На пути к освоению гетерогенных супервычислений в газовой динамике[1]

А.В. Горобец, С.А. Суков, П.Б. Богданов

Аннотация. Работа посвящена эффективному использованию современных гетерогенных вычислительных систем на основе массивно-параллельных ускорителей, применяющихся в качестве математических сопроцессоров. В статье демонстрируется адаптация алгоритмических операций на примере конечно-объемных методов для моделирования задач газовой динамики на неструктурированных сетках. Рассматриваются различные типы устройств, в том числе, графические процессоры NVIDIA и AMD, ускорители Intel Xeon Phi.

Ключевые слова: газовая динамика, неструктурированные сетки, MPI, OpenMP, OpenCL, GPU.

Введение

Появление вычислительных систем, построенных по кластерному принципу, можно без преувеличения назвать одним из наиболее значимых событий в истории математического моделирования. Наращивание вычислительной мощности путем объединения коммуникационной средой вычислительных модулей и появление соответствующего программного обеспечения, в первую очередь MPI (Message Passing Interface) [1], позволили расширить круг рассматриваемых проблем и повысить качество результатов численных экспериментов. Следующим шагом в развитии суперкомпьютеров стало появление многоядерных процессоров. Когда процессоры с 6-ю, 8-ю и даже с 16-ю ядрами пришли на смену двух- и четырех ядерным процессорам, потребовалась существенная перестройка распараллеливания и переход на более сложную двухуровневую модель, сочетающую модели с распределенной и общей памятью.

Производительность суперкомпьютеров продолжала расти, и число процессорных ядер в наиболее крупных системах уже перевалило за миллион. Тем не менее стало очевидно, что масштабирование системы с многоядерными процессорами до эксафлопсного уровня производительности на практике вряд ли неосуществимо. Дальнейшая эволюция кластерных систем пошла по пути перехода к гетерогенной архитектуре. Мощность систем такой архитектуры обусловлена применением массивно-параллельных вычислительных ускорителей в дополнение к центральным процессорам CPU (Central Processing Unit). Сейчас в качестве таких ускорителей широко используются графические процессоры GPU (Graphics Processing Unit) производства NVIDIA и AMD. В качестве альтернативы Intel выпустило ускорители Xeon Phi существенно-многоядерной архитектуры MIC (Many Integrated Core). Главным плюсом ускорителей является лучшее по сравнению с центральными процессорами соотношение производительности на единицу потребляемой энергии. При этом система, естественно, получается еще и более компактной.

Однако высокий потенциал гетерогенных вычислительных комплексов омрачается существенно возросшей сложностью программирования и адаптации численных алгоритмов с учетом особенностей новой архитектуры массивно-

[1] Работа выполнена при финансовой поддержке РФФИ, проект 12-01-33022

параллельного вычислителя и системы в целом. А это, очевидно, существенные затраты времени и ресурсов, которые в итоге должны быть оправданы реальной практической выгодой.

Цель данной работы заключается в демонстрации подходов к адаптации алгоритмов и оптимизации вычислений, а также в теоретической и практической оценке преимуществ и недостатков использования гетерогенных вычислительных систем по сравнению с обычной чисто многоядерной архитектурой на примере одного из наиболее сложных классов алгоритмов с точки зрения вычислений на ускорителях. Рассматриваются параллельные конечно-объемные алгоритмы на неструктурированных сетках, базовое MPI распараллеливание которых позволяет задействовать тысячи вычислительных модулей суперкомпьютера для моделирования задач газовой динамики. Сложность в достижении высокой эффективности вычислений, с одной стороны, состоит в крайне низкой удельной вычислительной стоимости на единицу данных. Вследствие этого, основным фактором, ограничивающим производительность, становится пропускная способность памяти. С другой стороны, нерегулярный шаблон доступа к памяти, обусловленный применением неструктурированных сеток, делает латентность доступа к памяти другим существенным ограничителем.

Проблемы возникают уже на самом базовом уровне – уровне представления данных для пространственной дискретизации. Было разработано много подходов и форматов для хранения топологии сеточных связей, разреженных неструктурированных матриц. Примеры адаптации форматов хранения разреженных матриц, возникающих при расчетах на неструктурированных сетках, представлены в [2, 3], обзор различных форматов и показатели производительности для различных типов матриц представлены в [4]. На примере простой, по сути, операции в этих статьях демонстрируются сложные подходы, в основном связанные с переупорядочиванием данных и оптимизацией доступа к памяти.

Такое внимание к организации структуры данных не случайно. Так, время решения одной и той же задачи для различных вариантов индексации сеточных элементов геометрической модели может различаться в разы. В то же время для логически сложных алгоритмов трудно доказать оптимальность выбранного подхода к хранению и доступу к данным. Тем не менее, комплексная информация о производительности вычислений как отдельных модулей приложения, так и всего приложения в целом, вместе с соответствующими показателями соотношения объемов вычислений на единицу данных, позволяет судить об эффективности реализованных алгоритмов. К сожалению, в отечественных публикациях на тему параллельных вычислений подобные показатели приводятся достаточно редко. В то же время, часто встречающийся показатель ускорения относительно CPU сам по себе неинформативен.

Кроме того, одна из целей данной работы заключается в сравнении двух архитектур суперЭВМ. И здесь целесообразно дополнительно ввести некоторый критерий оценки оправданности переноса вычислений на гетерогенные системы. Например, установить минимальный уровень ускорения вычислений на GPU, ниже которого реализация алгоритма для гетерогенной системы становится практически нецелесообразной. Поскольку главное преимущество ускорителя заключается в лучшем соотношении производительности на единицу потребляемой энергии, то за минимальный уровень ускорения примем соотношение энергопотребления ускорителя и CPU.

1. Архитектурные особенности CPU и ускорителей

Прежде чем перейти к рассмотрению проблем разработки алгоритмов для CPU и GPU, необходимо в общем виде обозначить особенности архитектуры этих устройств. В первую очередь рассмотрим соотношение производительности и пропускной способности памяти. Например, для процессора Intel Xeon Processor X5670 2.93 GHz теоретическая пиковая производительность составляет примерно 72 GFLOPS, максимальная пропускная способность памяти – 32 GB/s. Отношение первого ко второму составляет 2.25. Для арифметики с числами с плавающей точкой двойной точности

(double), размер которых составляет 8 байт, это соотношение следует умножить на 8. Таким образом, получим соотношение 18 операций с плавающей точкой на один аргумент из памяти.

Если взять для примера GPU NVIDIA C2050, то соотношение 515 GFLOPS / 144 GB/s будет примерно 3.6, то есть порядка 30 операций с плавающей точкой на один аргумент из памяти. Аналогичное значение будет и для GPU AMD Radeon 7970 с пиком 947 GFLOPS и пропускной способностью 264 GB/s. Получается, что по сравнению с процессором на ускорителе ограничения по пропускной способности памяти приобретают заметно больший вес.

В то же время, для многих приложений математического моделирования, связанных с подробной пространственной дискретизацией, и в особенности для задач газовой динамики, характерна достаточно низкая удельная вычислительная стоимость на единицу данных. Так, для рассматриваемых алгоритмов характерное среднее число арифметических операций на один аргумент из памяти варьируется от 2-х до 6-ти. Это означает, что достижение даже одной пятой от пиковой производительности устройства на практике едва ли осуществимо для такого класса приложений.

Поскольку работа с памятью имеет первостепенное значение, рассмотрим в общем виде отличия между процессором и ускорителем с этой точки зрения. Как для GPU производства AMD и NVIDIA, так и для ускорителей Intel Xeon Phi, характерны общие особенности. Во-первых, ускорители, по пиковой производительности на порядок превосходящие CPU, имеют весьма ограниченный объем оперативной памяти, обычно не превышающий 8 GB, что на порядок меньше доступной CPU памяти.

Во-вторых, память ускорителей имеет в несколько раз большую латентность доступа по сравнению с процессором. Компенсируется этот недостаток механизмом сокрытия латентности доступа к памяти, когда на каждый вычислительный элемент приходится сразу несколько заданий и пока одни задания ожидают поступления данных, другие выполняются. Для повышения эффективности доступа к памяти на GPU необходимо, чтобы число нитей превосходило число вычислительных ядер хотя бы на порядок. На Intel Xeon Phi сокрытию латентности помогает работающая по схожему принципу аппаратная поддержка четырех потоков на одном ядре. Следует отметить, что латентность доступа особенно влияет на производительность в случае применения неструктурированных сеток, для которых неизбежен разрозненный доступ к памяти и кэш промахи.

2. Особенности программирования ускорителей и средства разработки

В настоящее время существует выбор из нескольких средств разработки. Компанией NVIDIA для одноименных графических процессоров предлагается технология разработки приложений CUDA [5]. Для ускорителей Intel Xeon Phi может применяться та же технология параллельного программирования, что и для центральных процессоров. Это может быть, например, стандарт OpenMP (Open Multiprocessing) [6]. В то же время, единым открытым вычислительным стандартом для работы с различными типами ускорителей является OpenCL (Open Computing Language) [7]. Он поддерживается основными производителями компьютерного оборудования, включая Intel, AMD, IBM, NVIDIA. В дополнение, независимо от выбора средства разработки для ускорителей, на верхнем уровне алгоритма должно быть реализовано распараллеливание с распределенной памятью, для того чтобы задействовать множество вычислительных модулей суперкомпьютера. В области вычислительной газовой динамики примеры подобных алгоритмов, в которых сочетаются, в частности, MPI и CUDA, представлены в [8-10], правда для более простых классов численных методов, чем рассматривается в данной статье. В работах [8, 9] – это методы на структурированных сетках, что, естественно, упрощает структуру данных и доступ к памяти, а в [10] – метод решёточных уравнений Больцмана, существенно более простой для реализации на GPU (за счет присущей таким методам "явности" и простоты структуры данных).

Технология разработки приложений на OpenCL для массивно-параллельных вычислительных устройств во многом схожа с CUDA и

базируется на следующих принципах. Вычислительные задания выполняются множеством легковесных процессов (нитей). При формальной схожести такого подхода с моделями программирования для CPU, между ними существует концептуальное различие: если время жизни запускаемого на CPU параллельного процесса совпадает со временем работы соответствующего программного модуля, то время жизни нити на ускорителе равно времени обработки этой нитью некоторого элементарного задания, совокупность которых формирует общий объем вычислений для рассматриваемой задачи (так называемый потоковый параллелизм).

Внутри множества выполняющихся нитей существует иерархия: нити группируются в локальные группы (или в терминологии CUDA – блоки). Внутри таких подмножеств нитей существует возможность обмена данными через разделяемую память и синхронизации при помощи специальных функций. Поэтому все нити блока выполняются на одном мультипроцессоре ускорителя. Распределение же самих блоков между мультипроцессорами происходит динамически по мере освобождения ресурсов, следовательно, нельзя прогнозировать очередность завершения выполнения блоков.

Мультипроцессоры GPU ускорителя управляют выполнением множества нитей одного или нескольких активных блоков, обеспечивая тем самым оптимальную загрузку своих ресурсов, но непосредственно вычисления и обращения к памяти происходят группами фиксированного размера (для NVIDIA C2050 это 32 нити), называемыми варпами (warp). Нити варпа выполняют одновременно одну и ту же инструкцию, но с различными данными, что соответствует модели параллелизма SIMD (Single Instruction Multiple Data). Если пути выполнения для нитей варпа расходятся по какому-либо условному переходу, то инициируется последовательное выполнение каждого из путей ветвления с отключением нитей, для которых текущее условие ложно. Поэтому с точки зрения оптимальной производительности важно не столько отсутствие условных переходов в алгоритме вообще, сколько минимизация вероятности расхождения путей выполнения для нитей одного варпа. Также очевидно, что для достижения максимальной эффективности вычислений число нитей в блоке должно быть кратным размеру варпа.

Для оптимизации вычислений на ускорителе необходимо обеспечивать высокий коэффициент загрузки устройства, поскольку с увеличением загрузки мультипроцессоров минимизируется вероятность простаивания его ресурсов (затраты времени на чтение и запись данных могут скрываться за вычислениями и наоборот). Но единственный, по сути, способ повышения коэффициента загрузки – снижение ресурсоемкости алгоритма: необходимо минимизировать число регистров и объем разделяемой памяти, которые требуются для запуска одного блока нитей. И здесь есть еще одно принципиальное ограничение по ресурсоемкости. Регистровая память мультипроцессора является разделяемым ресурсом, но максимальное число регистров, которое может использовать одна нить, ограничивается, например, в случае NVIDIA C2050 63 единицами. Если компилятору не удается оптимизировать код до уровня использования 63 регистров, недостающие регистры эмулируются в разделяемой или глобальной памяти устройства, что, как показывает практика, приводит к снижению быстродействия до неприемлемо низкого уровня. В таком случае решением проблемы может быть преобразование логики исходного алгоритма путем его разделения на несколько более простых и менее ресурсоемких подзадач. А это, очевидно, будет связано с большими трудозатратами, необходимостью выделения дополнительных массивов в глобальной памяти устройства, потерей модифицируемости и прозрачности алгоритма.

Из существенных отличий между CUDA и OpenCL стоит отметить возможность компиляции программных ядер OpenCL во время выполнения CPU программы, что позволяет широко применять директивы препроцессора. Теперь, подытоживая сказанное в данном разделе, сформулируем основные методы оптимизации алгоритмов и программного обеспечения для массивно-параллельных ускорителей:

- поиск минимального зерна параллелизма, т.е. представление исходного логически сложного алгоритма в виде наиболее простых с алгоритмической точки зрения подзадач;

- минимизация ресурсоемкости операций для исключения эмуляции регистров (scratching);
- оптимизация доступа к данным, расположенным в глобальной памяти устройства, учет выравнивания (alignment) и достижение слияния доступа (coalescing);
- минимизация числа операций деления (разница в скорости между умножением и делением намного больше, чем на CPU);
- замена скалярных переменных, известных до выполнения ядра, на константы, развертка циклов, замена ветвлений на директивы препроцессора.

3. Численный алгоритм моделирования задач газовой динамики

3.1. Математическая модель и численный метод

В данном разделе приводится формулировка численного алгоритма, на примере которого будут исследоваться проблемы достижения максимальной эффективности вычислений на CPU и GPU. Рассматривается проблема численного моделирования сжимаемых течений, описываемых системой уравнений Эйлера

$$\frac{\partial \boldsymbol{Q}}{\partial t}+\frac{\partial \boldsymbol{F}_1(\boldsymbol{Q})}{\partial x}+\frac{\partial \boldsymbol{F}_2(\boldsymbol{Q})}{\partial y}+\frac{\partial \boldsymbol{F}_3(\boldsymbol{Q})}{\partial z}=0 \quad (1)$$

где $\boldsymbol{Q}=\left(\rho,\rho u,\rho v,\rho w,E\right)^T$ – вектор консервативных газодинамических переменных. Конвективные потоки $\boldsymbol{F_1(Q)}$, $\boldsymbol{F_2(Q)}$ и $\boldsymbol{F_3(Q)}$ определяются следующим образом:

$$\boldsymbol{F}_1(\boldsymbol{Q})=\begin{pmatrix}\rho u\\ \rho u^2+p\\ \rho uv\\ \rho uw\\ u(E+p)\end{pmatrix},\quad \boldsymbol{F}_2(\boldsymbol{Q})=\begin{pmatrix}\rho v\\ \rho uv\\ \rho v^2+p\\ \rho vw\\ v(E+p)\end{pmatrix},\quad \boldsymbol{F}_3(\boldsymbol{Q})=\begin{pmatrix}\rho w\\ \rho uw\\ \rho vw\\ \rho w^2+p\\ w(E+p)\end{pmatrix}. \quad (2)$$

В формуле (2) введены обозначения:

u = (*u,v,w*) – декартовы компоненты скорости;

ρ, p – плотность и давление;

$E = \rho(u^2+v^2+w^2)/2 + \rho\varepsilon$ – полная энергия, ε – внутренняя энергия.

Система уравнений (1), (2) замыкается уравнением состояния совершенного газа $p = \rho\varepsilon(\gamma - 1)$, где γ есть показатель адиабаты.

В соответствии с методом конечного объема расчетная область задачи покрывается неструктурированной сеткой и разбивается на расчетные ячейки (контрольные объемы). В ячейках задаются среднеинтегральные значения консервативных газодинамических переменных:

$$f_{ID}=\frac{1}{V_{ID}}\int_{V_{ID}} f\left(x,y,z\right)dV,$$

где f_{ID} – значение одной из пяти переменных: ρ_{ID}, ρu_{ID}, ρv_{ID}, ρw_{ID}, или E_{ID}, индекс *ID* обозначает номер ячейки. При использовании явной схемы интегрирования по времени с первым порядком точности значения газодинамических переменных на n+1 временном слое $\boldsymbol{Q}_{ID}^{n+1}$ вычисляются по формуле

$$\boldsymbol{Q}_{ID}^{n+1}=\boldsymbol{Q}_{ID}^{n}+\frac{\Phi_{ID}\tau}{V_{ID}},$$

где τ – текущий шаг интегрирования по времени. Конвективный поток в ячейке Φ_{ID} соответствует сумме конвективных потоков через грани этой ячейки

$$\Phi_{ID}=\sum_{k=0}^{NS_{ID}-1}\Phi S_{ID/I_k^{ID}}.$$

В свою очередь, конвективный поток $\Phi S_{ID/I_k^{ID}}$ через грань ID/I_k^{ID} вычисляется с использованием схемы Роу (F_{ROE}) [11] и зависит от геометрических параметров грани и значений консервативных газодинамических переменных «слева» ($\boldsymbol{QL}_{ID/I_k^{ID}}$) и «справа» ($\boldsymbol{QR}_{ID/I_k^{ID}}$)) от центра ее масс

$$\Phi S_{ID/I_k^{ID}}=F_{ROE}\left(QL_{ID/I_k^{ID}},QR_{ID/I_k^{ID}},\overrightarrow{n_{ID/I_k^{ID}}}\right). \quad (3)$$

Если считать, что значения переменных не меняются внутри объема ячейки, то тогда верны равенства $\boldsymbol{QL}_{ID/I_k^{ID}}=\boldsymbol{Q}_{ID}$ и $\boldsymbol{QR}_{ID/I_k^{ID}}=\boldsymbol{Q}_{I_k^{ID}}$, и формула (3) принимает вид

$$\Phi S_{ID/I_k^{ID}}=F_{ROE}\left(\boldsymbol{Q}_{ID},\boldsymbol{Q}_{I_k^{ID}},\overrightarrow{n_{ID/I_k^{ID}}}\right),$$

что соответствует схеме первого порядка аппроксимации по пространству. Повышение порядка пространственной аппроксимации достигается либо за счет использования полиномиальной реконструкции значений газодинамических переменных внутри сеточных ячеек [12], либо за счет применения квази-одномерной реконструкции переменных [13]. Более подробно описание моделей и алгоритмов можно найти в [14]. Далее в деталях будут рассмотрены особенности программной реализации и структуры вычислений.

3.2. Общая структура алгоритма

В общем виде схема алгоритма выполнения шага интегрирования по времени показана на Рис. 1. В конечно-объемном методе на неструктурированных сетках основу составляет расчет потоков через грани контрольных объемов. Этот этап полностью задействует программное представление неструктурированной расчетной области и геометрических данных по расчетным ячейкам и, таким образом, определяет шаблон вычислений и структуру доступа к памяти. Расчет потоков состоит из нескольких операций, включающих в себя вычисление коэффициентов полиномов, реконструкцию значений в центрах граней, расчет потоков, суммирование результирующих значений потоков с граней в центры ячеек. В следующем разделе подробно рассматриваются некоторые аспекты реализации этих операций.

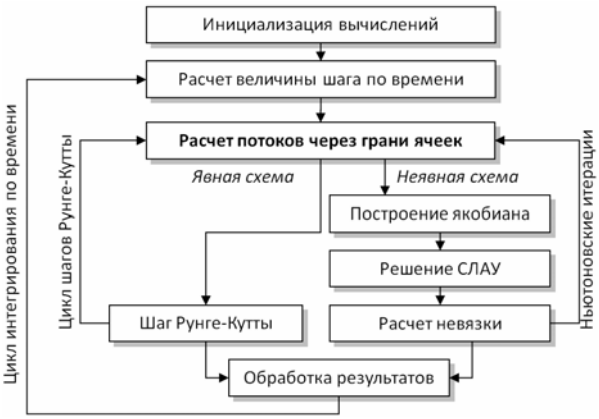

Рис. 1. Схема алгоритма выполнения шага интегрирования по времени

4. Адаптация к архитектуре графических процессоров

Расчет потоков, основа алгоритма, представляет собой множество однотипных заданий по вычислению потоков через грани расчетных ячеек по имеющимся значениям газодинамических переменных в ячейках, которым эта грань принадлежит. Для реконструкции значений в центрах граней по значениям в центрах ячеек используется один из методов повышения порядка аппроксимации: полиномиальная или квази-одномерная реконструкция газодинамических переменных [12, 13]. Также для расчетов необходимы геометрические параметры грани (площадь, вектор нормали и т.д.). Задания по вычислению потоков независимы по входным данным и могут обрабатываться параллельно. Однако возникает проблема с выходными данными. Определенный с использованием схемы Роу [11] поток через общую грань двух расчетных ячеек i и j суммируется в соответствующие массивы потоков по ячейкам, которые далее используются для нахождения значений газодинамических переменных на новом слое по времени. Тогда, если, например, параллельно вычислять потоки через грани между ячейками i - j и i - k, то существует большая вероятность возникновения ошибки при одновременном суммировании потоков для ячейки i двумя разными нитями. Различные варианты решения этой проблемы для CPU и для GPU рассматриваются, например, в [14] и [15], соответственно. В нашем случае расчет потоков разделен на несколько этапов с сохранением промежуточных результатов. Для наглядности, на Рис. 2 показан шаблон доступа к данным при расчете потоков через грань ячейки.

Сначала в каждой ячейке находятся коэффициенты полиномиальной реконструкции. Вычисление сводится для каждой ячейки к перемножению двух плотных матриц – матрицы геометрических коэффициентов размерности 9 на N_i и матрицы размерности N_i на 5, содержащей разности между значениями 5-ти газодинамических переменных в ячейках шаблона и значениями в центральной ячейке. Здесь N_i число ячеек в шаблоне схемы для i-й ячейки. В отличие от вычислений на процессоре, где зерном параллелизма, то есть элементарным заданием, является расчет коэффициентов в одной

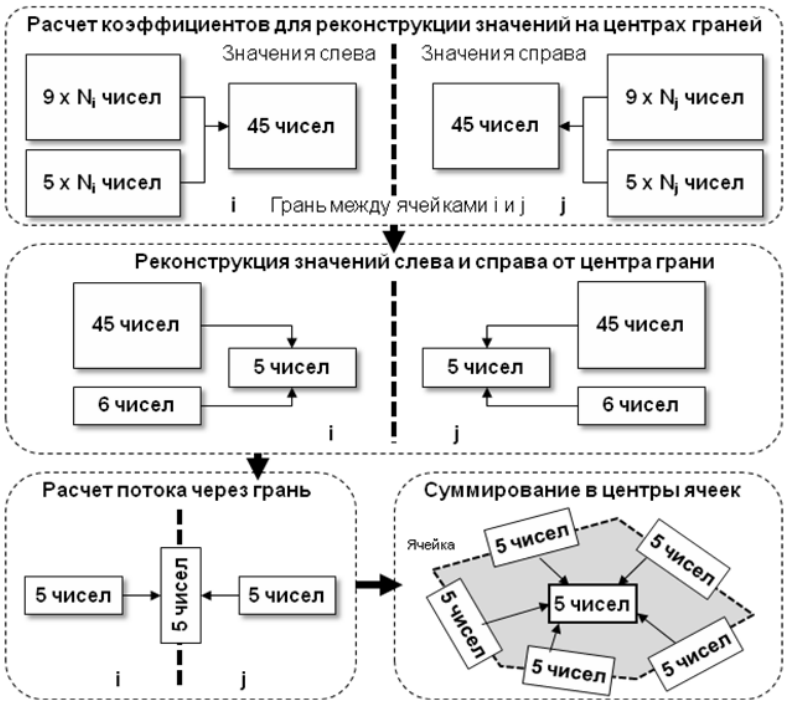

Рис. 2. Шаблон доступа к данным при расчете потоков через грань ячейки

ячейке, в случае GPU элементарное задание – это расчет лишь трех из 45 результирующих значений. При этом для достижения еще большей загрузки мультипроцессора одна ячейка обрабатывается группой из 32 нитей, которая разделяется на 2 подгруппы по 16 нитей (одна нить из которых простаивает). Каждая подгруппа выполняет половину задания, затем промежуточные результаты суммируются одной из подгрупп.

Затем реконструируются значения “слева” и “справа” от центров масс граней, по ним рассчитываются результирующие потоки через грани, которые суммируются в центры ячеек. В отличие от процессорной реализации, где эта операция выполняется за один этап, в случае GPU вычисления разделены на три этапа с использованием дополнительного массива для хранения промежуточных значений. Такое разделение на этапы необходимо для уменьшения ресурсоемкости алгоритма, то есть уменьшения расхода локальной и регистровой памяти, что позволяет повысить загрузку мультипроцессора.

На первом этапе выполняется реконструкция, затем на втором этапе вычисляется результирующий поток через грань, который записывается в промежуточный массив по граням, что исключает пересечение по данным между параллельными нитями. Наконец, на заключительном этапе выполняется операция суммирования вычисленных потоков в ячейки.

Реализация такого подхода требует выделения дополнительных массивов, хранящих значения газодинамических переменных на сеточных гранях, конвективные потоки через грани, индексы принадлежащих ячейкам граней. При таком подходе каждая из операций представляет собой набор независимых заданий, не представляющий проблем для распараллеливания ни на CPU, ни на GPU.

Как уже было отмечено, для рассматриваемого алгоритма ключевую роль играет оптимизация доступа к памяти. Последовательность доступа к памяти существенным образом определяется индексацией сеточных элементов. Для того чтобы на практике определить влияние метода индексации элементов дискретной модели на производительность программного обеспечения, будем сравнивать время работы приложений для двух вариантов индексации сеточных элементов: позитивной и негативной.

Позитивный метод индексации направлен на повышение эффективности работы с памятью

устройства. Индексация строится таким образом, чтобы как можно большее число соседних ячеек имели близкие индексы, то есть, чтобы соседние ячейки располагались как можно ближе в памяти для уменьшения числа кэш промахов. Индексация, то есть порядок обхода, организован по следующему принципу. Первый элемент последовательности обхода выбирается произвольным образом. Далее в конец последовательности в произвольном порядке добавляются индексы элементов, имеющих с первым элементом общую грань. Затем в последовательность добавляются элементы, имеющие общую грань со вторым элементом и ранее не вошедшие в последовательность, и так далее до момента обхода всех сеточных элементов. Негативная индексация элементов формируется из противоположных соображений.

Естественно, независимо от индексации ячеек, используется лексикографическая сортировка граней по индексам разделяемых ими ячеек, иначе производительность алгоритма снизится до недопустимого уровня (что, кстати, сильно увеличит ускорение на GPU по сравнению с CPU). Грани группируются по индексу первой ячейки и группы упорядочиваются в соответствии с этим индексом. Внутри групп грани сортируются по индексу второй ячейки.

Представленные далее результаты получены для CPU на параллельной реализации с OpenMP, реализация для GPU выполнена на OpenCL 1.1. Ускорение в Табл. 1-Табл.4 приводится относительно последовательного расчета на одном ядре процессора Intel Xeon X5670 2.93 GHz.

Результаты сравнения позитивной и негативной индексации представлены в Табл. 1 (моделируется обтекание сферы на сетке, содержащей 679339 элементов). Видно, что ускорение на GPU отличается примерно в полтора раза в зависимости от нумерации сеточных элементов. На CPU позитивная нумерация уменьшает число кэш промахов, значительно повышая производительность, в то время как на GPU влияние индексации намного меньше за счет механизма сокрытия латентности доступа к памяти. Такие результаты наглядно демонстрируют, что надо с осторожностью относиться к ускорению на GPU относительно CPU.

В Табл. 2 представлены сводные данные по производительности для позитивной индексации на CPU Intel Xeon X5670 (72 GFLOPS), GPU NVIDIA C2050 (515 GFLOPS), GPU AMD 7970 (945 GFLOPS). Ускорение на GPU относи-

Табл. 1. Сравнение позитивной и негативной индексации сеточных элементов

Вычислитель	Время, секунд		Ускорение	
	негативная	позитивная	негативная	позитивная
Вычисление коэффициентов полиномиальной реконструкции				
1 ядро Intel X5670	1.1	0.71	1	1
6 ядер Intel X5670	0.23	0.145	4.8	4.9
NVIDIA C2050	0.05	0.045	22.2	15.8
Реконструкция конвективных потоков				
1 ядро Intel X5670	0.45	0.26	1	1
6 ядер Intel X5670	0.11	0.06	4.1	4.3
NVIDIA C2050	0.018	0.017	25	15.3

Табл. 2. Производитель вычислений на GPU и CPU

Вычислитель	Время, секунд	GFLOPS	Процент от пика	Ускорение
Вычисление коэффициентов полиномиальной реконструкции				
1 ядро CPU	0.71	3.1	26%	1
6 ядер CPU	0.145	15.2	21%	4.8
NVIDIA C2050	0.045	49	10%	15.6
AMD 7970	0.014	160	17%	50.6
Реконструкция конвективных потоков				
1 ядро CPU	0.26	2.5	21%	1
6 ядер CPU	0.06	11	16%	4.3
NVIDIA C2050	0.017	38	7.4%	15.3
AMD 7970	0.006	115	12%	47

тельно 6-ядерного процессора составило не менее 3.3 раз, что превосходит соотношение в энергопотреблении, равное примерно 2.5. Также следует отметить, что на GPU AMD 7970 достигается в несколько раз большая производительность, чем на NVIDIA C2050. При этом эффективность использования устройства также оказывается намного выше.

5. Производительность на ускорителе архитектуры MIC

Массивно-параллельные ускорители архитектуры Intel MIC (Many Integrated Core) представляют собой альтернативу ускорителям GPU, однако устроены по иному принципу. На ускорителе Intel Xeon Phi 5110P доступны 60 ядер, имеющих общий кэш первого уровня. Устройство на аппаратном уровне поддерживает одновременное выполнение 240 независимых потоков команд, по 4 потока на ядро, что обеспечивает сокрытие латентности доступа к памяти, которая, как и в случае GPU, достаточно высока. Теоретическая пиковая производительность устройства составляет 1 TFLOPS. С точки зрения разработки наиболее существенное отличие от GPU – это возможность выполнять без каких-либо изменений тот же код, что и на CPU. В нашем случае на ускорителе Xeon Phi использовалась реализация с OpenMP распараллеливанием.

В Табл. 3 представлены результаты измерения производительности на MIC ускорителе при запуске различного числа потоков. Видно, что на данной архитектуре аппаратная поддержка множественных потоков на ядре оказывает существенно большее влияние на производительность, чем на CPU. Так, использование технологии Hyper-threading на CPU Intel Xeon E5-2690 для рассматриваемых алгоритмов дает выигрыш порядка 20% (2 нити на ядро). В случае MIC выигрыш составляет более чем 200%.

Сводные данные по производительности представлены в Табл. 4 для различных вычислителей, включая ускоритель NVIDIA GeForce GTX TITAN архитектуры Kepler (1.306 TFLOPS, 288 Gb/s). Из результатов видно, что на MIC получен самый низкий процент от теоретической пиковой производительности устройства. Фактическая производительность на MIC оказалась ниже, чем на GPU. С другой стороны, OpenMP реализация для MIC, в отличие от OpenCL реализации для GPU, не потребовала существенных трудозатрат.

Табл. 3. Производитель вычислений на MIC ускорителе

Число потоков	Время, секунд	GFLOPS	Процент от пика	Ускорение
Вычисление коэффициентов полиномиальной реконструкции				
16	0.46	4.7	0.5%	1.5
32	0.26	8.5	0.8%	2.7
60	0.125	17.6	1.7%	5.7
120	0.078	28.3	2.8%	9.1
240	0.059	37.4	3.7%	12
Реконструкция конвективных потоков				
16	0.20	3.25	0.3%	1.3
32	0.12	5.4	0.53%	2.2
60	0.054	12	1.2%	4.8
120	0.046	14.1	1.4%	5.7
240	0.028	23.2	2.3%	9.3

Табл. 4. Сравнение производительности на различных устройствах

Вычислитель	Время, секунд	GFLOPS	Процент от пика	Ускорение
1 ядро Intel Xeon X5690	0.97	2.9	24%	1
Intel Xeon X5690	0.205	14.3	20%	4.7
Intel Xeon E5-2690	0.131	22	11.9%	7.4
Intel Xeon Phi	0.087	33	3.3%	11.1
NVIDIA C2050	0.062	47	9.1%	15.6
NVIDIA GTX TITAN	0.031	86	6.7%	31.3
AMD 7970	0.02	147	15.5%	48.5

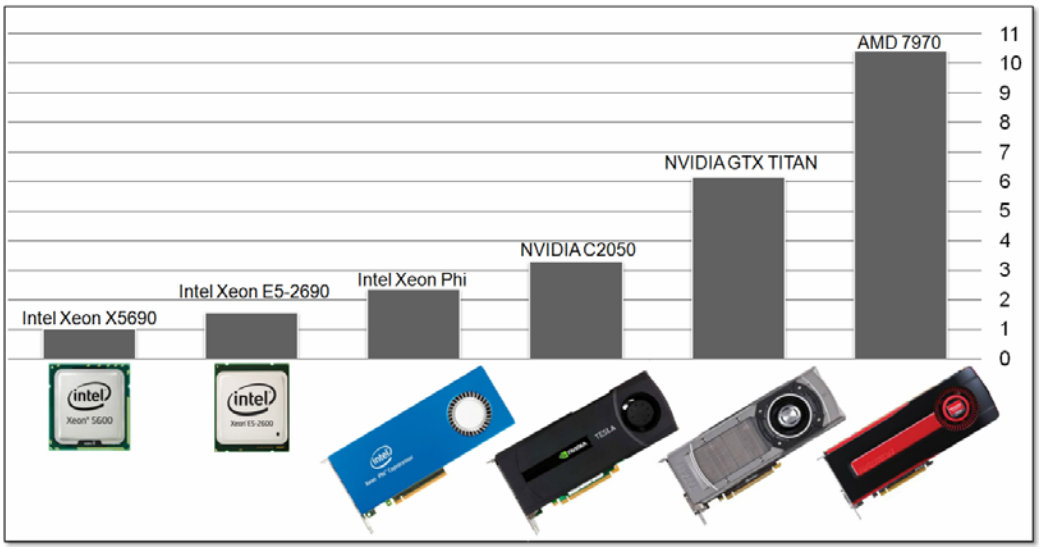

Рис. 3. Сравнение производительности различных вычислительных устройств

Табл. 5. Внутреннее ускорение на Intel Xeon Phi и сравнение с CPU Intel Xeon E5-2690

Операция	Внутреннее ускорение на MIC относительно 1 ядра	Производительность относительно 8-ядерного CPU
Расчет диссипативных потоков	148	0.65
Расчет узловых градиентов	131	0.83
Реконструкция потоков	198	1.08
Суммарно шаг по времени	181	0.96

Наконец, на Рис. 3 показано сравнение производительности для схемы повышенного порядка с полиномиальной реконструкцией в тестовом расчете на сетке, содержащей 679339 элементов на различных вычислительных устройствах.

Для того чтобы показать еще одно интересное явление, рассмотрим производительность на MIC для другой численной схемы, с определением значений переменных в сеточных узлах [13]. В отличие от схемы с центрами в элементах, число соседей контрольного объема уже не фиксировано (у тетраэдра – 4, у гексаэдра – 6), а может варьироваться в широком диапазоне (обычно порядка 20-30), что, естественно, влияет на организацию доступа к памяти. В качестве тестовой задачи рассматривается обтекание конуса сжимаемым течением. Расчетная сетка содержит 1018818 узлов, 6463947 тетраэдров.

Алгоритм имеет улучшенную OpenMP реализацию с двухуровневой декомпозицией расчетной области (для уменьшения числа интерфейсных элементов), рассчитанную на сотни параллельных потоков. Измерения выполнены для нескольких основных операций на шаге интегрирования по времени. Результаты представлены в Табл. 5. Внутреннее ускорение на MIC относительно последовательного расчета на одном ядре ускорителя составило 181 из 240 возможных, что показывает высокую внутреннюю параллельную эффективность. Однако любопытно, что, несмотря на такое высокое внутреннее ускорение, MIC все же проиграл CPU.

Данный пример показывает, что оптимизации под большое число потоков недостаточно, требуется переработка алгоритма для улучшения векторизации, ведь на Intel Xeon Phi векторные регистры уже вмещают 8 значений типа double (что в 2-4 раза больше, чем на CPU).

Заключение

Для конечно-объемного газодинамического алгоритма повышенного порядка аппроксимации на неструктурированных сетках выполнена реализация для массивно-параллельных ускорителей различных архитектур с использованием открытого вычислительного стандарта

OpenCL. В подробностях рассмотрены основные алгоритмические операции, представлены эффективные реализации для GPU. Полученное ускорение на GPU по сравнению с CPU заметно превысило соотношение в энергопотреблении сравниваемых устройств. Максимальная фактическая производительность, достигнутая на одном GPU устройстве, составила 160 GFLOPS. Выполнено сравнение производительности на CPU, GPU и ускорителях MIC. Хотя в представленных алгоритмах реализовано MPI распараллеливание, рассчитанное на тысячи параллельных процессов, в данной работе были рассмотрены только вычислительные аспекты в рамках архитектуры одиночного ускорителя.

На основании накопленного опыта работы с ускорителями AMD и NVIDIA, средствами разработки CUDA и OpenCL, авторы пришли к некоторым субъективным выводам относительно тенденций развития суперкомпьютерных архитектур. Рост производительности за счет применения все более изощренных типов ускорителей компенсируется все возрастающей сложностью их эффективного использования. Так, достигнутая авторами фактическая производительность на CPU составляет порядка 20% от теоретической пиковой производительности, что является достаточно высоким показателем для таких алгоритмов с низкой удельной вычислительной стоимостью на единицу данных и нерегулярным доступом к памяти, сопряженным с частыми кэш промахами. В случае GPU достигнутый процент от пика в среднем уже порядка 10%. Тут также следует отметить, что эффективность использования GPU AMD оказалась значительно выше, чем NVIDIA, как, впрочем, и производительность (что ставит под сомнение целесообразность использования CUDA в качестве средства разработки). В случае ускорителя MIC с трудом удается достичь 3% от пика. Если продолжить этот тренд, то в пределе получается вычислительное устройство с бесконечно большой производительностью и нулевой эффективностью использования.

Если при этом учесть возросшую сложность разработки и отладки, сложность многоуровневой параллельной модели MPI+OpenMP+ OpenCL(или CUDA), сложность вычислительной модели, и дополнить все это существенными различиями в архитектурах ускорителей и необходимостью адаптировать код под каждую новую архитектуру, то вопрос о целесообразности повсеместного применения ускорителей остается открытым. Складывается впечатление, что достижение эксафлопсной производительности является самоцелью. Если раньше процессоры разрабатывались, чтобы удовлетворять как можно более широкому кругу задач, то теперь скорее наоборот: вычислительные архитектуры создаются ради достижения все более высоких показателей производительности в ущерб применимости к реальным приложениям. Похоже, главной целью создания суперкомпьютеров стали показатели на тестах LINPACK. Продолжая эту тенденцию, можно представить сменяющие друг друга суперкомпьютеры будущего, обладающие все большей производительностью, но способные при этом решать все меньший круг задач, пока, наконец, не будет создан эксафлопсный или даже зеттафлопсный суперкомпьютер, на котором нельзя будет посчитать абсолютно ничего.

Справедливости ради стоит отметить, что совершенствуются и программные модели. Для облегчения программирования на гетерогенных системах разрабатываются стандарты, такие как OpenACC и OpenHMPP, в основе которых, по аналогии с OpenMP, лежат специальные директивы, а также продвинутые логические модели с концепцией исполняемого потока задач и трактов передачи данных, такие как StarPU, CGCM.

По сути, появление ускорителей, наконец, заставило серьезно задумываться над тем, как реально устроены вычислители и как работает на них программная реализация алгоритма. Если CPU прощали низкую культуру разработки и позволяли относительно легко получать приемлемую производительность, то теперь, похоже, время “дармовой” производительности закончилось.

Расчеты выполнялись на вычислительных ресурсах ИПМ им. М.В. Келдыша РАН, НИИСИ РАН и МСЦ РАН. Авторы выражают благодарность этим организациям.

Литература

1. MPI: A Message-Passing Interface Standard, version 3.0, 2012. http://www.mpi-forum.org/docs/docs.html

2. Monakov A., Lokhmotov A., Avetisyan A., Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures, High Performance Embedded Architectures and Compilers, Lecture Notes in Computer Science, 2010, Volume 5952/2010, 111-125.
3. M. Heller, T. Oberhuber, Adaptive row-grouped CSR format for storing of sparse matrices on GPU, ArXiv e-prints, 2012, http://arxiv.org/pdf/1203.5737.pdf
4. Nathan Bell and Michael Garland, Efficient Sparse Matrix-Vector Multiplication on CUDA, NVIDIA Technical Report (2008), NVR-2008-004, http://www.nvidia.com/object/nvidia_research_pub_001.html
5. NVIDIA-Corporation, NVIDIA CUDA C Programming Guide, http://docs.nvidia.com/cuda/index.html
6. OpenMP Application Program Interface, Version 3.1, July 2011, http://www.openmp.org/mp-documents/OpenMP3.1.pdf
7. Khronos Group, The OpenCL Specification, Version: 1.1, 2010; Version 1.2, 2011; http://www.khronos.org/opencl/
8. Peter Zaspel, Michael Griebel, Solving incompressible two-phase flows on multi-GPU clusters, Computers & Fluids (2012), In Press, Corrected Proof, Available online 31 January 2012, doi:10.1016/j.compfluid.2012.01.021
9. Е.В. Коромыслов, А.А. Синер, М.В. Усанин, Расчет на видеокартах генерации звука реактивной струей при истечении из модельного сопла, 4-я всероссийская конференция “Вычислительный эксперимент в аэроакустике” , 19-22 сентября 2012 года, г. Светлогорск, издательство МАКС пресс, Москва, стр. 99.
10. Christian Obrecht, Frédéric Kuznik, Bernard Tourancheau, Jean-Jacques Roux, Multi-GPU implementation of a hybrid thermal lattice Boltzmann solver using the TheLMA framework, Computers & Fluids (2012), In Press, Corrected Proof, Available online 6 March 2012, doi:10.1016/j.compfluid.2012.02.014
11. Roe P.L., Approximate Riemann Solvers, Parameter Vectors and Difference Schemes, J. Comput. Phys. 1981. 43, N2. 357--372.
12. Barth T., Numerical methods for conservation laws on structured and unstructured meshes, VKI for Fluid Dynamics, Lectures series, 2003-03.
13. Abalakin I., Dervieux A., Kozubskaya T., High accuracy finite volume method for solving nonlinear aeroacoustics problems on unstructured meshes, Chinese Journal of Aeroanautics. 2006. 19, N2. 97--104.
14. Абалакин И.В., Бахвалов П.А., Горобец А.В., Дубень А.П., Козубская Т.К., Параллельный программный комплекс NOISETTE для крупномасштабных расчетов задач аэродинамики и аэроакустики, Вычислительные методы и программирование, т.13 (2012), стр. 110-125.
15. А.В.Горобец, С.А.Суков, А.О.Железняков, П.Б.Богданов, Б.Н.Четверушкин, Применение GPU в рамках гибридного двухуровневого распараллеливания MPI+OpenMP на гетерогенных вычислительных системах, Вестник ЮУрГУ, №25 (242), 2011, стр. 76-86.

Горобец Андрей Владимирович. Старший научный сотрудник ИПМ им. М.В. Келдыша РАН. Окончил МГУ им. М.В. Ломоносова в 2003 году. Кандидат физико-математических наук. Автор 22 печатных работ. Область научных интересов: параллельные вычисления, газовая динамика, турбулентность, аэроакустика. E-mail: andrey.gorobets@gmail.com

Суков Сергей Александрович. Старший научный сотрудник ИПМ им. М. В. Келдыша РАН. Окончил МГТУ Станкин в 2000 году. Кандидат физико-математических наук. Автор 18 печатных работ. Область научных интересов: параллельные вычисления, газовая динамика, неструктурированные сетки. E-mail: pm18@imamod.ru

Богданов Павел Борисович. Научный сотрудник НИИСИ РАН. Окончил МГУ им. М. В. Ломоносова в 2003 году. Автор 3 печатных работ. Область научных интересов: суперЭВМ, параллельные вычисления, гетерогенные вычисления, математическое моделирование, вычислительные методы. E-mail: bogdanov@niisi.msk.ru