АВТОМАТИЧЕСКОЕ РАСПАРАЛЛЕЛИВАНИЕ ПРОГРАММ В ТЕХНОЛОГИИ ГРАФОСИМВОЛИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Д.П. Крамаренко, В.В. Жидченко

Самарский государственный аэрокосмический университет им. С.П. Королёва

Визуальное программирование - способ создания программы для ЭВМ путём манипулирования графическими объектами вместо написания её текста.

Технология графосимволического программирования (ГСП), созданная на кафедре программных систем СГАУ, является одним из способов наглядного представления алгоритмов программы в виде графа управления. Данная технология предоставляет пользователям широкие возможности анализа параллельной структуры алгоритма, корректности применяемых механизмов синхронизации вычислений, поиска тупиковых ситуаций, оценки эффективности алгоритма и т.д.

Данная технология программирования исповедует два основополагающих принципа:

- визуальную, графическую форму представления алгоритмов программ и других компонент их спецификации;
- принцип структурированного процедурного программирования.

В качестве методологической основы для представления алгоритмов в работе используется модель объекта с дискретными состояниями. Основу такой модели составляет предположение, что для любого объекта программирования тем или иным способом можно выделить конечное число состояний, в которых он может пребывать в каждый момент времени. Тогда развитие вычислительного процесса можно ассоциировать с переходами объекта из одного состояния в другое.

Существует множество графических моделей описания алгоритмов параллельных вычислений. В области описания вычислительных алгоритмов наиболее удобной представляется форма, близкая описанию блок-схемам, которая реализована в параллельной версии технологии ГСП.

На данный момент имеется большой фонд последовательных граф-программ. Так как в среде ГСП реализована возможность параллельного исполнения кода, то преобразование последовательных программ в параллельные представляет большой интерес. Однако выполнение подобного преобразования граф-программ вручную является трудоемкой задачей. Поэтому в данной работе ставится задача разработать алгоритм для автоматического распараллеливания готовых последовательных граф-программ, что позволит существенно облегчить этот процесс.

Для решения данной задачи производится поиск информационной зависимости между вершинами в граф-программе, организация необходимых структур данных, получение карты всех маршрутов граф-программы и на основе проведенного анализа происходит изменение структуры графа. На данный момент в тестовую версию среды ГСП добавлен программный модуль, исполняющий алгоритм автоматического преобразования последовательной граф-программы в параллельную. Так же в разработке находится алгоритм, позволяющий на основе анализа зависимостей и структуры графа распараллелить выполнение циклов.

ОПТИМИЗАЦИЯ ПЕРЕСЫЛОК ДАННЫХ ПО MPI ПРИ РЕШЕНИИ СИСТЕМЫ УРАВНЕНИЙ НАВЬЕ-СТОКСА НА GPU-КЛАСТЕРЕ

М.А. Кривов[1,3], М.Н. Притула[1], А.А. Гаврилов[2], А.А. Дектерёв[2]

[1]ООО «ТТГ Лабс»

[2]Институт теплофизики им. С.С. Кутателадзе СО РАН

[3]Московский госуниверситет им. М.В. Ломоносова

В работе описан опыт оптимизации пересылок данных между узлами GPU-кластера при решении системы уравнений Навье-Стокса. Приведены три схемы обмена граничными элементами между смежными доменами и представлены результаты их тестирования на GPU-кластере из 12 узлов.

Введение

При использовании графических ускорителей (GPU) для решения промышленных задач аэрогидродинамики крайне важным свойством выбранного решателя является поддержка вычислительных систем с множеством GPU. Причём причина кроется отнюдь не в необходимости многократного сокращения времени расчётов (что иногда тоже является важным), а в ограниченности размера памяти отдельного ускорителя, размер которой обычно не превышает нескольких гигабайт. Как следствие, для проведения расчётов на сетке с требуемым уровнем детализации необходим вычислительный кластер как минимум с несколькими, а то и несколькими десятками графических ускорителей, суммарный размер памяти которых окажется достаточным для хранения сетки и всех сопутствующих массивов.

Для обеспечения данного свойства от разработчика может потребоваться осуществление дополнительной оптимизации решателя. Действительно, если в случае классических гомогенных кластеров при выполнения одной MPI-операции происходит лишь копирование данных в рамках одного класса памяти из некоего массива в MPI-буфер, то при использовании GPU-системы этому будет предшествовать дополнительное копирование данных по шине PCI Express, которое является достаточно медленной операцией. Более того, так как при фиксированном размере домена один GPU оказывается в разы производительнее центрального процессора, то и вклад MPI-операций в суммарное время работы GPU-версии пакета становится намного больше (в процентном соотношении). Другими словами, при переходе на GPU-кластер время обмена между узлами возрастает, и с большой вероятностью подобные операции становятся узким местом.

В данной статье описывается опыт осуществления подобных оптимизаций в GPU-версии пакета SigmaFlow [1], разрабатываемого институтом теплофизики им. С.С. Кутателадзе СО РАН.

Схема обмена граничными элементами сетки

В рассматриваемом пакете озвученная оптимизация проводилась для решения системы уравнений Навье-Стокса, в котором при обмене данными больше всего времени тратилось на пересылку значений граничных элементов сетки. В данной реализации исходная сетка с помощью программы METIS [2] разбивалась на домены, для каждой

пары которых строились два массива индексов вершин, являющихся граничными элементами, что схематично показано на следующей иллюстрации:

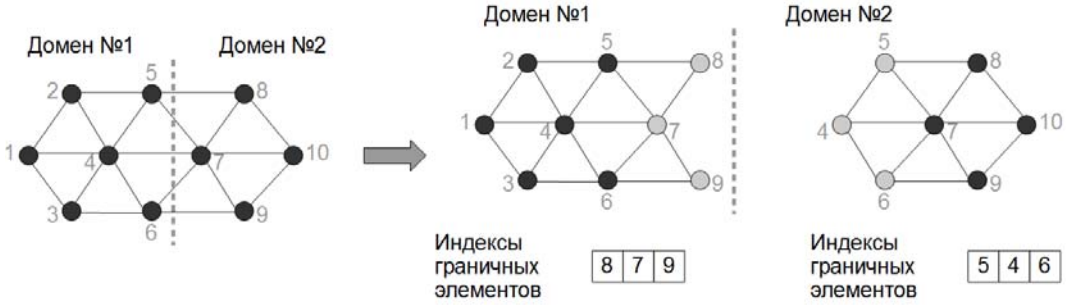

Рис. 1. Схема разбиения сетки на домены

В этом примере узел, обрабатывающий домен №1, на каждой итерации будет пересылать узлу, обрабатывающему домен №2, массив со значениями искомой величины для вершин {5, 4, 6}, в то время как обратно получать значения для вершин {8, 7, 9}. В случае, если количество доменов больше двух, то подобные пересылки осуществляются в рамках каждой пары.

В оптимизируемом пакете все расчёты проводятся на неструктурированных сетках, поэтому для хранения данных в большинстве случаев использовался формат, в котором сетка задаётся двумя массивами, описывающими рёбра соответствующего графа. Так, в первом массиве хранятся индексы первой вершины, соединяемой соответствующим ребром, во втором – второй. Значения же всех искомых величин (например, трёх компонент скорости, плотности и давления) задаются в виде дополнительных массивов, индексы элементов которых совпадают с индексами вершин, как показано на следующей иллюстрации:

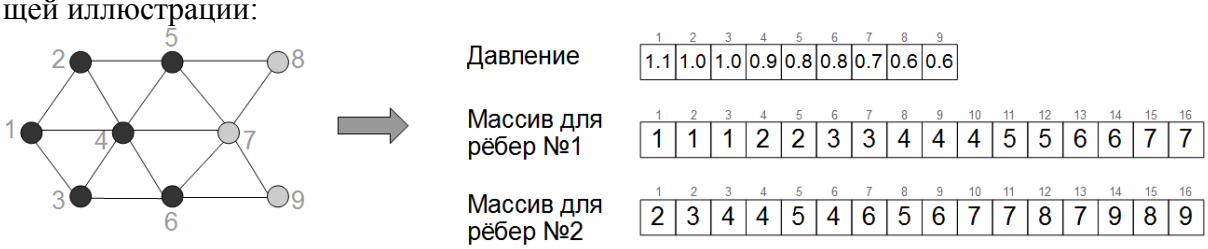

Рис. 2. Формат хранения сеток и заданных на них величин

В таком случае схема обмена граничными элементами сводится к выполнению трёх этапов: 1) копирование всех граничных элементов в рамках каждой пары доменов в общий MPI-буфер, 2) обмен нужными значениями по схеме «каждый узел с каждым», 3) копирование обновлённых граничных элементов в исходные массивы. Стоит отметить, что данное описание является достаточно схематичным и не совсем точно соответствует реализованной в пакете более сложной схеме асинхронного обмена, но для дальнейшего изложения решаемых проблем его вполне достаточно.

При переходе к вычислениям на GPU основной проблемой становится необходимость осуществления операций копирования данных по шине PCI Express, так как они являются крайне медленными. В рамках рассматриваемой схемы это означает, что массив с элементами искомой величины (в примере выше – давлением), расположен только в памяти GPU, и некоторые из его элементов необходимо копировать в MPI-буфер, созданный в памяти центрального процессора. В данной работе было реализовано три варианта формирования данного MPI-буфера, которые описаны в следующем разделе. Также стоит сказать, что на момент написания статьи отдельно не были рассмотрены аппаратные решения типа технологии NVidia GPUDirect [3], применение которых является дальнейшим развитием данной работы.

Варианты формирования массива граничных элементов

Первым и самым простым (и в тоже время самым неэффективным) вариантом является копирование массива с искомой величиной целиком в память центрального процессора, осуществление обмена граничными элементами, после чего копирование обновлённого массива обратно в память GPU. Чтобы потеря производительности не была крайне большой, копирование массива осуществлялось в pinned-память, полученную с помощью функции cudaHostRegister() и обладающую лучшими характеристиками за счёт отказа от страничной адресации. Данный вариант схематично представлен на следующей иллюстрации:

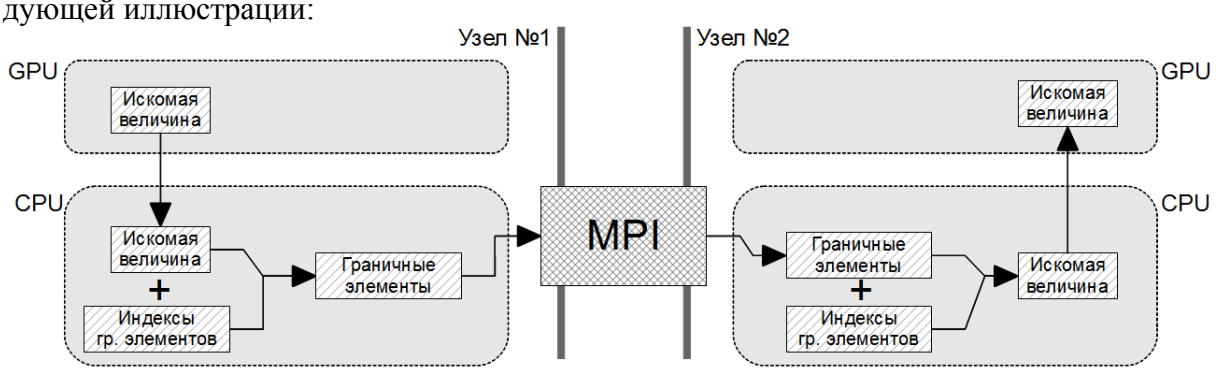

Рис. 3. Первая (исходная) схема пересылки значений граничных элементов

Количество подобных операций равно (N-1)·N·M, где N – количество MPI-процессов, а M – число синхронизируемых массивов, которое обычно равно либо одному (скалярная величина), либо трём (векторная величина), либо девяти (градиенты компонент векторной величины). При этом за счёт укрупнения запросов реально одним узлом производится (N-1) ·2 асинхронных MPI-операций, и всего M·2 операций копирования по шине PCI Express.

В следующей реализации формирование массива граничных элементов производилось силами GPU, после чего в память центрального процессора копировался лишь требуемый массив граничных элементов намного меньшего размера, в результате чего многократно уменьшался объём пересылаемых данных, однако и увеличивалось число операций копирования – с M·2 до M·(N – 1)·2.

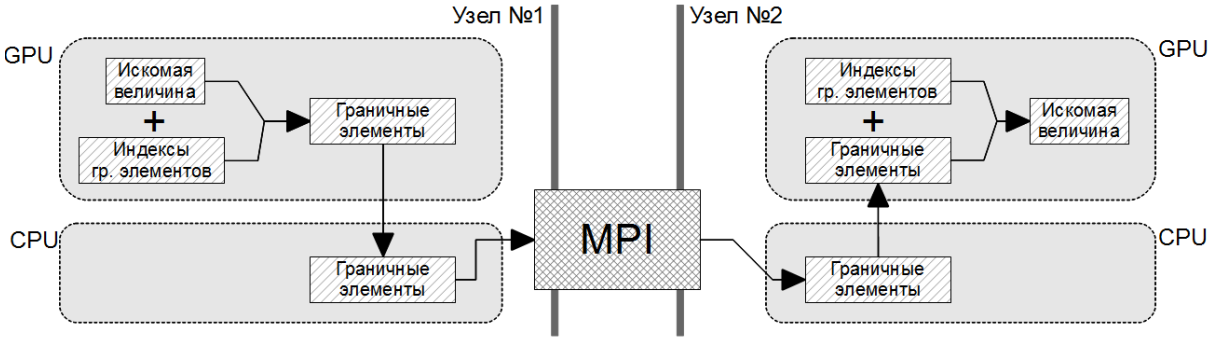

Рис. 4. Вторая схема пересылки граничных элементов

Недостатком данной реализации является как раз излишне большое количество вызовов CUDA-функций, что при работе с сетками небольшого размера может заметно замедлить проведение расчётов из-за задержки шины, которая на практике может доходить до долей миллисекунды. Поэтому также была разработана третья реализация, в которой формирование MPI-буфера для обмена сразу между несколькими узлами проводится исключительно силами GPU, после чего полученный буфер лишь один раз копируется по шине PCI Express, как это показано на следующей иллюстрации:

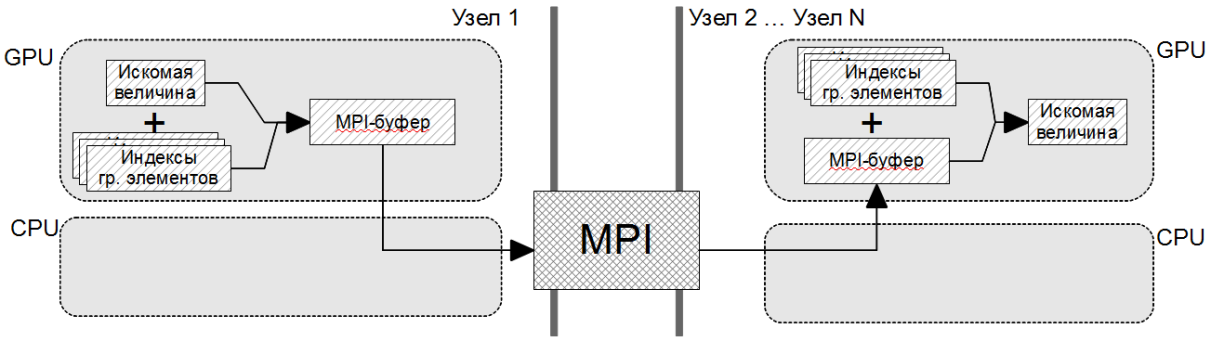

Рис. 5. Третья схема пересылки граничных элементов

Когда количество узлов равно двум, вторая и третья реализация оказываются практически идентичными. При этом с ростом количества узлов во втором варианте также растёт количество CUDA-операций, в то время как в третьем варианте оно остаётся инвариантным и равным двум операциям на узел. Минусом же последнего варианта является синхронность – в нём не удаётся инициировать обмен до тех пор, пока весь MPI-буфер не будет готов.

Для упрощения, в дальнейшем первая реализация пересылок будет обозначаться как «original», так как именно она была реализована в пакете изначально, вторая - «bandwidth-optimized», так как по оценкам она должна была оказаться предпочтительней для больших сеток, третья - «latency-optimized», так как в ней делается акцент на минимизацию задержек. Вышеперечисленные отличия всех трёх версий приведены в следующей таблице 1.

Таблица 1. Сравнение трех предложенных реализаций пересылок данных

Реализация	**Объём пересылаемых данных по шине PCI-E**	**Количество операций копирования по шине PCI-E**	**Асинхронность обмена по MPI**
Original	Большой (M*2*S_1)	M * 2	Да
Bandwidth-optimized	Небольшой (порядка M*2*S_2)	M * (N-1) * 2	Да
Latency-optimized	Небольшой (порядка M*2*S_2)	2	Нет

Здесь S_1 – размер массива с синхронизируемой величиной, S_2 – размер массива со всеми граничными элементами (для всех смежных доменов), N – количество доменов (количество GPU), M – число синхронизируемых скалярных величин.

Результаты тестирования

Тестирование проводилось на кластере из 12 узлов, каждый из которых оснащён двумя шестиядерными процессорами и тремя ускорителями NVidia Tesla C2050. При этом каждый домен сетки в зависимости от теста обрабатывался либо одним ядром центрального процессора, либо одним ускорителем, а на каждый узел создавался ровно один MPI-процесс, которому поручалась обработка соответствующего домена. Таким образом стоит признать, что кластер использовался лишь частично, однако выбранная схема позволяет более качественно оценить именно время пересылок по межсетевому соединению InfiniBand.

Перед тем, как переходить к тестированию всех трёх реализаций, стоит сказать несколько слов об объёме пересылаемых данных по шине PCI Express. Объем напрямую зависит как от типа сетки, так и числа доменов, на которое она разбивается. Ниже приведён график, показывающий, насколько сократился объём пересылаемых данных при синхронизации одного скалярного поля, заданного на сетке из 500 тысяч узлов.

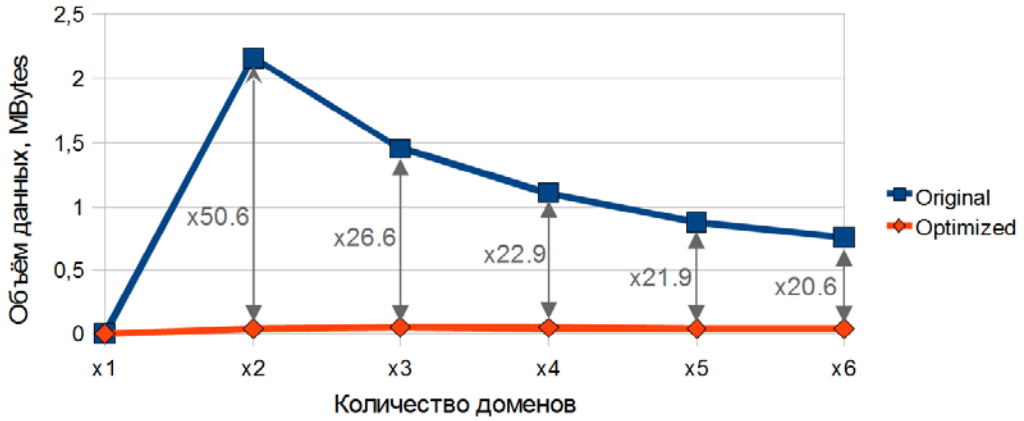

Рис. 6. Объём пересылаемых данных по шине PCI-E в зависимости от реализации

Как видно, проведённые оптимизации позволили сократить объём пересылаемых данных в 20-50 раз и, как показано на рис. 7, также существенно ускорить сами расчёты.

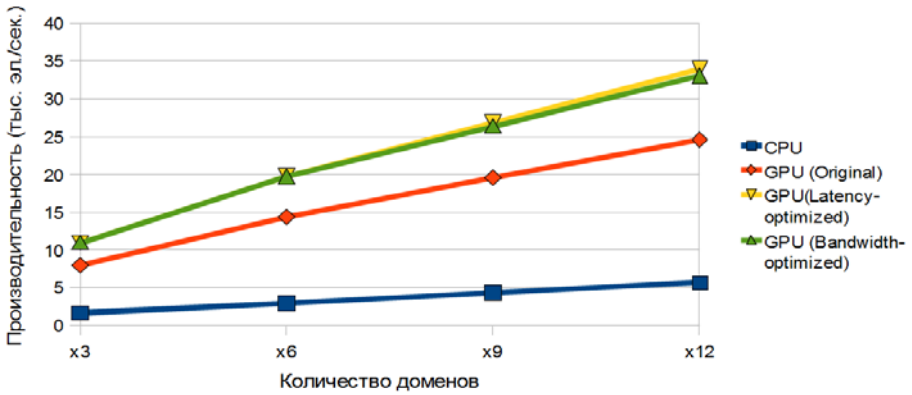

Рис. 7. Скорость проведения расчётов на сетке из 2 млн узлов

Поскольку на рис. 7 отличия двух реализаций друг от друга практически незаметны, то отдельный интерес представляет следующий график, где приведено только время, затрачиваемое на пересылку данных.

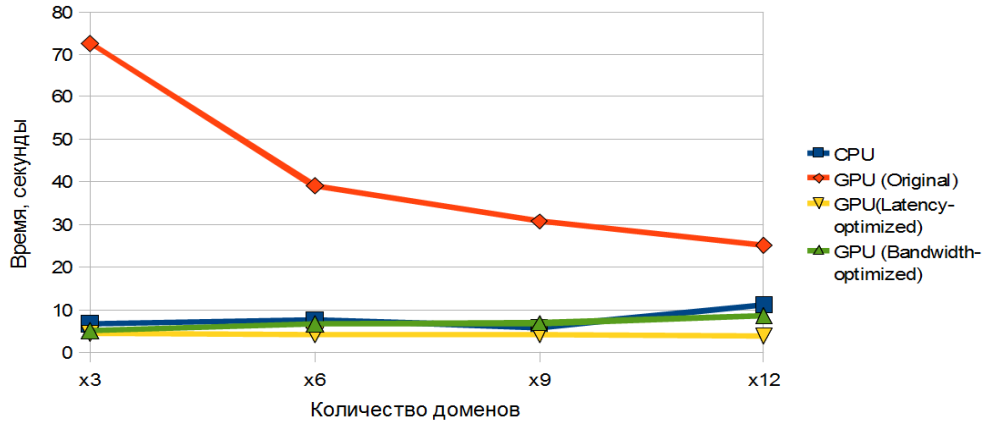

Рис. 8. Время, затрачиваемое на пересылки во всех реализациях, 2 млн узлов

Таким образом оптимальной реализацией оказалась версия, в которой MPI-буфер целиком формируется силами GPU. Случай, когда граничные элементы по частям копируются в память центрального процессора, оказался почти в два раза медленнее, однако по сравнению с общим временем работы выигрыш практически незаметен. Наконец, изначальная версия, в которой весь массив с искомой величиной целиком копиру-

ется в память центрального процессора, как и ожидалось, оказалась крайне медленной. Также стоит отметить, что в результате проведённых оптимизаций время на MPI-пересылки в GPU-версии оказалось даже меньше, чем в исходной реализации для центрального процессора. Это объясняется тем, что формирование массива с граничными элементами силами GPU происходит намного быстрее, что полностью компенсирует затраты на копирование данных по шине PCI Express.

Стоит отдельно сравнить вклад операций по пересылке граничных элементов в общее время расчётов, что проиллюстрировано на рис. 9.

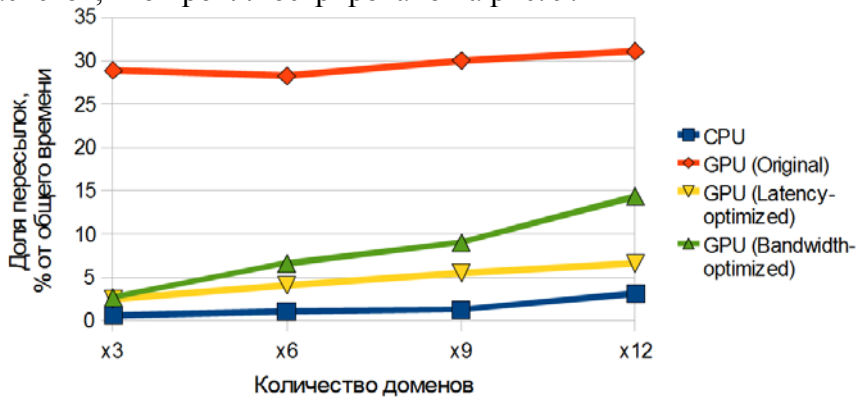

Рис. 9. Доля операций по пересылке данных в общем времени расчётов, 2 млн узлов

Как видно, благодаря схеме с формированием MPI-буфера на GPU удалось сократить время пересылок в процентном соотношении ровно в 5 раз - с 30% до 6%, в результате чего рассматриваемые операции перестали быть узким местом.

Заключение

Три описанные в работе схемы пересылок граничных элементов между смежными доменами являются достаточно очевидными, и в тоже время применимыми для решения разнообразных уравнений математической физики. Однако к вопросу выбора подходящей схемы стоит подходить основательно, так как скорость расчётов с использованием той или иной реализации может существенно изменяться как в зависимости от типа решаемой задачи, так и выбранной аппаратной платформы.

Проведённое в рамках работы тестирование показало, что при решении векторного уравнения на GPU-кластере имеет смысл формировать MPI-буфер исключительно силами GPU. В частности, данная оптимизация позволила сократить время выполнения операций обмена граничными элементами по сравнению с исходной реализацией в 7.5 раз (с 30 до 4 секунд на сетке из 2 млн ячеек и кластере из 12 узлов), или в 5 раз в процентном отношении к итоговому времени расчётов (с 30% до 6%), благодаря чему данные операции перестали быть узким местом выбранного модуля пакета.

Литература

1. Гаврилов А.А., Кривов М.А., Гризан С.А., Дектерёв А.А. GPU-версия CFD пакета SigmaFlow: портирование и оптимизация с использованием инструментария TTG Apptimizer // Параллельные вычислительные технологии (ПаВТ'2013). Челябинск: Издательский центр ЮУрГУ, 2013. С. 106-115.
2. Karypis G., Kumar V. A fast and high quality multilevel scheme for partitioning irregular graphs // SIAM Journal on Scientific Computing. 1999. Vol. 20, No. 1. P. 359 – 392.
3. Электронный ресурс https://developer.nvidia.com/gpudirect.