О решении систем линейных алгебраических уравнений на многоядерных вычислительных системах с графическими ускорителями*

Б.И. Краснопольский[1,2], А.В. Медведев[2]

НИИ механики МГУ[1], ЗАО "Т-Сервисы"[2]

В работе обсуждаются текущие результаты исследования, проводимого авторами с целью поиска и разработки эффективных алгоритмов распараллеливания методов решения больших разреженных систем линейных алгебраических уравнений на современных многопроцессорных системах. Рассматриваются проблемы, возникающие при реализации итерационных методов подпространства Крылова, в частности стабилизированного метода бисопряжённых градиентов (BiCGStab), с алгебраическим многосеточным предобуславливателем на многопроцессорных системах с гибридными узлами, имеющими в своем составе как многоядерные процессоры, так и графические ускорители. Приводятся результаты анализа наиболее эффективных алгоритмов распараллеливания и технологических приёмов реализации для ряда задач, возникающих при аппроксимации эллиптических уравнений на больших сетках.

1. Введение

Разработка и реализация эффективных методов решения эллиптических уравнений является необходимым условием для решения подавляющего большинства задач механики сплошных сред. Зачастую этап решения таких уравнений является одним из наиболее трудоемких и может занимать более 90% от общего времени решения задачи. В зависимости от формы расчетных областей и топологии используемых сеток на практике могут применяться как прямые, так и итерационные методы решения систем уравнений, получаемых при разностной аппроксимации исходных эллиптических уравнений. Прямые быстрые методы, например [1], имеют ограниченную область применимости, однако в ряде случаев оказываются существенно более эффективными при проведении последовательных расчетов. Итерационные методы в свою очередь требуют большего времени расчета, но при этом обладают большей гибкостью относительно вида матриц систем линейных алгебраических уравнений (СЛАУ) и хорошим параллелизмом. Наиболее широко используемыми в настоящее время для решения больших сильно-разреженных СЛАУ для эллиптических уравнений являются итерационные методы подпространства Крылова [2] и/или многосеточные методы [3]. Многосеточные методы могут использоваться как отдельные решатели, либо применяться в качестве предобуславливателей для итерационных методов. С точки зрения эффективности они сравнимы с методами неполной факторизации [2] при последовательных расчетах, но обладают при этом рядом ключевых особенностей, критических при параллельной реализации методов. К таковым можно отнести инвариантность многосеточных методов относительно переупорядочения элементов матрицы, что играет существенную роль при построении псевдо-оптимального разбиения матрицы между вычислительными процессами, а также слабую зависимость скорости сходимости методов от размера задачи (например, [4]).

Современная архитектура вычислительных систем предполагает наличие нескольких многоядерных процессоров внутри каждого узла (в настоящее время, 12-48 ядер в одном узле), которые могут быть дополнены одним или несколькими ускорителями (GPU, FPGA,

*Работа выполнена при поддержке РФФИ (гранты № 11-01-00088-а и 12-01-31002-мол_а) и компании ЗАО "Т-Сервисы".

Intel Xeon Phi). Столь сложная и комплексная архитектура вычислительных узлов приводит к необходимости разработки соответствующих реализаций численных методов, учитывающих все основные особенности имеющейся аппаратуры. Для достижения приемлемого уровня масштабируемости и эффективности методов необходим пересмотр алгоритмов распараллеливания численных методов с целью выделения нескольких иерархических уровней параллелизма, отвечающих аппаратной платформе.

Отдельную обширную тему для исследований представляет эффективное распараллеливание вычислений для задач линейной алгебры на графических ускорителях. В литературе известно большое количество работ в различных областях вычислительной математики, где были опубликованы результаты многократного ускорения вычислений при использовании графических ускорителей. Однако достижения при портировании задач линейной алгебры в большинстве случаев оказываются существенно скромнее. Основная сложность заключается в том, что рассматриваемый класс задач сводится к алгоритмам, в которых преобладают операции чтения из памяти (memory bound), тогда как графические ускорители демонстрируют преимущество на алгоритмах с преобладанием вычислений. Наблюдаемый прирост производительности для таких задач обычно лежит в пределах 1.5-3 по сравнению с одним процессором, и обусловлен в первую очередь более высокой пропускной способностью шины памяти графических ускорителей по сравнению с центральными процессорами.

Несмотря на широкий практический интерес к реализации многосеточных методов на графических ускорителях и большое количество соответствующих проектов, этот вопрос все еще является актуальной темой для исследований. Говоря о текущем состоянии исследований следует упомянуть наиболее интересные работы [5–10], выполняющиеся как крупными компаниями, так и отдельными группами исследователей. Одной из первых библиотек для задач линейной алгебры, распространяемых в исходных кодах, была библиотека CUSP [5]. В ней был реализован набор основных итерационных методов решения СЛАУ, в т.ч. один из семейства многосеточных методов, однако реализация этих методов до сих пор ограничивается возможностью использования только одного графического ускорителя. Схожий функционал реализован в библиотеке CUSPARSE [6], разрабатываемой компанией NVIDIA, и распространяемой только в бинарном виде. По сравнению с CUSP, данная библиотека обладает чуть более высокой производительностью, но не содержит реализации многосеточных методов. Реализация многосеточных методов была официально анонсирована компанией NVIDIA в библиотеке NVAMG [7] в конце 2012 года, но также позволяет проводить расчеты только на одном ускорителе. Релиз следующей версии библиотеки с возможностью проведения расчетов на нескольких ускорителях анонсирован на середину 2013 года и эта библиотека должна войти в состав пакета ANSYS Fluent 15. Схожая ситуация с проектом GAMPACK [8], в котором сделана попытка портировать многосеточный метод из библиотеки hypre: опубликованы результаты для расчетов на одном ускорителе, тогда как распределенная версия, по заявлениям разработчиков, находится на стадии финального тестирования. Проект LibAMA [9] также ориентирован на разработку распределенной версии многосеточных методов, базирующейся на библиотеке hypre, однако в отличие от предыдущих двух библиотек является исследовательской и свободно распространяемой реализацией. В [11] опубликованы результаты исследования масштабируемости реализованных многосеточных методов на нескольких тестовых задачах в простых расчетных областях. Однако проведенные тесты этой библиотеки показали, что эффективность реализации базового блока для итерационных методов, произведения разреженной матрицы на вектор, уступает ряду других реализаций. Еще один проект Cufflink [10] является в некотором смысле промежуточным между последовательной и полностью распределенной реализациями и основан на принципе “domain decomposition”. При этом и в данном случае распределенная версия кода находится в режиме отладки и тестирования, а представленные результаты и отзывы пользователей свидетельствуют о наличии проблем с масштабируемо-

стью методов при решении практических задач на реальных топологиях расчетных сеток.

Приведенный выше список проектов не претендует на полноту обзора состояния вопроса разработки библиотек численных методов для распределенных вычислений на графических ускорителях, однако, по мнению авторов, в нем отражены наиболее интересные проекты. Несмотря на большой практический интерес к распределенной реализации многосеточных методов, стабильными хорошими результатами масштабируемости пока не отличается ни один из заявленных в открытых источниках проектов. Исходя из этого, целью настоящей работы является: совершенствование алгоритмов распараллеливания итерационных методов подпространства Крылова и многосеточных методов для многоядерных процессоров и реализация параллельной версии в рамках гибридной модели программирования, а также проработка базовых алгоритмических вопросов по переносу части вычислений на графические ускорители, и в перспективе - сопроцессоры Intel Xeon Phi.

2. Основные типы вычислительных операций

Наиболее трудоемким этапом реализации итерационных методов является разработка хорошо масштабируемого набора функций для выполнения операции произведения разреженной матрицы на вектор. В общем случае, никакое распределение данных между вычислительными процессами не обеспечивает полной независимости по данным между параллельно выполняющимися частями алгоритма, и требуется обмен элементами вектора-множителя, находящегося на соседних узлах многоузловой вычислительной системы. Использованная в работе схема распределения данных между вычислительными узлами приведена на рис. 1: матрица и вектора распределеяются по узлам полосами построчно. Коммуникация между узлами в таком случае может осуществляться разными способами. На текущем этапе в работе использованы неблокирующие операции типа “точка-точка” из стандарта MPI-1, однако могут быть использованы и альтернативные механизмы передачи данных. В частности, возможно использование односторонних операций чтения из памяти другого узла. Реализация такого рода операций возможна, например, с использованием операций односторонних коммуникаций из MPI-2 (MPI_Create_window, MPI_Put), или с использованием специальных библиотек, реализующих в том или ином виде идеи разделенного глобального адресного пространства (PGAS) [14], например [15, 16]. Исследование таких вариантов реализации может стать одним из направлений дальнейших исследований.

Исходя из того, что передача данных от одного узла к другому в любых современных коммуникационных системах эффективнее выполняется в виде больших блоков, целесообразно построение алгоритма умножения матрицы на вектор в виде последовательного перемножения частей исходной матрицы, соответствующих частям вектора-множителя, расположенного на каждом из узлов. Для реализации такой схемы предлагается разбиение полос матрицы на блоки пропорционально количеству строк, приходящихся на каждый из вычислительных процессов, а результаты умножения каждого из блоков суммируются после обработки последнего блока матрицы. Здесь целесообразна предварительная оптимизация данных, поскольку большая часть таких блоков не содержит ни одного, либо всего несколько ненулевых элементов. Для недиагональных блоков матрицы возможно провести “сжатие” данных таким образом, чтобы в ходе коммуникаций между процессами пересылались только те значения вектора-множителя, которым соответствуют ненулевые элементы в блоках матриц. Указанные преобразования позволяют существенно уменьшить размер пересылаемых векторов и шаблон коммуникаций вычислительных процессов, сократив количество пересылок в среднем до порядка десяти на один MPI-процесс. При этом, за счет использования неблокирующих операций, время на пересылку данных удается эффективно “замаскировать” вычислениями с локальным фрагментом вектора-множителя.

Помимо произведения матрицы на вектор, в используемых в работе итерационных и многосеточных методах фигурируют еще два типа базовых операций. Один из них – сло-

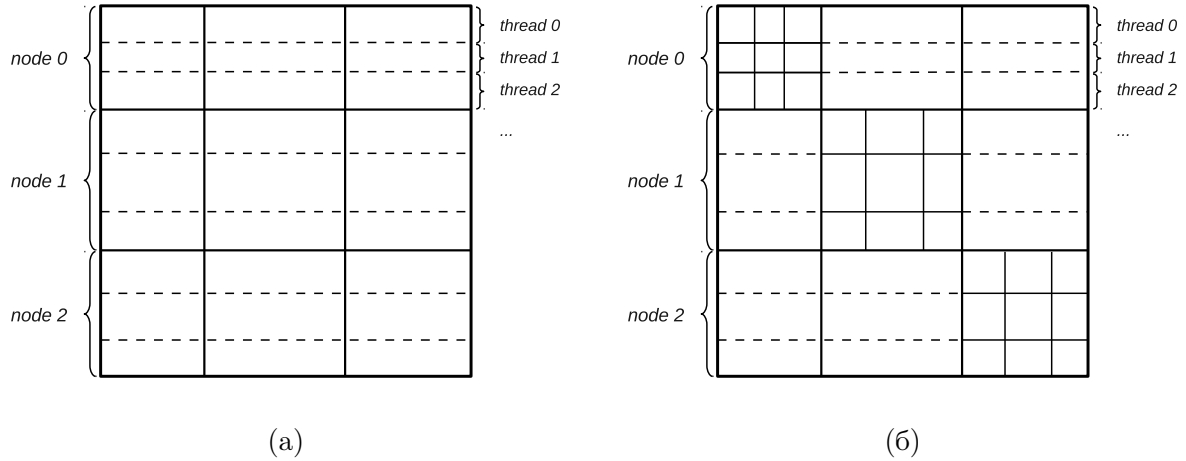
жение векторов/умножения вектора на число. Исходя из принятой схемы разбиения данных между вычислительными процессами, каждый из вычислительных процессов содержит определенные фрагменты векторов, так что указанные операции могут быть выполнены локально без каких-либо коммуникаций с соседними узлами. Еще одной распространенной операцией в итерационных методах является скалярное произведение векторов. Данная операция может быть выполнена локально для имеющихся фрагментов векторов, однако после этого необходимо выполнение глобальной редукции по всем вычислительным процессам, что приводит к возникновению точки глобальной синхронизации всех вычислительных процессов, сказывается на масштабируемости при использовании большого количества вычислительных ядер. Возможные варианты минимизации влияния таких операций на масштабируемость выходят за рамки настоящей статьи, но обсуждались ранее в [4, 13].

3. Гибридная модель MPI+Posix ShM для многоядерных процессоров

"Стандартный" MPI-подход распараллеливания приложения на практике оказывается неприменимым не только при использовании различных ускорителей, но и при проведении вычислений на многоядерных центральных процессорах. При запуске большого количества MPI-процессов на узлах для вычислительно-интенсивных приложений, активно использующих ресурсы коммуникационной сети для пересылки сообщений, возникает перегрузка адаптеров коммуникационной сети, что приводит к увеличению времени передачи сообщений и деградации масштабируемости приложения. Ситуация несколько улучшается с уменьшением количества процессов, распределяемых на каждый из вычислительных узлов, однако это приводит к неэффективной загрузке ресурсов и простою значительной части оборудования. Подробный анализ зависимости эффективности вычислений для задач линейной алгебры от количества вычислительных узлов и используемых ядер обсуждался в одной из предыдущих работ [12].

Решением данной проблемы является использование гибридных моделей параллельного программирования, когда на каждый вычислительный узел приходится только один коммуникационный MPI-процесс, а обмен данными и распараллеливание вычислений внутри узла реализуется тем или иным способом через общую память (OpenMP, Posix Shared Memory, Pthreads и пр.). Такой подход, в зависимости от шаблона коммуникаций приложения, позволяет на один-три порядка сократить общее количество сообщений, пересылаемых через коммуникационную сеть. Первые результаты по разработке двухуровневой схемы распараллеливания в рамках гибридной модели вычислений были представлены авторами в работах [4,12,13]. В настоящей статье обсуждается дальнейшее усовершенствование предложенной схемы многоуровневого распараллеливания вычислений и ее оптимизация с учетом NUMA-архитектуры современных вычислительных систем.

Первая попытка создания алгоритмов распараллеливания выбранных численных методов для решения больших систем линейных алгебраических уравнений в рамках гибридной модели программирования была предпринята в [4]. Изначально разрабатывавшаяся на вычислительной системе СКИФ МГУ "Чебышёв", построенной на базе процессоров с UMA-архитектурой, реализация гибридной модели была достаточно простой. Поскольку все вычислительные ядра имели одинаковую скорость доступа ко всему диапазону оперативной памяти на узле, б*о*льшая часть данных помещалась в область памяти, "видимую" всеми вычислительными процессами (shared memory), а синхронизация и управление доступом к памяти теми или иными вычислительными процессами были реализованы посредством семафоров (рис. 1.а). Разработанная реализация демонстрировала хорошие результаты масштабируемости в сравнении с MPI-версией на вычислительных системах с UMA-архитектурой, однако оказалась не столь эффективной для NUMA-архитектуры [12]: результаты тестов на 24-ядерных узлах на базе процессоров AMD Opteron на одном узле демонстрировали 3-



(а) (б)

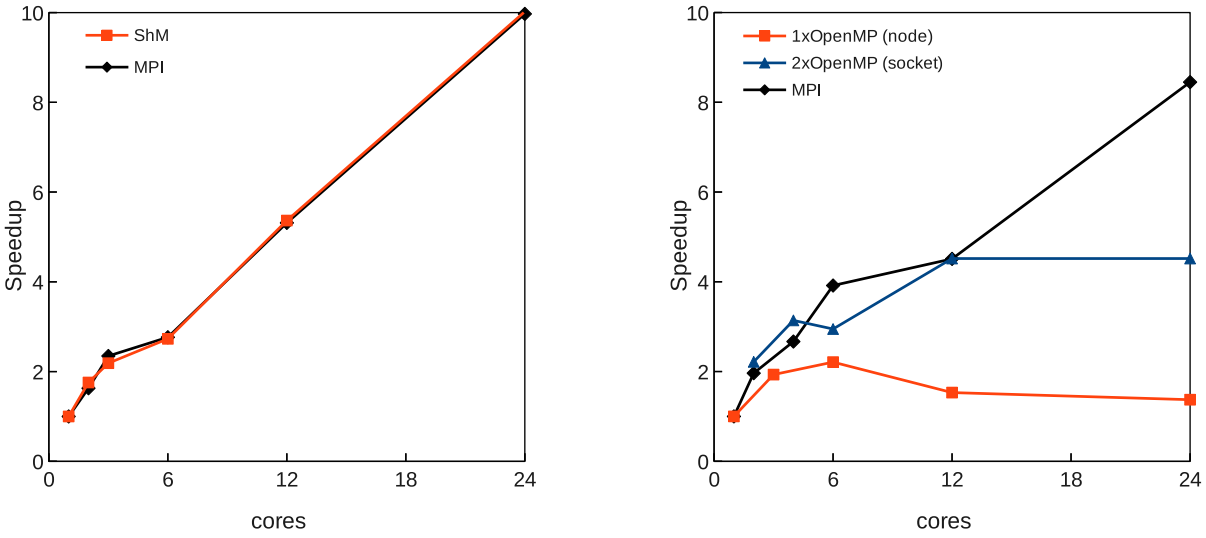
Рис. 1. Схема распределения матрицы между вычислительными процессами: (а) исходная версия для UMA-архитектуры, (б) модифицированная версия для NUMA-архитектуры

кратное превосходство MPI-реализации над гибридной моделью. Данный факт был вызван особенностями организации доступа к памяти. Область общей для всех процессов памяти выделялась одним процессом, в результате чего она физически оказывалась расположенной преимущественно в пределах одного NUMA-узла[1]. Для процессов, запущенных на вычислительных ядрах из других NUMA-узлов, происходило обращение к памяти не локального NUMA-узла через интерфейс HyperTransport, что существенно замедляло скорость доступа к памяти.

Основной целью переработки исходного алгоритма распараллеливания стало изменение схемы распределения данных и их локализация в пределах ближайшего NUMA-узла. Для этого была пересмотрена стратегия разбиения данных между процессами внутри вычислительного узла. Разбиение матриц было реализовано в виде двухэтапной процедуры. Первоначально выполняется разбиение данных между узлами. После этого диагональные блоки матрицы повторно разбиваются на подблоки по количеству вычислительных процессов, запускаемых внутри одного узла (рис. 1.б), но располагаются в ближайших к использующим эти данные ядрам NUMA-банках памяти. При этом, фрагменты вектора-множителя каждого из процессов, как и в предыдущей схеме, остаются видимыми для всех процессов внутри узла, но также располагаются в памяти в локальных для вычислительных процессов NUMA-банках памяти.

Реализованная модификация схемы распределения данных в памяти вычислительного узла позволила существенно улучшить эффективность гибридной версии численных методов. Для тестовых расчетов была выбрана задача решения СЛАУ, полученная при аппроксимации уравнения Пуассона в кубической области на равномерной сетке размером 150^3 ячеек. В качестве решателя использован итерационный метод BiCGStab с алгебраическим многосеточным предобуславливателем. Для построения иерархии матриц многосеточного метода использована библиотека hypre [17]. Разработанная гибридная модель в пределах одного узла демонстрирует результаты масштабируемости, совпадающие с MPI-версией программы и обеспечивает 10-кратное ускорение вычислений (рис. 2.а). Для сравнения на рис. 2.б приведены результаты масштабируемости идентичных методов, реализованных в библиотеке hypre в рамках гибридной модели MPI+OpenMP. При в целом сравнимой производительности MPI-версий решателей (hypre демонстрирует ускорение в 8.6 раза), эффективность реализованной в hypre гибридной модели MPI+OpenMP оказывается существенно хуже, и обеспечивает ускорение всего в 1.4 раза. Несколько лучшие результаты

[1]Использовалась стратегия выделения памяти в максимально близких NUMA-банках памяти.



(а) (б)

Рис. 2. Сравнение масштабируемости различных гибридных моделей в пределах одного узла: (а) разработанные алгоритмы, реализованные в рамках модели MPI+Posix ShM, (б) библиотека hypre, гибридная модель MPI+OpenMP.

наблюдаются при использовании 2 MPI-процессов с 11 OpenMP тредами на каждый (график “2xOpenMP”, ускорение в 4.5 раза), однако и в этом случае они оказываются более чем вдвое хуже результатов предложенной схемы распараллеливания, реализованной в рамках модели MPI+Posix ShM. Наилучшие результаты, по-видимому, могут быть получены для 4 MPI-процессов с 5 OpenMP тредами и с привязкой процессов по NUMA-узлам процессоров, но такая ситуация нивелирует сам эффект от разработки гибридной модели.

4. Применение графических ускорителей для задач линейной алгебры

Вторая часть настоящей работы посвящена проработке основных вопросов использования ускорителей для рассматриваемого класса задач линейной алгебры. Говоря о переносе базовых алгоритмов рассматриваемой задачи, сосредоточимся прежде всего на наиболее трудоёмком алгоритме умножения матрицы на вектор, поскольку на данном этапе исследования эффективная реализация этого алгоритма предоставит прямую возможность для переноса всего программного кода решателя СЛАУ для гибридных вычислительных систем на графические ускорители.

Алгоритм умножения разреженной матрицы на вектор в целом не обладает большими возможностями по поиску вариантов реализации. Соотношение минимального количества чтений из основной памяти, записи в основную память и вычислительных операций выглядит как: $\mathrm{NNZ}+\mathrm{N} : \mathrm{N} : 2\,\mathrm{NNZ}$, где NNZ – количество ненулевых элементов матрицы, N – размер вектора-множителя. Такая оценка предполагает, что все данные, принадлежащие вектору, на который выполняется умножение, могут быть размещены в сверхбыстродействующей памяти (shared memory). Если такое размещение невозможно, или возможно частично, то оценка количества чтений из основной памяти увеличивается, и в предельном случае, когда данные вектора повторно не используются совсем, возрастает до $2\,\mathrm{NNZ}$.

Однако, на практике такое минимально необходимое количество операций чтения из основной памяти достижимо только для разреженных матриц специального вида. Примером такого формата может служить формат DIA [18], предназначенный для хранения диагональных матриц. Для представления разреженных матриц общего вида используют-

ся форматы, хранящие дополнительные массивы, которые позволяют определить значения номера строки и столбца для каждого из элементов матрицы. Вследствие этого, машинное время выполнения операции перемножения разреженной матрицы на вектор может значительно отличаться от теоретического минимума. Простым примером того, насколько существенным может оказаться это отличие, является формат хранения матриц COO [18]. Данный формат предполагает хранение для каждого ненулевого элемента матрицы двух дополнительных индексов: номера строки и столбца элемента. В случае если значения элементов матрицы представлены числами с плавающей точкой одинарной точности, количество требуемых операций чтения возрастает до величины в диапазоне от 3 NNZ+N до 4 NNZ (исходя из тех же соображений о возможном размещении вектора-множителя в сверхбыстродействующей памяти).

Все прочие форматы хранения разреженных матриц также вносят свой дополнительный вклад в объём требуемых операций чтения из основной памяти. Дополнительную трудность в выборе формата хранения и реализации алгоритма умножения матрицы на вектор представляет фактор, специфичный для архитектуры подсистемы доступа к основной памяти графических ускорителей. Для эффективной работы подсистемы памяти графического ускорителя необходимо выполнение требования, предъявляемого к инструкциям чтения из основной памяти. Это требование, называемое coalescing [20], предполагает, что в рамках одной параллельно выполняемой соседними нитями операции чтения из памяти запросы на чтение должны относиться к последовательно расположенным в памяти адресам. Кроме того, необходимо выполнение ряда требований по выравниванию таких адресов. Невыполнение этого требования приводит к снижению эффективности чтения из основной памяти. Ниже кратко описано несколько форматов хранения разреженных матриц, для каждого из которых обсуждаются особенности соблюдения этого условия при построении алгоритма умножения матрицы на вектор.

4.1. CSR формат

Формат хранения CSR [18], помимо основного массива значений, предполагает наличие двух дополнительных массивов. Один из них равен размеру массива значений и содержит соответствующие номера столбцов для ненулевых элементов матрицы. Второй массив хранит индексы начала строк, и его размер меньше или равен размеру вектора-множителя. Общее количество необходимых операций чтения для рассматриваемого алгоритма минимально оценивается как 2 NNZ+2 N. Очевидный способ построения алгоритма умножения матрицы на вектор в массивно-параллельной парадигме ставит в соответствие каждой параллельно выполняющейся нити один элемент массива индексов строк. Таким образом, параллельно выполняется цикл умножения каждой строки на вектор-множитель. Этот подход имеет две принципиальных проблемы при реализации на графических ускорителях. Во-первых, здесь нарушается правило coalesced чтений входных данных, поскольку, в общем случае, нити с соседними номерами не будут читать массивы номеров столбцов и входных значений по последовательным адресам: между адресами, по которым расположены первые элементы каждой строки, есть смещение, равное длине соответствующей строки. Во-вторых, нити могут обладать значительно отличающейся от соседних нитей последовательностью выполнения команд (divergent branching), что также является недостатком реализации, влияющим на общую производительность алгоритма на графическом ускорителе [20].

Алгоритм, который позволяет более эффективно выполнить операцию умножения матрицы, представленной в CSR формате, на вектор [21], предполагает сопоставление одному элементу матрицы индексов строк не одной нити, а группы из 32-х нитей, которые составляют единицу планирования для SIMD-процессоров графических ускорителей. Таким образом, нити входящие в одну 32-нитевую группу будут читать данные, относящиеся к одной и той же строке, а значит, чтение в рамках этого блока будет выполняться по последова-

тельным адресам. Однако, алгоритм работает идеально только в тех случаях, когда размер каждой строки матрицы кратен 32. В остальных же случаях часть пропускной способности шины памяти всё же будет расходоваться впустую, поскольку в блоке будут появляться простаивающие нити из-за того, что им не соответствует ни один ненулевой элемент данной строки. Такой алгоритм широко известен и реализован, в том числе, в рамках открытой библиотеки программ для графических ускорителей CUSP [5].

Существует также описание алгоритма умножения матрицы в формате CSR на строку с "балансировкой" рабочей нагрузки по всем параллельным нитям, предполагающий предварительное применение алгоритма Segmented Scan. Описание алгоритма можно найти на сайте автора [22], однако используется ли данный алгоритм в каких-либо из библиотечных кодах неизвестно. Внедрение такого алгоритма может быть одной из перспективных тем для дальнейшего исследования.

4.2. ELLPACK формат

Формат хранения разреженных матриц ELLPACK исключает массив, хранящий или указывающий на номера строк. Вместо этого, для каждой строки хранится S элементов массива, где S – максимальная длина строки в данной матрице. Каждый элемент содержит либо значение соответствующего по порядковому номеру ненулевого элемента строки матрицы, либо маркер отсутствия элемента, если строка содержит меньше, чем S ненулевых элементов. Дополнительно хранится аналогичный по структуре массив, содержащий номера столбцов для соответствующего ненулевого элемента матрицы.

Общее количество элементов, которые необходимо прочитать из основной памяти, чтобы выполнить операцию умножения матрицы, представленной в формате ELLPACK, на вектор, минимально оценивается как (2 S+1)*N. Если все строки матрицы содержат некоторое фиксированное число ненулевых элементов, то произведение S*N окажется равным NNZ, и требуемое количество операций чтения будет равно 2 NNZ+N, что несколько лучше, чем аналогичный минимальный показатель для CSR-матриц. Однако, если условие равенства количества ненулевых элементов во всех строках не выполняется, то матрица, записанная в ELLPACK-формате будет содержать некоторое количество элементов-маркеров, которые заметно увеличат необходимый объём считываемой информации. Таким образом, эффективность применения формата ELLPACK для описываемого алгоритма существенным образом зависит от такого свойства матрицы, как равномерность распределения числа ненулевых элементов по её строкам.

Оценивая возможность построения алгоритма умножения матрицы на вектор, использующего графические ускорители, можно отметить, что ни проблема, касающаяся нарушения последовательности доступа к памяти (coalescing), ни проблема различных путей выполнения алгоритма для соседних нитей (divergent branching) для данного формата не возникают, или, по крайней мере, в отношении divergent branching, не стоят так остро. Возможна такая реализация массивно-параллельного варианта алгоритма, при котором каждой нити соответствует свой элемент двух матриц, составляющих формат ELLPACK. Адресация считываемых данных соседними нитями при этом будет полностью последовательной.

4.3. HYB формат

Компромиссный формат HYB предполагает устранение принципиальной проблемы формата ELLPACK: для строк матрицы, количество ненулевых элементов в которых превышает некоторое пороговое значение S, используется формат хранения COO. Таким образом, задача умножения матрицы на вектор преобразуется в две такие задачи, выполняемые для двух подматриц. Такой формат расширяет границы возможности эффективного применения формата ELLPACK, реализацию которого на графических ускорителях можно назвать

достаточно выигрышной. Однако, такой формат требует проведения предварительного анализа исходной матрицы с целью определения оптимальной величины S, и реализация которого может потребовать дополнительных усилий, а выполнение – дополнительного машинного времени.

Подводя итог краткому рассмотрению форматов, следует отметить, что здесь описаны не все возможные форматы хранения разреженных матриц. Однако, выбор именно этих форматов для первоочередного исследования продиктован простым практическим соображением – все эти форматы (исключая, в некоторой степени, HYB), требуют минимальных алгоритмических усилий по разбиению матрицы на блоки по столбцам и строкам, и операций сжатия, упомянутых в разделе 2 настоящей статьи. Эти операции являются необходимой алгоритмической составляющей программы решения СЛАУ на параллельных системах с большим количеством узлов, поэтому необходимость их эффективной реализации для выбранного формата следует иметь в виду с самого начала работы над алгоритмом. Форматы CSR и ELLPACK обладают достаточно хорошим потенциалом по созданию эффективных массивно-параллельных алгоритмов для их преобразований.

Как было показано ранее, оценка эффективности реализации алгоритма умножения матрицы на вектор на графических ускорителях, существенным образом зависит от свойств и распределения ненулевых элементов в матрице. Исходя из этого, для экспериментального анализа эффективности операции произведения матрицы на вектор для различных форматов хранения матриц был использован набор тестовых матриц, полученных в ходе построения иерархии матриц алгебраического многосеточного предобуславливателя. Исходный размер матрицы составлял 8 млн. ячеек, основные параметры набора тестовых матриц приведены в табл. 1. В табл. 2 представлено сравнение машинного времени выполнения процедур умножения матрицы на вектор для тестового набора матриц. Для сравнения использовалась реализация алгоритмов из библиотеки CUSP 0.3.1; тестовые расчеты проводились на вычислительной системе “Ломоносов” (IB QDR, NVIDIA X2070, CUDA 4.0).

Анализ результатов показывает, что производительность алгоритма существенным образом зависит от вида матрицы и для матриц разного уровня вложенности нет единого, близкого к оптимальному, формата хранения данных. Это говорит о необходимости поиска альтернативных форматов хранения или альтернативных реализаций алгоритмов умножения матрицы на вектор для имеющихся форматов. Еще одним вариантом может быть выработка каких-либо простых априорных критериев применимости того или иного формата хранения для данной матрицы. Проведя оценку матрицы перед началом расчета, можно выбрать один из форматов её хранения, а значит и соответствующую процедуру умножения.

Поскольку имеет большое значение то, насколько хорошо распараллелен алгоритм в конфигурации вычислительной системы с множеством узлов, приведём графики масштабируемости процедуры выполнения операции умножения матрицы на вектор на вычислительной системе “Ломоносов” (рис. 3). Для этих расчетов использовались тестовые матрицы, сформированные для решения уравнения Пуассона на равномерной декартовой сетке с 27-точечным шаблоном. Операция умножения выполнялась для формата хранения матриц ELLPACK, который наилучшим образом подходит для такого вида матриц. Приведенные графики демонстрируют достаточно хорошие результаты масштабируемости, обеспечивая 20-кратное ускорение для матрицы размером 8 млн. неизвестных на 32 вычислительных узлах. При этом, следует отметить, что на данном этапе для графических ускорителей не были реализованы все перечисленные в разделе 2 варианты оптимизации процесса обмена сообщений между вычислительными процессами, так что приведенные графики масштабируемости могут быть заметно улучшены.

Таблица 1. Параметры матриц из тестового набора.

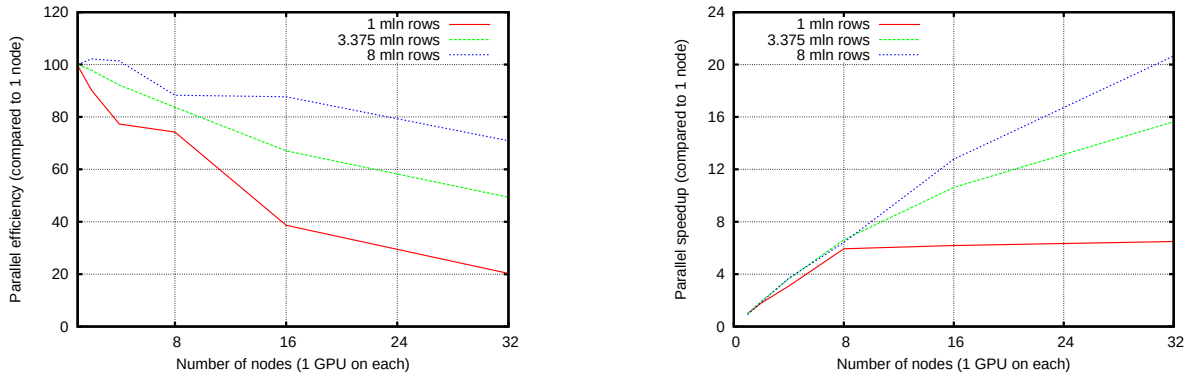
Матрица	**Nrows**	**NNZ**	**NNZ/row, avg**	**NNZ/row, max**
8M_0L	8000000	55760000	7.0	7
8M_1L	4007253	56627171	14.1	27
8M_2L	1752500	36167002	20.6	70
8M_3L	870201	26754707	30.7	178
8M_4L	464875	21050671	45.3	306
8M_5L	239559	15106487	63.1	465
8M_6L	118146	9961480	84.3	877
8M_7L	56793	5962117	105.0	904
8M_8L	26484	3350818	126.5	1058
8M_9L	13258	1965270	148.2	1007

Таблица 2. Сравнение времени выполнения операции SpMV на графическом ускорителе для тестового набора матриц в разных форматах хранения, мсек (GFLOP/s).

Матрица	**COO**	**CSR**	**ELLPACK**	**HYB**
8M_0L	15.06 (7.4)	7.46 (14.9)	4.02 (27.7)	4.02 (27.7)
8M_1L	19.04 (5.9)	8.34 (13.5)	9.24 (12.2)	7.94 (14.2)
8M_2L	16.32 (4.4)	13.32 (5.4)	9.66 (7.4)	7.45 (9.7)
8M_3L	12.26 (4.3)	9.38 (5.7)	11.60 (4.6)	6.69 (7.9)
8M_4L	8.81 (4.7)	7.10 (5.9)	11.66 (3.6)	6.66 (6.3)
8M_5L	5.78 (5.2)	4.95 (6.1)	9.66 (3.1)	5.34 (5.6)
8M_6L	3.71 (5.3)	3.07 (6.4)	7.81 (2.5)	3.81 (5.2)
8M_7L	2.17 (5.4)	1.65 (7.1)	4.51 (2.6)	2.31 (5.1)
8M_8L	1.16 (5.7)	0.83 (8.0)	2.98 (2.2)	1.35 (4.9)
8M_9L	0.68 (5.7)	0.39 (10.0)	1.41 (2.7)	0.78 (4.9)

5. Заключение

На примере метода BiCGStab с алгебраическим многосеточным предобуславливателем разработан двухуровневый алгоритм распараллеливания вычислений. Предложенный алго-



(а) (б)

Рис. 3. Масштабируемость алгоритма умножения матрицы на вектор, выполняемого на узлах с графическими ускорителями (1 ускоритель на узел) для задач разного размера: (а) параллельная эффективность в сравнении с одним узлом, (б) ускорение в сравнении с одним узлом.

ритм реализован в рамках гибридной модели MPI+ShM. Представлены результаты масштабируемости в пределах одного узла, демонстрирующие ускорение, соответствующее ускорению MPI-схемы распараллеливания. Полученные результаты в 7-8 раз превосходят результаты масштабируемости гибридной модели MPI+OpenMP, реализованной в библиотеке hypre.

При распараллеливании ключевого алгоритма умножения матрицы на вектор с использованием графических ускорителей показано, что выбор оптимального формата хранения данных зависит от вида матрицы и существенно влияет на эффективность алгоритма. На выбор формата представления матрицы также влияют соображения по возможности создания эффективных алгоритмов для проведения распределенных расчетов на графических ускорителях. Показана возможность получения приемлемой масштабируемости на многоузловых вычислительных системах для алгоритма умножения матрицы на вектор, выполняемого на графических ускорителях. Дальнейшая работа будет направлена на разработку распределенной реализации итерационных и многосеточных методов на гибридных вычислительных системах и сопряжении этих методов с пакетом OpenFOAM.

Представленные в работе результаты расчетов были получены на вычислительных системах “Ломоносов” суперкомпьютерного комплекса Московского университета и “Зилант” компании ЗАО “Т-Сервисы”.

Литература

1. Swartzreuber P.N. A direct method for the discrete solution of separable elliptic equations // SIAM Journal on Numerical Analysis. 1974. Vol. 11, N. 6. P. 1136–1150.

2. Saad Y. Iterative methods for sparse linear systems, 2-nd edition. SIAM, 2003. 528 p.

3. Trottenberg U., Oosterlee C.W., Schuller A. Multigrid. N.Y.: Academic Press, 2001. 631 p.

4. Krasnopolsky B. The reordered BiCGStab method for distributed memory computer systems // Procedia Computer Science. 2010. Vol. 1. P. 213–218.

5. Bell N., Garland M. Cusp: Generic Parallel Algorithms for Sparse Matrix and Graph Computations. URL: http://cusp-library.googlecode.com (дата обращения: 02.12.2012).

6. CUSPARSE library. URL: https://developer.nvidia.com/cusparse (дата обращения: 02.12.2012).

7. Strzodka R. Accelerated ANSYS Fluent: Algebraic Multigrid on a GPU. Supercomputing, 2012.
URL: http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/RobertStrzodka_Accelerated_ANSYS_Fluent_SC12.pdf (дата обращения: 02.12.2012).

8. Esler K., Natoli V., Samardžić A. Accelerating Reservoir Simulation and Algebraic Multigrid with GPUs. GPU Technology Conference, 2012.
URL: http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/S0140-GTC2012-Reservoir-Simulation-Algebraic.pdf (дата обращения: 02.12.2012).

9. LAMA – Library for Accelerated Math Applications.
URL: http://www.libama.org/overview.html (дата обращения: 02.12.2012).

10. Combest D.P., Day J. Cufflink: a library for linking numerical methods based on CUDA C/C++ with OpenFOAM. URL: http://cufflink-library.googlecode.com (дата обращения: 02.12.2012).

11. Kraus J., Förster M., Brandes T., Soddemann T. Using LAMA for efficient AMG on hybrid clusters // Computer Science - Research and Development. May 2012. P. 1–10.

12. Краснопольский Б.И. Об особенностях решения больших систем линейных алгебраических уравнений на многопроцессорных вычислительных системах различной архитектуры // Вычислительные методы и программирование. 2011. Т. 12, № 1. С. 176–182.

13. Краснопольский Б.И. Исследование эффективности переупорядоченного метода BiCGStab на вычислительных системах СКИФ МГУ «Чебышёв» и «Ломоносов» // Вестник ЮУрГУ. Серия “Математическое моделирование и программирование”. 2011. Т. 7, № 4(221). С. 56–65.

14. PGAS website. URL: http://www.pgas.org/ (дата обращения: 02.12.2012).

15. Bonachea D., Jeong J. GASNet: A Portable High-Performance Communication Layer for Global Address-Space Languages. CS258 Parallel Computer Architecture Project. 2002.

16. Корж. А.А. Результаты масштабирования бенчмарка NPB UA на тысячи ядер суперкомпьютера Blue Gene/P с помощью PGAS-расширения OpenMP // Вычислительные методы и программирование. 2010. Т. 11, № 1. С. 164–174.

17. Hypre: a library of high performance preconditioners.
URL: https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html (дата обращения: 02.12.2012).

18. Hugues M.R. Sparse Matrix Formats Evaluation and Optimization on a GPU // High Performance Computing and Communications (HPCC), 12th IEEE International Conference. 2010. P. 122–129.

19. CUDA C Programming Guide (PG-028290-001_v5.0). NVIDIA Corporation, 2012.

20. Sandres J., Kandrot E. CUDA by example. An introduction to General-Purpose GPU Programming. Addison-Wesley, 2010. 312 p.

21. Bell N., Garland M. Efficient Sparse Matrix-Vector Multiplication on CUDA. NVIDIA Technical Report NVR-2008-004. NVIDIA Corporation, 2008.

22. Baxter S. MGPU Sparse Library. URL: http://www.moderngpu.com/sparse/spmxv.html (дата обращения 02.12.2012).