УДК 519.6

РЕАЛИЗАЦИЯ ДВУМЕРНОЙ ГАЗОДИНАМИЧЕСКОЙ МЕТОДИКИ Д НА ГИБРИДНОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Е. Е. Чупраков
(РФЯЦ-ВНИИЭФ)

Предлагается способ программной реализации разностных уравнений двумерной газодинамики на гибридной вычислительной системе, содержащей арифметические ускорители. Исследуются вопросы эффективности распараллеливания на нескольких ускорителях, описывается разработанная схема совместного использования ядер универсального процессора и арифметических ускорителей, реализованная на основе технологий MPI и CUDA.

Ключевые слова: арифметические ускорители (GPU), CUDA, MPI, методика Д.

Введение

Одним из основных требований к повышению качества математического имитационного моделирования, помимо собственно развития физических моделей и математических методов их решения, является требование роста производительности суперЭВМ. Глобальной целью для всей области высокопроизводительных вычислений в настоящее время является построение суперЭВМ производительностью 1 Эфлопс.

Реальной альтернативой традиционным процессорам становятся специализированные процессоры, ускоряющие вычисления за счет задействования большого количества энергоэффективных ядер. Одним из видов подобных ускорителей являются арифметические ускорители (GPU). Все более востребованы гибридные вычислительные системы на их основе, совмещающие вычислительные процессоры общего назначения и специализированные арифметические процессоры. Актуальной становится проблема совместного применения CPU и GPU в таких системах для решения прикладных научных задач.

Существенным препятствием к широкому использованию арифметических ускорителей является специализированная модель программирования для них. Чтобы получить преимущества от использования гибридных вычислительных систем, существующие программы, написанные для универсальных процессоров, должны быть значительно модифицированы. Очень часто алгоритмы для арифметических ускорителей требуется разрабатывать заново, отказываясь от традиционных подходов.

Существует уже достаточное количество успешных примеров применения арифметических ускорителей для задач, связанных с областями гидро- и газодинамики. Например, ускорители применяются для моделирования турбулентности по методу LES (крупных вихрей) [1], решения уравнений газодинамики [2, 3], расчета течений вязкой жидкости [4]. В подавляющем большинстве реализованных на арифметических ускорителях программ (как в указанных выше примерах, так и во многих других) используются численные методы, основанные на методе Годунова и неподвижной расчетной сетке. К ним, в частности, относится метод конечного объема.

В данной статье на примере давно и успешно эксплуатируемой в РФЯЦ-ВНИИЭФ газодинамической методики Д [5] рассматривается вопрос программной реализации решения уравнений двумерной газодинамики на гибридных вычислительных системах с несколькими GPU по методу конечных разностей с применением лагранжева подхода к расчету течений газа. Используется разнесенная структурированная расчетная сетка, движущаяся вместе с веществом. Сама конечно-разностная методика Д основана на явной разностной схеме, имеющей сеточный шаблон типа *крест*.

В статье исследуется эффективность распараллеливания такой схемы на арифметических ускорителях с помощью технологии CUDA [6]. Подробно рассматривается разработанный алгоритм решения уравнений газодинамики по методике Д на гибридной вычислительной системе. Приводятся результаты тестовых расчетов и полученные ускорения.

Исходные уравнения и их разностная аппроксимация

Исходная система газодинамических уравнений имеет следующий вид:

$$\begin{aligned} \rho\frac{d\vec{U}}{dt} &= -\operatorname{grad} p; \\ \frac{d\rho}{dt} &= -\rho \operatorname{div}\vec{U}; \\ \frac{dE}{dt} &= -p\frac{d\,(1/\rho)}{dt}; \\ \frac{d\vec{r}}{dt} &= \vec{U}, \end{aligned} \tag{1}$$

где $\vec{r} = (x, y)$ — радиус-вектор точки; $\vec{U} = (U, V)$ — вектор скорости; ρ — плотность; p — давление; E — внутренняя энергия; t — время. Система (1) замыкается уравнением состояния вида

$$p = p\,(\rho, E)\,.$$

В методике Д используется разнесенная структурированная разностная сетка, в узлах которой хранятся скорости и координаты (*узловые* величины), а в центрах ячеек – термодинамические величины (*ячеечные*).

В [7] получена аппроксимация уравнения движения системы (1) для узла 0 (рис. 1). Движение узла сетки описывается уравнением вида

$$m_0\frac{d\vec{U}_0}{dt} = -\vec{P}_0. \tag{2}$$

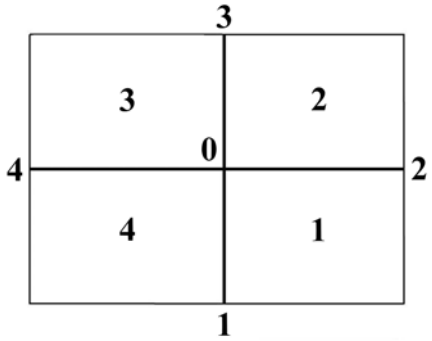

Рис. 1. Фрагмент расчетной сетки

Здесь $m_0 = \frac{1}{4}\sum\limits_{i=1}^{4} M_i$ — масса узла 0, где M_i — масса ячейки i, прилегающей к узлу 0; $\vec{P}_0 = (P_{x0}, P_{y0})$ — вектор разностного градиента давления. Выражения для компонент вектора $\vec{P}_0$ приведены в [7] и имеют следующий вид:

$$\begin{aligned} P_{x0} = \frac{1}{6}\Bigg\{ & \sum_{i=1}^{3} (y_{i+1} + y_0 + y_i)\,(y_{i+1} - y_i)\,p_i + \\ & + (y_1 + y_0 + y_4)\,(y_1 - y_4)\,p_4 \Bigg\}; \end{aligned}$$

$$\begin{aligned} P_{y0} = -\frac{1}{6}\Bigg\{ & \sum_{i=1}^{3} \Big[(x_{i+1} - x_0)\,y_{i+1} + 2\,(x_{i+1} - x_i)\,y_0 + \\ & + y_i\,(x_0 - x_i)\Big]p_i + \Big[(x_1 - x_0)\,y_1 + \\ & + 2\,(x_1 - x_4)\,y_0 + y_4\,(x_0 - x_4)\Big]p_4 \Bigg\}. \end{aligned}$$

Таким образом, для вычисления скорости на сетке используется разностный шаблон типа *крест* (рис. 2), для которого необходим доступ к узловым величинам во всех пяти точках шаблона, а также к ячеечным величинам в окружающих центральный узел ячейках.

Рассмотрим разностную аппроксимацию уравнения движения. В методике Д значения скоростей отнесены к полуцелым временным слоям $t^{n+1/2}$, остальные величины — к целым слоям t^n. Уравнение (2) будет иметь следующий разностный вид:

$$\begin{aligned} & m_0\frac{\vec{U}_0^{\,n+1/2} - \vec{U}_0^{\,n-1/2}}{\tau^n} = -\vec{P}_0^{\,n}; \\ & t^{n+1} = t^n + \tau^{n+1/2}; \\ & \tau^n = \frac{\tau^{n+1/2} + \tau^{n-1/2}}{2}. \end{aligned}$$

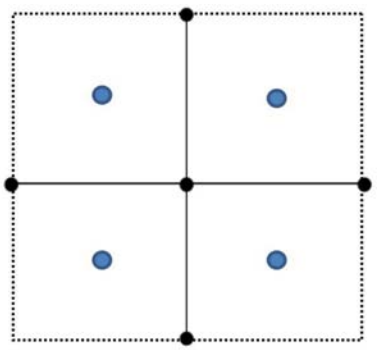

Рис. 2. Разностный шаблон методики Д

Координаты узла 0 (см. рис. 1) на следующем временном слое задаются выражением

$$\vec{r}_0{}^{n+1} = \vec{r}_0{}^{n} + \vec{U}_0^{n+1/2}\tau^{n+1/2}.$$

Для вычисления новых координат узла сетки достаточно знать только новые скорости в том же узле и шаг по времени, одинаковый для всех узлов.

Рассмотрим механизм вычисления термодинамических величин в ячейках.

Новые плотности вычисляются из закона сохранения массы

$$\rho^{n+1}\omega^{n+1} = \rho^{n}\omega^{n},$$

где ω — объем ячейки.

Для сквозного расчета ударных волн вводится искусственная счетная вязкость типа вязкости Неймана—Рихтмайера, имеющая размерность давления. Она вычисляется по формуле

$$q = \beta H^2 \frac{1}{\rho}\left(\frac{\partial\rho}{\partial t}\right) = \beta\frac{H^2}{2}\frac{1}{\rho}\frac{\partial\rho}{\partial t}\left(\left|\frac{\partial\rho}{\partial t}\right| + \frac{\partial\rho}{\partial t}\right),$$

где H — характерный размер ячейки; β — постоянный коэффициент. Характерный размер ячейки есть функция координат ее вершин.

Вычисление энергии и давления на момент времени t^{n+1} осуществляется на основе системы уравнений

$$E^{n+1} = E^{n} - 0{,}5\left(p^{n+1} + p^{n}\right)\left(\frac{1}{\rho^{n+1}} - \frac{1}{\rho^{n}}\right);$$
$$p^{n+1} = f\left(\rho^{n+1}, E^{n+1}\right) + q^{n+1},$$

которая решается методом итераций.

Таким образом, для вычисления термодинамических величин в ячейке необходимо иметь доступ к величинам в ее вершинах (узлах сетки) и в ее центре (рис. 3).

От описанных шаблонов доступа к сеточным величинам, которые следуют из разностных уравнений, будет зависеть и организация данных в памяти GPU.

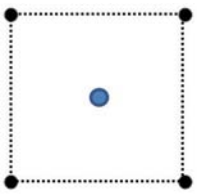

Рис. 3. Шаблон доступа к ячейке

Схема совместного применения CPU и GPU

Рассмотрим алгоритм совместного использования ядер универсального процессора и арифметических ускорителей.

Схема совместного использования CPU и GPU предполагает разделение сетки в домене на две зоны (рис. 4). Зона 1 — *внешний слой* узлов и ячеек (может иметь разную глубину в зависимости от используемых алгоритмов) — это слой MPI-обменов и/или (нестационарных) граничных условий. Расчет узлов и ячеек этой зоны ведется на CPU. Зона 2 — *центральная область*, рассчитывается на GPU.

Для независимого расчета обеих зон между ними необходим обмен данными на каждом счетном шаге. Обмен осуществляется с помощью трех обменных слоев (*гало*), схематично изображенных на рис. 5. Слой 1 — это слой узлов, рас-

Рис. 4. Декомпозиция в домене

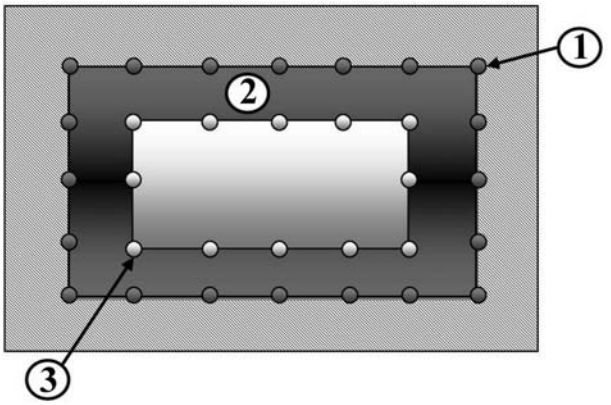

Рис. 5. Обменные слои

считываемых на CPU и передаваемых на GPU для расчета внутренних узлов и ячеек зоны 2. Слой 2 — это слой ячеек, рассчитываемых на GPU и передаваемых на CPU. Слой 3 — это слой узлов, рассчитываемых на GPU и передаваемых на CPU. Слои 2 и 3 необходимы для расчета узлов слоя 1 на CPU, которые являются общими узлами зоны 1 и зоны 2.

Таким образом, между CPU и GPU реализована парадигма перекрытия по данным, стандартная для любой MPI-программы. Отличие состоит лишь в том, что в MPI-программах перекрытие по данным осуществляется между соседними процессорами, тогда как здесь обмен данными происходит между процессором и ускорителем.

Для рассматриваемой декомпозиции счетный шаг решения системы разностных уравнений, аппроксимирующих исходные газодинамические уравнения, можно представить схемой, изображенной на рис. 6 (см. также рис. 4, 5).

Поскольку при данной схеме распараллеливания на счетном шаге совершаются только минимально необходимые односторонние обмены между CPU и GPU — двумя слоями узлов и одним слоем ячеек, такая схема является наиболее экономичной по числу обменов данными между CPU и GPU из всех возможных для рассматриваемого численного метода.

Для осуществления *выровненного* доступа (memory coalescing) к глобальной памяти GPU каждая строка ячеек сетки разбивается на блоки размером BLOCK_SIZE. При этом возможно появление фиктивных ячеек для полного заполнения блоков. На рис. 7 блоки, содержащие пустые ячейки для выравнивания, представлены в виде неполностью закрашенных прямоугольников.

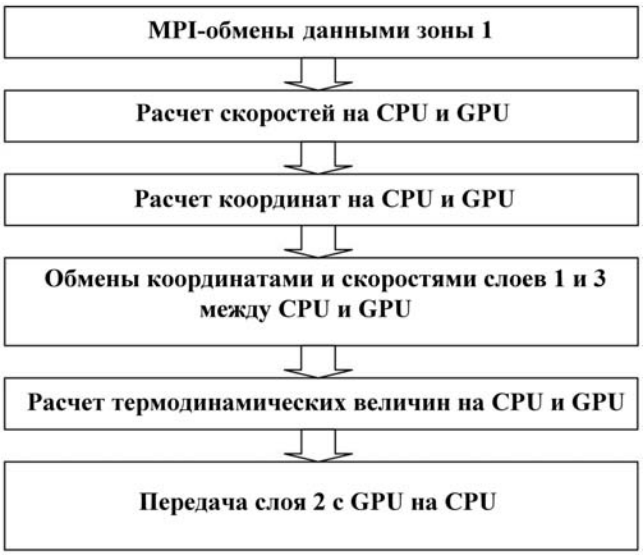

Рис. 6. Схема счетного шага

При расчете на GPU количество независимых потоков исполнения (нитей) в блоке равно длине блока BLOCK_SIZE. При расчете ячеечных величин каждая нить блока рассчитывает величину в собственной ячейке (рис. 8).

При расчете скорости массивы в разделяемой памяти расширяются за счет ячеек-фантомов справа (крайний узел слева всегда будет принадлежать границе гало и не обсчитывается). Это дает возможность задействовать все нити блока — по одной на каждый узел (рис. 9).

Для хранения промежуточных данных в ядрах CUDA (программах, исполняемых на GPU) используется разделяемая (shared) память, которая заполняется всеми нитями блока.

Еще одна проблема, связанная с разнесенной структурированной сеткой, — несовпадение длин ячеечных и узловых массивов: количество узлов в каждой строке сетки всегда на единицу больше количества ячеек (рис. 10). Для выравнивания по длине узловых и ячеечных массивов из узловых массивов на GPU исключаются элементы, соответствующие крайним левым узлам всех строк сетки (рис. 11). При рассматриваемой де-

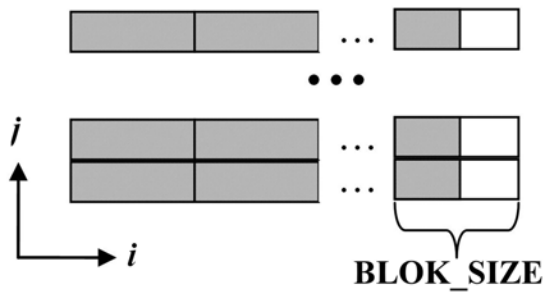

Рис. 7. Декомпозиция на GPU (i — индекс столбца, j — индекс строки)

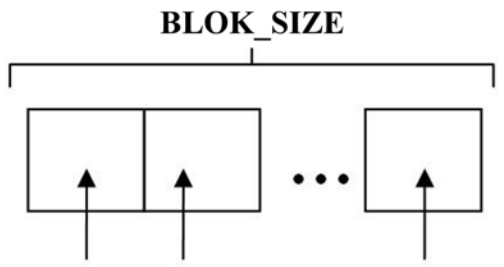

Рис. 8. Распределение нитей при расчете ячеечных величин

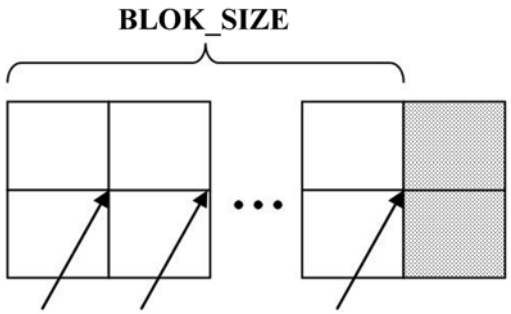

Рис. 9. Распределение нитей при расчете скорости

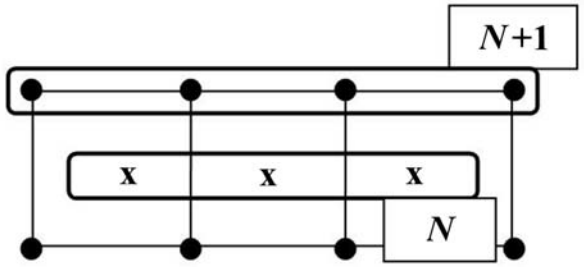

Рис. 10. Структурированная разнесенная сетка

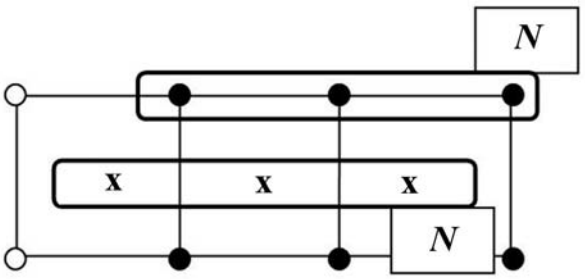

Рис. 11. Выравнивание размеров узловых и ячеечных массивов

композиции они хранятся в массивах слоя 1, передаваемых с CPU.

При работе на GPU с разностным шаблоном, невыровненным в глобальной памяти GPU, имеется доступ к узлам и ячейкам (к значениям рассчитываемых в них величин), расположенным в одном столбце сетки, но недоступны узлы и ячейки, лежащие в одной строке. Возможно, в дальнейшем эту проблему удастся решить, но в текущей реализации она остается.

Модель программирования для нескольких GPU

Сценарий использования узлов гибридной вычислительной системы может выглядеть следующим образом (рис. 12). Взаимодействие между узлами осуществляется с помощью библиотеки MPI, между ядрами одного узла возможно взаимодействие посредством OpenMP или MPI. Связь ядра CPU с GPU программно реализуется с помощью CUDA. Для межпроцессорных обменов в данной реализации всюду использовалась библиотека MPI.

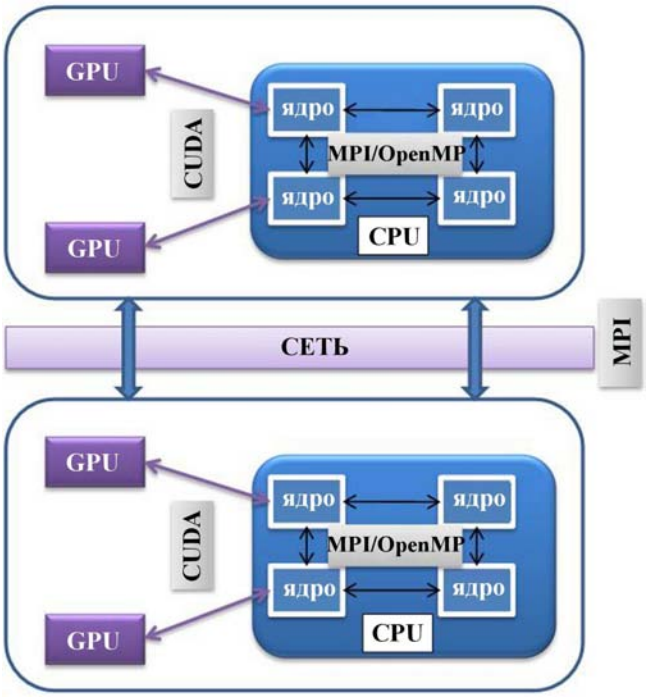

Рис. 12. Схема коммуникаций в гибридном кластере

Существует несколько возможных сценариев использования связки MPI + CUDA на гибридном узле.

Первый вариант — когда желательно задействовать все имеющиеся процессорные ядра, но их больше, чем GPU-устройств на узле. Этот вариант на практике является труднореализуемым, так как требует сложной синхронизации действий по переключениюю контекстов на GPU и управлению памятью (здесь возможно использование закрепленной памяти, но не понятно, как это скажется на производительности).

Второй вариант — когда используется число ядер, равное числу GPU. В идеальном случае задействуются все процессорные ядра узла, но если ядер физически больше, чем GPU-устройств, то они будут простаивать.

Третий вариант, когда число доступных процессорных ядер меньше числа GPU-устройств, легче для реализации, чем обратная ситуация, и тоже может рассматриваться как перспективный.

В данном случае реализован наиболее простой сценарий с прикреплением одного ядра CPU к одному GPU. Очевидный недостаток такого распределения состоит в том, что незадействованные ядра CPU могут простаивать. В дальнейшем возможно исследование более совершенного распределения нагрузки на ядра CPU со стороны приложения посредством использования незадействованных в связке с GPU ядер для выполнения некоторой части работы.

Каждое ядро CPU на узле «видит» только GPU-устройства на том же узле. Для однозначного выбора GPU каждым MPI-процессом необходимо знать идентификаторы всех процессоров на том же узле. Можно предложить следующий алгоритм:

1) получить имя узла с помощью функции MPI_Get_processor_name();
2) вычислить хэш (целое число) на основе имени узла;
3) создать внутриузловой коммуникатор, используя значение хэша в качестве параметра цвета для функции MPI_Comm_split();
4) с помощью идентификатора процесса во внутриузловом коммуникаторе "привязать" MPI-процесс к GPU-устройству функцией cudaSetDevice(node_rank).

При распараллеливании на нескольких GPU выбирается одномерная декомпозиция по строкам (из-за построчного расположения сетки в памяти). Одному потоку MPI соответствует один GPU (рис. 13).

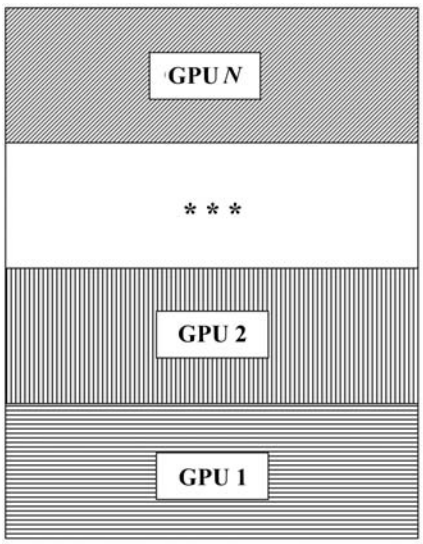

Рис. 13. Декомпозиция задачи в случае нескольких GPU

Результаты тестов

Для тестирования была выбрана двумерная осесимметричная задача о сходящейся автомодельной ударной волне [8]. Постановка задачи взята из [9]: начальная расчетная область представляет собой полукруг радиусом $R_0 = 1$ с центром в начале координат, заполненный идеальным газом с показателем адиабаты $\gamma = 5/3$ и плотностью 1. Внешняя граница является свободной поверхностью с заданной на ней зависимостью давления от времени.

Постановка расчета производилась в области $0^\circ < \varphi < 180^\circ$ на двумерной сферической сетке размерами 200×200 (40 тыс.), 400×400 (160 тыс.), 800×800 (640 тыс.) и $1\,000 \times 1\,000$ (1 млн) узлов до момента времени 0,5. Расчет выполнялся в режиме двойной точности. Для поиска минимального шага по времени в ячейках применялась библиотека Thrust v.1.3. Результаты расчетов с использованием GPU совпадают с эталонными результатами, полученными с использованием только CPU, и отдельно не приводятся.

В табл. 1 приведены результаты замеров длительности выполнения расчета для варианта, в котором вычислитель, состоящий из одного ядра CPU и одного GPU-устройства, сравнивается с одним ядром CPU (библиотека MPI не задействована). При этом использовалась следующая тестовая конфигурация: CPU Intel(R) Core (TM) i7 920 2.67GHz, GPU NVIDIA Tesla C2050, компилятор GCC v.4.1.2 с ключом —O2, CUDA v.3.0 с ключом —arch sm_13. Из таблицы видно, что с увеличением размера задачи ускорение возрастает, в частности, для задачи с 1 млн счетных ячеек получено 14-кратное пиковое ускорение.

Для изучения эффективности распараллеливания на нескольких GPU использовалась следующая тестовая конфигурация (для одного узла гибридной вычислительной системы): два процессора CPU Intel X5670 2.93GHz (6 ядер), три ускорителя GPU NVIDIA Tesla C2050, компилятор Intel C++ v.11.1.069 с ключом —O2, CUDA v.4.0 с ключом —arch sm_13.

В табл. 2, 3 приведены замеры длительности выполнения расчета и полученные ускорения относительно одного ядра CPU. Режим совместного использования CPU и GPU подразумевает взаимодействие одного ядра CPU с одним GPU-устройством (для N GPU понадобится N ядер CPU). При этом CPU рассчитывает только небольшой слой окружающих центральную об-

Таблица 1

Длительность выполнения расчета (с) и ускорение, полученные без использования MPI

Конфигурация	200 × 200 (40 000)	400 × 400 (160 000)	800 × 800 (640 000)	1 000 × 1 000 (1 млн)
1 CPU	28,8	239,9	1 896,4	3 718,2
1 CPU + 1 GPU	6,41	26,5	143,9	269,8
Ускорение	4,5	9	13,2	**13,8**

Таблица 2

Длительность выполнения расчета (с) при использовании MPI

Размер сетки	1 GPU + 1 CPU	2 GPU + 2 CPU	3 GPU + 3 CPU	1 CPU	12 CPU	24 CPU
200 × 200	11,9	9,7	9,1	28,19	2,8	——
400 × 400	37,62	28,89	24,66	232,16	21,87	11,48
800 × 800	189,79	118,64	94,9	1 870,11	172,21	88,91
1 000 × 1 000	362,01	208,64	163,12	3 692,77	337,57	178,54

Таблица 3

Ускорение относительно одного CPU

Размер сетки	1 GPU + 1 CPU	2 GPU + 2 CPU	3 GPU + 3 CPU	1 CPU	12 CPU	24 CPU
200 × 200	2,4	2,9	3,1	1	10,0	——
400 × 400	6,2	8,0	9,4	1	10,6	20,2
800 × 800	10,0	15,8	19,7	1	10,9	21,0
1 000 × 1 000	10,2	17,7	22,6	1	10,9	20,7

ласть ячеек, а основной расчет проводится на GPU. Режим, обозначенный в таблицах как N CPU, подразумевает выполнение расчета на N ядрах центрального процессора без использования GPU.

Из-за более производительного (по сравнению с рассмотренным ранее) процессора в данной тестовой конфигурации пиковое ускорение при сравнении с одним ядром CPU вычислителя, состоящего из одного ядра CPU и одного GPU-устройства, на задаче с 1 млн ячеек составило 10,2 раза. Ускорение по сравнению с одним ядром CPU для вычислителя из трех GPU и трех CPU составило 22,6 раза и также достигнуто при максимальном размере сетки. В то же время наибольшее ускорение для CPU-варианта программы достигло высшего предела (21 раз) при 24 ядрах CPU и получено уже на задаче с 640 тыс. ячеек. Можно предположить, что в отличие от расчетов с использованием CPU с увеличением размера задачи ускорение на гибридной вычислительной системе продолжит расти.

Чтобы понять, во сколько раз быстрее программа будет выполняться при одинаковом увеличении количества ядер CPU и количества GPU-устройств, оценим полученные ускорения распараллеливания для CPU- и GPU-вариантов программы по следующей формуле:

$$S_N = \frac{T_1}{T_N},$$

где T_N — время выполнения распараллеленной программы на N ядрах CPU (либо на одном вычислителе, состоящем из N GPU и N CPU); T_1 — время выполнения исходной программы на одном ядре CPU (либо на одном вычислителе из одного GPU и одного CPU). Результаты приведены в табл. 4.

Эффективность распараллеливания оценим по формуле

$$E_N = \frac{S_N}{N} = \frac{T_1}{NT_N}.$$

Из результатов (табл. 5) видно, что с увеличением числа ячеек эффективность распараллеливания на гибридной вычислительной системе только повышается, тогда как для CPU-программы она достигает высшего предела уже на задаче с 640 тыс. ячеек (хотя и остается при этом достаточно высокой).

Размер задачи в 1 млн ячеек является характерным для расчетов по двумерной методике Д. Большее количество счетных ячеек выбирается достаточно редко и только для отдельных задач из-за исключительной экономичности самой методики. При этом очевидно, что большее количество ячеек и большее количество арифметических операций положительно скажется на эффективности GPU-распараллеливания. Сохраняется также и потенциал роста эффективности посредством оптимизации CUDA-программ, главным образом с помощью применения улучшенных шаблонов доступа к глобальной памяти GPU.

Заключение

Исследована реализация программы двумерной газодинамики для гибридной вычислительной системы с несколькими арифметическими ускорителями. В частности, разработана экономичная схема совместного использования ядер универсального процессора и арифметических ускорителей, реализованная на основе технологий MPI и CUDA. Пиковое ускорение, достигнутое вычислителем, состоящим из одного ядра CPU и одного GPU, составило 10 и 14 раз (в зависимости от тестовой конфигурации) по сравнению с одним ядром CPU.

Выполнены тестовые расчеты на гибридной вычислительной системе, по результатам которых оценена эффективность распараллеливания на нескольких GPU. С увеличением размера задачи эффективность GPU-распараллеливания повышается, тогда как для CPU-программы она достигает высшего предела без последующего роста.

Для более полной оценки асимптотики эффективности распараллеливания на гибридной вычислительной системе представляется целесообразным провести дальнейшие исследования для трехмерной схемы методики Д.

Список литературы

1. *DeLeon R., Senocak I.* GPU-accelerated large-eddy simulation of turbulent channel flows // 50th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition. Nashville, Tennessee. January 09—12, 2012.
2. *Геллер О. В., Васильев М. О., Холодов Я. А.* Построение высокопроизводительного вы-

Таблица 4

Ускорение распараллеливания

Размер сетки	1 GPU + 1 CPU	2 GPU + 2 CPU	3 GPU + 3 CPU	1 CPU	12 CPU	24 CPU
200×200	1	1,2	1,3	1	10,0	——
400×400	1	1,3	1,5	1	10,6	20,2
800×800	1	1,6	2,0	1	10,9	21,0
$1\,000 \times 1\,000$	1	1,7	2,2	1	10,9	20,7

Таблица 5

Эффективность распараллеливания

Размер сетки	1 GPU + 1 CPU	2 GPU + 2 CPU	3 GPU + 3 CPU	1 CPU	12 CPU	24 CPU
200×200	1	0,6	0,43	1	0,83	——
400×400	1	0,65	0,5	1	0,88	0,84
800×800	1	0,8	0,66	1	0,91	0,88
$1\,000 \times 1\,000$	1	0,85	0,73	1	0,91	0,86

числительного комплекса для моделирования задач газовой динамики // Компьютерные исследования и моделирование. 2010. Т. 2, № 3. С. 309—317.

3. *Рыбакин Б. П., Шидер Н. И.* Построение параллельных алгоритмов для решения задач гравитационной газовой динамики // Вычислительные методы и программирование. 2010. Т. 11. С. 388—394.
4. *Castonguay P., Williams D. M., Vincent P. E. et al.* On the development of a high-order, multi-GPU enabled, compressible viscous flow solver for mixed unstructured grids // 20th AIAA Computational Fluid Dynamics Conference. Honolulu, Hawaii. June 27—30, 2011.
5. *Дмитриев Н. А., Дмитриева Л. В., Малиновская Е. В., Софронов И. Д.* Методика расчета двумерных нестационарных задач газодинамики в переменных Лагранжа: Препринт № 59. М.: ИПМ АН СССР, 1976.
6. Технология CUDA. http://www.nvidia.ru/object/cuda_home_new_ru.html.
7. *Делов В. И., Сенилова О. В., Софронов И. Д.* Конструирование разностных схем для расчета двумерных нестационарных упругопластических течений на основе закона взаимного превращения кинетической и внутренней энергий // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 1996. Вып. 4. С. 67—72.
8. *Седов Л. И.* Методы подобия и размерности в механике. М.: Наука, 1981.
9. *Бондаренко Ю. А., Воронин Б. Л., Делов В. И. и др.* Описание системы тестов для двумерных газодинамических методик и программ. Ч. 1. Требования к тестам. Тесты 1—7 // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 1991. Вып. 2. С. 3—9.

Статья поступила в редакцию 20.08.12.