\, \boldsymbol{n}_j\right) = q_j(x, y, z, t), \quad 0 \leqslant t \leqslant T, \quad (x, y, z) \in \partial S_j^q, \quad j = 1, 2, \ldots, N_w^q, \end{aligned} \tag{2}$$

где $p_{wi}(t)$ — 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:

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

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, $\partial D = \partial D^q \cup \partial D^q$ — 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:

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

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:

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

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 [8].

Таблица 1

Загрузка центральных процессоров

Кол-во процессоров	Кол-во элементов, тыс.	Кол-во неизвестных, тыс.	Кол-во ненулевых элементов матрицы, тыс.
1	4263.344	1523.470	46340.996
2	2098.826 2205.684	743.479 779.991	22573.737 23767.259
...	...	...	...
10	426.157 426.403 414.480 426.475 438.528 426.320 426.214 439.124 408.924 430.719	153.523 153.648 146.943 152.647 156.356 153.025 151.309 156.505 145.005 154.509	4628.707 4700.486 4518.507 4706.225 4674.916 4635.907 4583.459 4752.943 4465.519 4674.327

4.2. Результаты расчетов. Тестовые расчеты проводились на сетке с 761.730 тыс. узлов. 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 1523.470 тыс. Для разделения конечно-элементной сетки на непересекающиеся подобласти использовалась библиотека подпрограмм Metis [6], предназначенная для разделения графов. В табл. 1 представлены данные загрузки центральных процессоров. На рис. 3 представлено разделение расчетной сетки на подобласти.

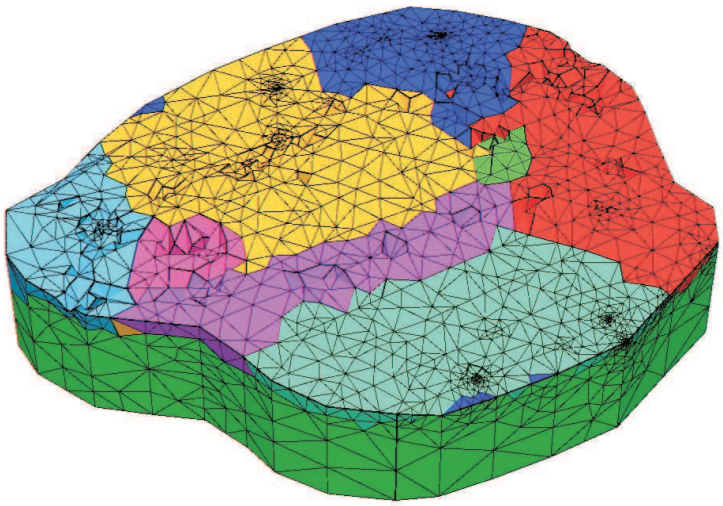

Рис. 3. Сетка расчетной области и ее разделение на подобласти

В табл. 2 приведены данные загрузки графических процессоров на пяти узлах кластера, на каждом из которых используется три графических процессора (n_{gpu}). В столбцах табл. 2 представлено для каждого узла: номер процессора, общее количество неизвестных элементов вектора, модифицируемых на данном узле кластера, количество внутренних элементов вектора для данного процессора, количество граничных элементов вектора, общее количество ненулевых элементов матрицы на узле, количество ненулевых элементов матрицы в строках, соответствующих внутренним элементам вектора, количество ненулевых элементов матрицы на каждом ГПУ, количество элементов вектора, модифицируемых на каждом из ГПУ. Из этой таблицы видно, что количество внутренних элементов вектора и количество соответствующих этим элементам ненулевых значений матрицы гораздо больше, чем количество граничных элементов и количество соответствующих граничным элементам вектора ненулевых значений матрицы. Предложенная схема декомпозиции второго уровня позволяет равномерно загрузить ГПУ на каждом из узлов гетерогенного кластера. Граничные элементы вектора модифицируются на центральном процессоре узла. Разница в количестве граничных элементов вектора на процессорах узлов всегда определяется типом граничных условий [8].

Таблица 2

Загрузка графических процессоров

N_p	N_{update} (тыс.)	N_{internal} (тыс.)	N_{border} (тыс.)	Nnz (тыс.)	Nz_{internal} (тыс.)	N_{updatei} (тыс.)	$nnzi$ (тыс.)
1	303.173	261.623	41.550	9145.805	8157.063	86.530 89.672 85.421	2719.012 2718.997 2719.054
2	303.950	302.890	1.060	9353.092	9045.732	100.869 105.058 96.963	3015.235 3015.249 3015.248
3	306.919	306.238	0.681	9353.953	9255.862	104.014 108.392 93.832	3085.256 3085.282 3085.324
4	305.070	282.844	22.226	9214.878	8693.760	96.183 100.373 86.288	2897.907 2897.911 2897.942
5	304.358	303.856	0.502	9273.268	9257.916	103.093 107.923 92.840	3085.958 3085.961 3085.997

На рис. 4–6 приведены графики ускорения вычислений, полученные для сформулированной выше тестовой задачи, на вычислительном кластере “ГрафИТ!/GraphIT!” для методов: BICGSTAB, CGS, GMRES с предобусловливанием полиномами метода наименьших квадратов. Все расчеты проводились с двойной точностью. На графиках сравнивается ускорение, полученное для различных моделей параллелизма, которые обсуждались в этой статье. Ускорение вычислений подсчитывалось делением времени решения задачи на одном процессоре кластера на время t_{n_p} решения на n_p процессорах кластера для модели MPI и делением на время $t_{n_p \times n_{\text{gpu}}}$ решения на n_p узлах кластера с использованием n_{gpu} графических устройств для моделей MPI+CUDA, MPI+OpenMP+CUDA и MPI+Boost+CUDA.

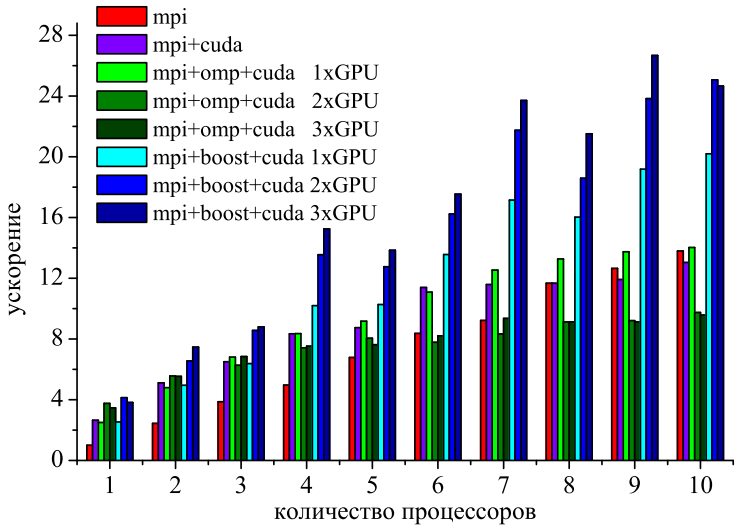

Рис. 4. Ускорение вычислений на гетерогенном кластере для метода BICGSTAB с полиномиальным предобусловливанием методом наименьших квадратов

Из графиков ускорения видно, что расчет с использованием технологии MPI (библиотека Aztec) позволяет получить ускорение на 10 процессорах почти в 14 раз. При этом интенсивность ускорения с увеличением числа процессоров уменьшается, что объясняется тем, что, несмотря на уменьшение объема данных, обрабатываемых каждым процессором, возрастает доля, приходящаяся на коммуникации между процессорами узлов кластера.

Ускорение, полученное с помощью модели MPI+CUDA, т.е. когда на каждом узле кластера используется только одно ГПУ, сначала возрастает почти в два раза по сравнению с моделью MPI, но с увеличением числа узлов (соответственно и числа ГПУ) разница в ускорениях постепенно уменьшается, а начиная с 8 узлов ускорение становится меньше, чем ускорение в модели MPI. Это объясняется тем, что в модели MPI+CUDA на одном из этапов умножения матрицы на вектор для обмена данными между ГПУ различных узлов необходимо копировать данные из памяти ГПУ в память ЦПУ и обратно (см. п. 3.2). Такое копирование данных, начиная с некоторого количества узлов, становится менее результативным по сравнению с непосредственным умножением вектора на ЦПУ в MPI модели (с увеличением количества используемых узлов кластера объем данных для каждого ЦПУ становится меньше и, следовательно, уменьшаются задержки при чтении данных из памяти ЦПУ).

В модели MPI+OpenMP+CUDA, в отличие от MPI+CUDA, на ГПУ происходит только умножение матрицы на вектор, а все векторы находятся в памяти ЦПУ и все операции с векторами реализуются на ЦПУ (следовательно, отсутствует по крайней мере одно копирование вектора из памяти ГПУ в память ЦПУ) по сравнению с моделью MPI+CUDA. Поэтому ускорение в модели MPI+OpenMP+CUDA, в случае использования одного ГПУ на каждом узле кластера, больше, чем в модели MPI и MPI+CUDA, но до тех пор, пока объем данных не станет таким, что ЦПУ в модели MPI будет быстрее обрабатывать их, чем в MPI+OpenMP+CUDA (начиная с 9 узлов). Иными словами, ускорение больше до тех пор, пока объем данных для кэш-памяти ЦПУ достаточно велик и задержки, связанные с их чтением из памяти ЦПУ, дольше, чем копирование данных на ГПУ. При использовании двух и трех ГПУ на каждом узле кластера в модели MPI+OpenMP+CUDA ускорение, как видно на рис. 4–6, больше, чем ускорение при использовании одного ГПУ только тогда, когда количество используемых узлов кластера не превышает двух. При дальнейшем увеличении количества узлов использование двух и более ГПУ

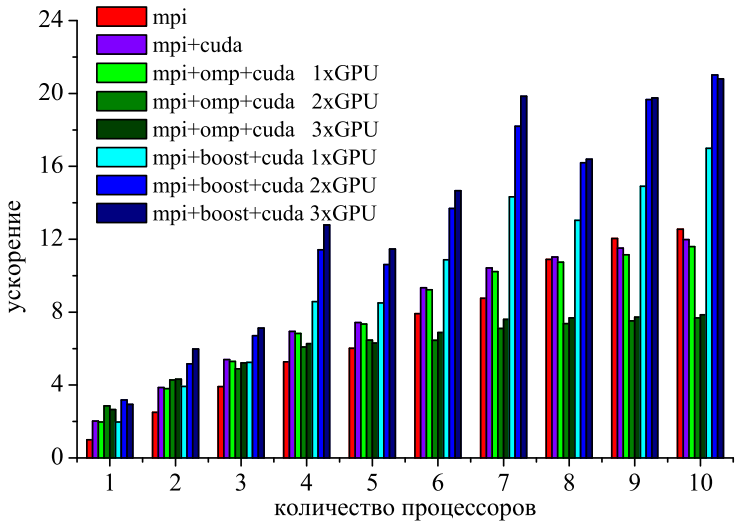

Рис. 5. Ускорение вычислений на гетерогенном кластере для метода CGS с полиномиальным предобусловливанием методом наименьших квадратов

в модели MPI+OpenMP+CUDA на каждом узле становится неэффективным не только по сравнению с одним ГПУ, но и с моделями MPI+CUDA и MPI, что, по-видимому, связано со спецификой порождения/завершения потоков в параллельной области в технологии OpenMP. Другого объяснения замедления расчетов для двух и трех ГПУ на узле в модели MPI+OpenMP+CUDA мы не находим. Это также подтверждается результатами, полученными при использовании модели MPI+Boost+CUDA. Как видно на рис. 4–6, эта модель позволяет получить максимальное ускорение по сравнению со всеми предыдущими. Наибольшее ускорение получено при использовании трех ГПУ на каждом узле. Результаты расчетов с помощью этой модели показывают также некоторую немонотонность ускорения, например для 5, 8 и 10 узлов на рис. 4, что объясняется неравномерностью распределения граничных элементов вектора (табл. 2) на каждом узле в случае использования граничных условий Неймана на скважинах [8]. Результаты, полученные для методов CGS и GMRES, имеют на рис. 5 и 6 почти такой же характер.

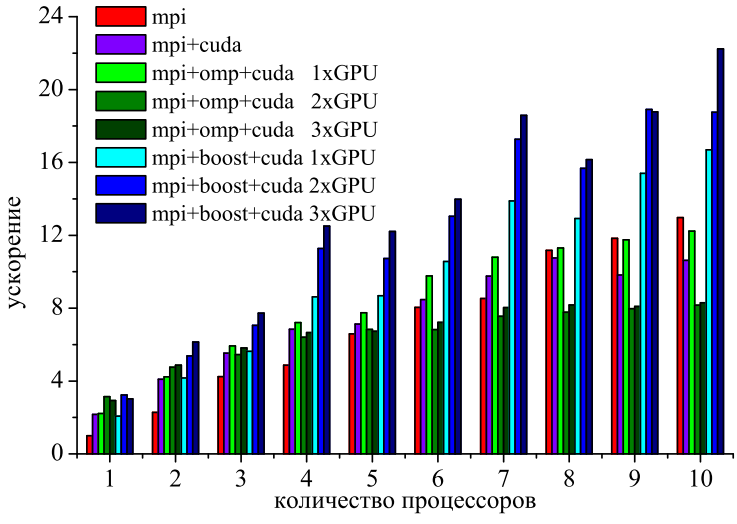

Рис. 6. Ускорение вычислений на гетерогенном кластере для метода GMRES с полиномиальным предобусловливанием методом наименьших квадратов

Таким образом, мы продемонстрировали некоторые эффективные применения библиотеки шаблонов итерационных методов подпространств Крылова с предобусловливанием на гетерогенном кластере с

графическими ускорителями NVIDIA.

5. Заключение. Построена библиотека gpu_sparse итерационных методов подпространств Крылова с предобусловливанием для решения разреженных СЛАУ с нерегулярной структурой, как симметричных, так и несимметричных на гетерогенных вычислительных системах, на которых вычисления на центральных процессорах совмещаются с вычислениями на графических процессорах NVIDIA. Библиотека записана в виде шаблонов классов на C++, что позволяет проводить вычисления с одинарной и с двойной точностью, а также позволяет скрыть от пользователя коммуникации, как между вычислительными узлами, так и между процессорными ядрами и графическими ускорителями.

Для организации вычислений используются такие технологии, как CUDA, MPI, OpenMP, Boost. Представлены различные шаблоны параллелизма, использующие преимущества архитектуры гетерогенного кластера, а также исследована их эффективность.

Предложен новый способ декомпозиции области, который позволяет получить дополнительное измельчение области без увеличения нагрузки на коммуникационную сеть.

Как видно из представленных расчетов, использование гетерогенного кластера на базе графических ускорителей позволяет значительно повысить производительность и эффективность вычислений.

Выражение признательности. Работа выполнена в рамках конкурса "Эффективное использование GPU-ускорителей при решении больших задач" компании Т-Платформы при поддержке МГУ им. М.В. Ломоносова, а также в рамках программы Президиума РАН № 14 "Интеллектуальные информационные технологии, математическое моделирование, системный анализ и автоматизация". Авторы благодарны Оргкомитету конкурса, а также НИВЦ МГУ им. М.В. Ломоносова за предоставленную возможность использования гетерогенного суперкомпьютера "ГрафИТ!/GraphIT!".

СПИСОК ЛИТЕРАТУРЫ

1. *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 // Вестн. Нижегородского ун-та им. Н.И. Лобачевского. Сер.: Информационные технологии. 2011. № 1. 205–212.
2. *Губайдуллин Д.А., Никифоров А.И., Садовников Р.В.* Библиотека gpu_sparse для численного решения задач механики сплошных сред на гибридной вычислительной системе // Вестн. Нижегородского ун-та им. Н.И. Лобачевского. Сер.: Информационные технологии. 2011. № 2. 190–196.
3. *Губайдуллин Д.А., Никифоров А.И., Садовников Р.В.* Библиотека шаблонов итерационных методов подпространств Крылова для численного решения задач механики сплошных сред на гибридной вычислительной системе // Вычислительные методы и программирование. 2010. **11**, № 2. 171–179.
4. NVIDIA Corporation. NVIDIA CUDA C. Programming Guide. May, 2011. Version 4.0.
5. *Hendrickson B., Leland R.* The Chaco user's guide. Version 1.0. Technical Report Sand93-2339. Sandia National Laboratories. Albuquerque, 1993.
6. METIS: family of multilevel partitioning algorithms (http://glaros.dtc.umn.edu/gkhome/views/metis).
7. Aztec: a massively parallel iterative solver library for solving sparse linear systems (http://www.cs.sandia.gov/CRF/-aztec1.html).
8. *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.
9. OpenMP Architecture Review Board (http://www.openmp.org).
10. Boost C++ Libraries (http://www.boost.org).
11. *Anderson J.A., Lorenz C.D., Travesset A.* General purpose molecular dynamics simulations fully implemented on graphics processing units // J. of Computational Physics. 2008. **227**, N 10. 5342–5359.
12. Суперкомпьютер "ГрафИТ!" / "GraphIT!" на основе графических процессоров (http://gpu.parallel.ru/graphit.-html).
13. *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
14.10.2011