ГЕТЕРОГЕННОЕ ПРОГРАММИРОВАНИЕ В РАМКАХ СТАНДАРТА OPENCL

П. Б. Богданов, О. Ю. Сударева

НИИ системных исследований РАН, г. Москва

1. Введение

В данной статье описаны результаты исследований и разработок в области гетерогенных вычислений, полученные в НИИСИ РАН. Разработана программная инфраструктура гетерогенных вычислений («гетерогенный планировщик»). Предложена методика оценки производительности алгоритма на основе модели гетерогенной вычислительной системы. С применением этой методики исследованы три алгоритма из NAS Parallel Benchmarks: 3-мерное ДПФ (FT), метод сопряженных градиентов (CG) и многосеточный метод (MG). Полученные теоретические оценки позволяют утверждать, что гетерогенная реализация оправданна для алгоритмов CG и MG и не оправданна для алгоритма FT. Разработаны гетерогенные реализации алгоритмов CG и MG, в стандарте OpenCL с использованием планировщика. Для этих реализаций проведены замеры производительности на мини-суперЭВМ НИИСИ РАН, а также на суперЭВМ К100 для процедуры MG. Результаты замеров подтверждают сделанный теоретический прогноз.

2. Подход НИИСИ РАН к исследованиям в области гетерогенных вычислений

Основное направление деятельности группы гетерогенных вычислений – разработка архитектуры перспективного многопроцессорного вычислительного комплекса, предназначенного для решения широкого класса задач трехмерного моделирования. Чтобы разработать такой комплекс, необходима встречная оптимизация архитектуры и алгоритмов. Другими словами, требуется ответить на два вопроса: для каких алгоритмов возможна эффективная реализация на выбранной архитектуре, и какими характеристиками должна обладать архитектура, чтобы на ней была возможна эффективная реализация того или иного алгоритма.

Среди исследуемых алгоритмов: BLAS и LINPACK, NAS Parallel Benchmarks, БПФ, отдельные алгоритмы вычислительной гидродинамики, итерационные решатели СЛАУ с разреженными матрицами. В качестве основы для гетерогенного программирования был выбран переносимый стандарт OpenCL [1], который поддерживается большинством современных CPU (универсальных процессоров) и GPU (графических ускорителей, или карт). Аппаратура, на которую мы ориентируемся: универсальные процессоры мировых производителей, графические ускорители общего назначения, сопроцессор Intel Xeon Phi, отечественный микропроцессор, разработанный в НИИСИ РАН – КОМДИВ128-РИО (в рамках данной работы подробно обсуждаться не будет).

Тестовые установки имеют так называемую гибридную архитектуру. Они собраны из розничных комплектующих, представленных в настоящее время на рынке: универсальных процессоров производства компании Intel различной архитектуры и графических ускорителей от компаний AMD и NVidia. Современные графические ускорители имеют пиковую производительность порядка 1–2 Тфлоп/с (на вычислениях с числами двойной точности) и скорость доступа к внутренней памяти 260–320 Гбайт/с.

С использованием ускорителей собрана суперЭВМ К100, развернутая в ИПМ им. М. В. Келдыша РАН [2], на базе CPU Intel Xeon X5670 и GPU NVidia Tesla C2050. Вычислительный комплекс на несколько узлов с использованием современных шасси исследователь может собрать у себя «на столе». Такой установки достаточно, чтобы выявить особенности и узкие места реализации того или иного алгоритма на гибридных архитектурах в целом, при этом исчезает проблема совместного использования суперЭВМ большим количеством пользователей.

Например, установки, на которых мы начали работать в 2014 году, собраны на базе двух платформ. На их основе, с использованием различных GPU, можно построить гибридные мини-суперЭВМ:

- на платформе с условным названием «TYAN»:
  – с пиковой производительностью 12–20 Тфлоп/с (при вычислениях с двойной точностью),
  – энергопотреблением приблизительно 2,5 кВт,
  – общей стоимостью 20–50 тыс. $;
- на платформе «SuperServer Supermicro»:
  – с пиковой производительностью 18–30 Тфлоп/с (при вычислениях с двойной точностью),
  – энергопотреблением приблизительно 4,2 кВт,
  – общей стоимостью 55–80 тыс. $.

Частичные результаты работы группы гетерогенных вычислений отражены в публикациях [3–6] и выступлениях на конференциях [7–11].

3. Программная инфраструктура гетерогенных вычислений

В НИИСИ РАН разработана программная инфраструктура гетерогенных вычислений – так называемый «гетерогенный планировщик» (подробно см. в [3]). Гетерогенный планировщик предназначен для гибридных установок, в которых имеется управляющий процессор и один или несколько сопроцессоров под его управлением. Инфраструктура представляет собой «надстройку» над API OpenCL, которая позволяет обращаться к основным командам в более короткой и удобной для программиста форме.

В парадигме ПИГВ сопроцессор выполняет инструкции (вычислительные ядра для устройства, т. е. программы, которые выполняются на устройстве и работают с данными, располагающимися в его памяти) на регистрах (линейных областях в глобальной памяти устройства). Планировщик может формировать и запускать 4 вида команд, которые образуют 4 очереди:

– LD – команды загрузки данных из общей памяти в память сопроцессора;
– ST – команды выгрузки данных из памяти сопроцессора;
– EXEC – исполнение инструкций (ядер) на сопроцессоре;
– CPU – процедуры, исполняемые на управляющем процессоре.

Команды, готовые к исполнению и стоящие в разных очередях, запускаются параллельно. При этом в очередях EXEC и CPU команды исполняются строго последовательно, а в очередях LD и ST – асинхронно. В целом, программа для планировщика представляется как граф зависимостей между командами различных типов. Команда считается готовой к исполнению, если выполнены все команды, от которых она зависит.

Как правило, на распределенной установке планировщик используется совместно с MPI: в каждом MPI-процессе запускается свой планировщик. При этом один MPI-процесс может управлять одним узлом и всеми сопроцессорами на нем либо одним сопроцессором.

Ранее в НИИСИ РАН были разработаны гетерогенные реализации на OpenCL с использованием ПИГВ для нескольких классических тестов. Работа по реализации этих тестов, а также по разработке самого гетерогенного планировщика проводилась при участии А. А. Ефремова. Первый набор тестов – процедуры третьего уровня BLAS (о наборе процедур см. [12]): DGEMM (умножение плотных матриц), DTRSM (решение СЛАУ с треугольной матрицей) и DGETRF (LU-разложение). Гетерогенные реализации НИИСИ РАН сопоставимы по эффективности с известной реализацией MAGMA (о MAGMA см. [13]), причем показывают лучшую масштабируемость: до 8 ускорителей. На основе разработанной гетерогенной библиотеки BLAS был также реализован тест High-Performance LINPACK (о тесте см. [14]). Подробнее о гетерогенных реализациях BLAS и HPL и полученных результатах смотри на форуме разработчиков AMD [6].

4. Модель гибридной системы

Предлагаемый группой гетерогенных вычислений подход к исследованию и оценке эффективности алгоритмов на гибридных установках:

1. Построить модель гибридной вычислительной системы, описываемую набором параметров.

2. Построить схему вычислений по алгоритму в модели. Для этого определить необходимый набор вычислительных ядер, другими словами, произвести декомпозицию алгоритма в виде элементарных вычислительных процедур. Схема вычислений представляет собой последовательность загрузок/выгрузок данных и вызовов ядер в нужной последовательности.

3. Получить формулу для оценки производительности вычислений по этой схеме, зависящую от параметров модели.

Подставляя в полученную формулу значения параметров для конкретной установки, получить предварительную оценку производительности вычислений. Это будет оценка сверху. В зависимости от полученной оценки можно принять решение о том, имеет смысл приступать к реализации алгоритма или нет. Возможно, оценку потребуется уточнить, усложнив модель установки или схему вычислений.

Будем рассматривать вычислительные системы следующей конфигурации: системная память, центральный процессор (CPU) и несколько сопроцессоров под его управлением (для простоты будем считать все сопроцессоры одинаковыми). Схема такой системы приводится на рис. 1. Доступ от сопроцессоров к системной памяти осуществляется по разделяемому каналу. Обмен данными между сопроцессорами возможен только через системную память. В роли центрального процессора могут выступать несколько универсальных процессоров, с собственными возможностями для параллельных вычислений. В качестве сопроцессоров могут использоваться графические ускорители общего назначения или другие виды сопроцессоров – например, Intel Xeon Phi. И GPU, и Xeon Phi можно определить, как массивно-параллельный ускоритель общего назначения, обладающий глобальной памятью и набором вычислителей, каждый – с собственной локальной памятью, поэтому в рамках данной работы мы будем рассматривать сопроцессоры такой архитектуры.

Параметры построенной модели:

– K – количество сопроцессоров;
– $HMEM$ – объем системной памяти;
– BW – пиковая скорость обмена данными между системной памятью и сопроцессорами;
– $DMEM$ – объем глобальной памяти одного сопроцессора;
– bw_g – пиковая скорость доступа к глобальной памяти на сопроцессоре;
– $DPEAK$ – пиковая производительность одного сопроцессора;
– $HPEAK$ – пиковая производительность центрального процессора.

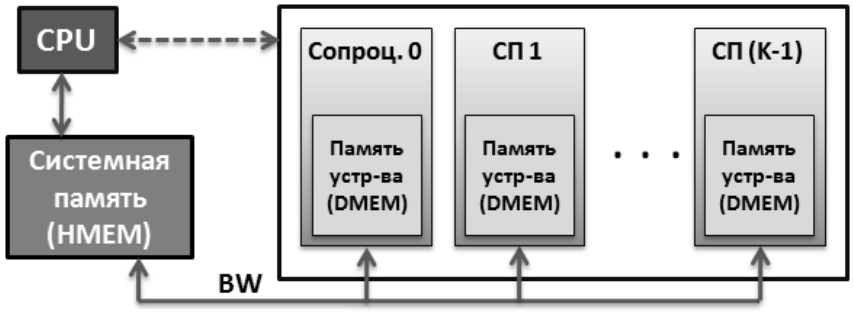

Рис. 1. Модель гибридной установки

Кроме того, ключевой параметр каждой задачи – $Perf_{kern}$ – производительность ядра на сопроцессоре. Чтобы получить его значение, можно либо написать собственное ядро и замерить его производительность, либо использовать готовое ядро (для широко известных алгоритмов), либо оценить производительность теоретически.

В табл. 1 приводятся значения параметров построенной модели для трех примеров гибридных установок: один из стендовых узлов НИИСИ РАН, гибридный кластер НИИСИ РАН из 4 узлов и один из 64 узлов суперЭВМ К100.

Таблица 1

Примеры гибридных установок: параметры модели

	Узел НИИСИ РАН	Кластер НИИСИ РАН	Узел суперЭВМ К100
CPU	Intel Xeon E5-2670		Intel Xeon X5670
GPU	NVidia GeForce GTX TITAN		NVidia Tesla C2050
Конфигурация	2x CPU + 4x GPU + PCI Express	4x [2x CPU + 2x GPU + PCI Express] + + InfiniBand	2x CPU + 3x GPU + PCI Express
K	4	8	3
HMEM	128 Гбайт	512 Гбайт	96 Гбайт
BW	32 Гбайт/с	6 Гбайт/с	16 Гбайт/с
DMEM	6 Гбайт		2,5 Гбайт
bw_g	288 Гбайт/с		144 Гбайт/с
DPEAK	1500 Гфлоп/с		515 Гфлоп/с
HPEAK	330 Гфлоп/с	1320 Гфлоп/с	140 Гфлоп/с

Чтобы написать эффективную гетерогенную процедуру, необходимо как минимум:

1) Разработать эффективные вычислительные ядра для сопроцессоров.

2) Организовать вычисления так, чтобы минимизировать количество пересылок данных между сопроцессорами и системной памятью.

3) Оставшиеся пересылки по возможности скрыть за вычислениями на сопроцессорах.

Предварительное исследование алгоритма позволяет установить, чем определяется производительность вычислений: производительностью ядер или скоростью передачи данных. Для удобства получения оценок в алгоритме по возможности выделяется одна или несколько базовых операций – относительно простых вычислительных процедур, которые занимают бóльшую часть общего времени вычислений (90 % или более). Далее теоретические оценки проводятся для базовых операций – если для них эффективная реализация возможна, то скорее всего она возможна и для алгоритма в целом.

5. NAS Parallel Benchmarks

Тесты, которые будут обсуждаться в данной статье, входят в состав NAS Parallel Benchmarks (Numerical Aerodynamics Simulation Parallel Benchmarks или NPB) – см. [15]. Это набор тестов, разработанных в центре NASA для оценки производительности параллельных суперкомпьютеров. Тесты представляют собой упрощенные базовые алгоритмы, на основе задач вычислительной гидродинамики, которые позволяют оценить характеристики вычислительной системы, существенные при решении реальных задач.

Все алгоритмы из NPB имеют так называемое «бумажное» описание: они полностью описаны в тексте. Выбор способа их реализации для конкретной системы остается за разработчиком, однако, на используемые средства накладывается ряд ограничений. В тексте также задаются входные данные и эталонные результаты для проверки корректности реализации алгоритмов. Все вычисления требуется производить с двойной точностью.

Классический состав NPB: 5 вычислительных «ядер» (базовых алгоритмов) и 3 «псевдоприложения» (более сложные задачи). Позднее к этому набору был добавлен ряд других тестов. Мы выбрали для исследования три базовых алгоритма:

– FT – дискретное преобразование Фурье;

– CG – метод сопряженных градиентов;

– MG – многосеточный метод.

В NPB вводится понятие классов задач – от S до F (последнего на настоящий момент). Различные классы соответствуют различным объемам входных данных, количествам итераций внутренних циклов в алгоритмах и т. п. С увеличением класса объем вычислительной работы возрастает экспоненциально. Соответственно, разные классы рассчитаны на вычислительные системы разных масштабов: S – маленькие задачи, для тестирования и отладки; W – задачи для рабочих станций 1990-х годов; A, B, C – стандартные размеры задач; D, E, F – масштабные задачи.

Имеются готовые реализации алгоритмов, доступные на сайте NPB [16], последняя редакция – NPB 3.3.1. Есть различные версии кодов: в основном на языке FORTRAN, частично на C; последовательная версия и параллельные с использованием MPI и OpenMP; версии на Java и HPF, и др. Коды представляют собой образец для облегчения работы программисту; никакая совместимость с ними не требуется.

6. NPB FT: преобразование Фурье

Первый алгоритм, который мы рассмотрим – FT, дискретное преобразование Фурье (ДПФ). Данное преобразование является основой всех спектральных алгоритмов. Задача тестирует передачи данных на больших дистанциях.

В формулировке задачи требуется решить уравнение в частных производных

$$\frac{\partial u(x,t)}{\partial t} = \alpha \nabla^2 u(x,t), \quad x \in \mathbb{R}^3, \tag{1}$$

с помощью дискретного преобразования Фурье. Базовой операцией здесь является трехмерное комплексное быстрое преобразование Фурье (БПФ), каждая размерность – степень 2.

В одномерном случае БПФ комплексного вектора длины $N = 2^n$ может быть вычислено по известному алгоритму Кули – Тьюки [17]. Имеется множество готовых реализаций этого алгоритма, в том числе для графических ускорителей. В двумерном случае дана матрица порядка $N_1 \times N_2$, и алгоритм БПФ состоит из двух шагов: на первом шаге вычисляется одномерное БПФ длины N_1 от каждого столбца матрицы, а на втором вычисляется одномерное БПФ длины N_2 от каждой строки полученной промежуточной матрицы. В трехмерном случае дан массив порядка $N_1 \times N_2 \times N_3$ и требуется вычислить, соответственно, $N_2 \times N_3$ БПФ длины N_1, $N_1 \times N_3$ БПФ длины N_2 и $N_1 \times N_2$ БПФ длины N_3.

Опишем схему вычислений в двумерном случае. Входные и выходные данные располагаются в системной памяти. Пусть размер задачи таков, что памяти одного сопроцессора не достаточно для всего необходимого объема данных, однако, достаточно для обработки векторов длины N_1 и N_2. На первом шаге весь набор столбцов делится на группы – страницы – по p_1 векторов. Каждая страница загружается в память одного сопроцессора, далее на всех сопроцессорах параллельно вызывается ядро БПФ длины N_1 для каждого вектора страницы, после чего страницы результатов со всех сопроцессоров выгружаются во временный массив в системной памяти. Второй шаг аналогичен первому: набор строк полученного временного массива разделяется на страницы по p_2 векторов длины N_2, каждая страница обрабатывается на своем сопроцессоре, и окончательный результат собирается в системной памяти. Описанная схема является общепринятой при вычислении двумерного БПФ. Для трехмерного БПФ схема вычислений аналогична: добавляется еще один шаг алгоритма.

Чтобы совместить вычисления с пересылками данных, в основном цикле на каждом шаге алгоритма обработка одной страницы на сопроцессорах совмещается с выгрузкой предыдущей страницы результатов и загрузкой следующей страницы входных данных. При такой организации вы-

числений общее время работы определяется максимальным из двух: суммарным временем выгрузки и загрузки или временем вычислений над одной страницей. Время на выгрузку и загрузку K векторов длины N, по одному на каждый сопроцессор, выражается через параметры построенной модели по формуле

$$T_{L+S} = \frac{2KN}{BW}. \quad (2)$$

Это оценка фактического времени снизу, в предположении, что данные передаются с максимальной возможной скоростью. Количество арифметических операций на вычисление БПФ одного вектора составляет $OP_{CP} = 5N\log N$ (здесь log – логарифм по основанию 2). Отсюда, время на вычисления в ядре с одним вектором, параллельно на K сопроцессорах, оценивается по формуле

$$T_C = \frac{5N\log N}{Perf_{kern}}. \quad (3)$$

В случае если $T_{L+S} > T_C$, общее время работы алгоритма (T) определяется временем пересылок данных и может быть оценено как время пересылки всего массива 3 раза на сопроцессоры и обратно:

$$T_{FT} \approx \frac{6N_1N_2N_3}{BW}. \quad (4)$$

Производительность вычислений по алгоритму оценивается (сверху) по формуле $PERF_{FT} \approx \approx OP_{ALL}/T_{FT}$, где $OP_{ALL} = 5N_1N_2N_3\log(N_1N_2N_3)$ – общее количество арифметических операций.

Рассмотрим в качестве примера гибридный вычислительный узел с четырьмя GPU GeForce TITAN и задачу класса C. Для этого класса $N_1 = N_2 = N_3 = 512$. Производительность ядра можно приблизительно оценить как 10 % от пиковой производительности устройства (такова эффективность большинства существующих в свободном доступе реализаций одномерного БПФ): $Perf_{kern} \approx 0{,}1$, $DPEAK$ = 150 Гфлоп/с. Подставляя все значения параметров узла в формулы (2) и (3), получаем, что время на пересылки на порядок больше, чем время на вычисления. Отсюда, общее время работы оценивается по формуле (4): $T_{FT} \approx 0{,}2013$ с, откуда $PERF_{FT} \approx 90{,}01$ Гфлоп/с, причем эта оценка не зависит от количества ускорителей. Для сравнения, если бы тот же алгоритм выполнялся на CPU, мы могли бы ожидать производительность порядка 60 Гфлоп/с (см., например, [18]). Таким образом, использование GPU в расчетах трехмерного БПФ не дало бы заметного выигрыша в производительности.

Данная ситуация является типичной для гибридных установок с GPU в целом: «узким местом» является доступ к системной памяти через PCI Express. Для алгоритмов, основанных на 2- и 3-мерном БПФ в чистом виде, этот фактор оказывается критическим, поэтому реализация таких алгоритмов на установках рассматриваемого типа не оправданна. Отметим, что отечественный микропроцессор КОМДИВ128-РИО в этом смысле более сбалансирован: в нем канал доступа к памяти обеспечивает загрузку и выгрузку данных со скоростью, достаточной для выполнения вычислений, что позволяет эффективно реализовать алгоритм БПФ. Для алгоритма FT реализация на OpenCL в НИИСИ РАН не разрабатывалась.

7. NPB CG: метод сопряженных градиентов

Второй алгоритм, который мы рассмотрим – CG, метод сопряженных градиентов. Задача тестирует неструктурированные вычисления и нерегулярный доступ к памяти. Операция SpMV (умножение разреженной матрицы на вектор), к которой сводится данный алгоритм, является ключевой во многих алгоритмах из различных областей естествознания.

Формулировка задачи: дана симметричная положительно определенная разреженная матрица A порядка $n \times n$; найти ее наибольшее собственное значение.

В NPB задается процедура генерации матрицы на языке FORTRAN, а также порядок для каждого класса, порядок матрицы n и другие параметры. Алгоритм решения задачи, описанный в текс-

те, – метод обратных степеней, который сводится к решению СЛАУ с разреженной матрицей методом сопряженных градиентов, а он, в свою очередь, – к операции SpMV. Таким образом, базовой операцией алгоритма, самой затратной по времени и ресурсам, является SpMV; замеры, проведенные нами для референсных кодов, показали, что SpMV занимает порядка 95 % от общего времени вычислений по алгоритму.

В процедуре генерации из NPB матрица генерируется в формате CSR, однако допускается перед началом вычислений переупаковать ее в любой другой формат – время на переупаковку при замерах времени вычислений не учитывается. Мы выбрали формат Sliced ELLpack, предложенный в [19]. В этом формате матрица разделяется на «слои» – группы по H строк (H – параметр формата).

До начала исследования алгоритмов из NPB в НИИСИ РАН была разработана процедура SpMV на OpenCL для GPU. Для упаковки матрицы использовался формат SlELL. Производительность процедуры существенно зависит от портрета входной матрицы. Для тестирования процедуры использовался стандартный набор из 14 матриц разного типа, полученных из различных практических задач (см., например, [20]). На рис. 2 приводится диаграмма с результатами замеров производительности разработанного на OpenCL ядра на различных GPU и, для сравнения, производительности библиотечной процедуры SpMV на CPU от Intel.

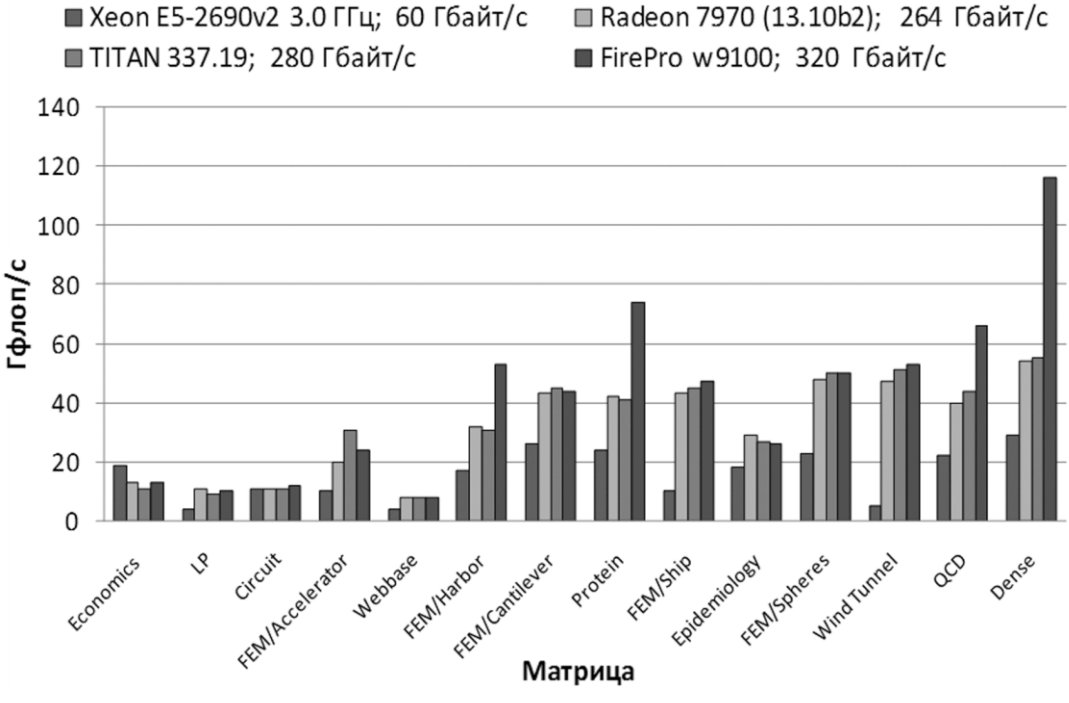

Рис. 2. Ядро процедуры SpMV на OpenCL

Опишем схему вычислений по алгоритму CG в построенной модели. Пусть суммарного объема памяти сопроцессоров достаточно для умножения всей матрицы на вектор. Каждый сопроцессор умножает на входной вектор некоторое количество слоев матрицы и выдает в результате фрагмент выходного вектора. При этом, поскольку во всем алгоритме используется одна и та же матрица, ее слои один раз загружаются в память сопроцессоров перед началом вычислений. Упрощенно можно считать, что весь алгоритм – цепочка SpMV с одной и той же матрицей, и на каждом шаге выходной вектор является входным для следующего шага: $y_i = Ax_i$, $x_{i+1} = y_i$. При этом после каждого SpMV требуется произвести следующий обмен данными: выгрузить фрагменты результата с сопроцессоров и затем загрузить полученный входной вектор для следующего SpMV в память каждого сопроцессора.

Совмещение вычислений с пересылками может быть организовано следующим образом. На каждом сопроцессоре входной вектор умножается на каждый слой по отдельности; обработка одного

слоя совмещается с выгрузкой с каждого сопроцессора предыдущего фрагмента результата и последующей загрузкой всех K выгруженных фрагментов на каждый сопроцессор. При такой организации работы общее время вычислений будет определяться максимальным из двух: временем вычислений в ядре с одним слоем (T_C) или временем пересылки $K+1$ фрагмента вектора-результата (T_{L+S}).

Обозначим за S общее количество слоев, NZ – длину массива значений в упакованной матрице (приблизительно равна количеству ненулевых элементов в исходной матрице). На вычисление SpMV для всей матрицы требуется $2NZ$ арифметических операций. Отсюда получаем следующие формулы для оценки T_{L+S} и T_C:

$$T_{L+S} \approx \frac{(K+1)H}{BW}, \quad T_C \approx \frac{2NZ}{S \cdot Perf_{kern}}. \tag{5}$$

Если $T_{L+S} \geq T_C$, то время вычислений по алгоритму оценивается как

$$T_{CG} \approx \frac{(K+1)n}{BW}, \tag{6}$$

если же $T_{L+S} \leq T_C$, то время вычислений оценивается как

$$T_{CG} \approx \frac{2NZ}{K \cdot Perf_{kern}}. \tag{7}$$

Производительность вычислений по алгоритму – $PERF_{CG} \approx 2NZ/T$.

В отличие от алгоритма БПФ, для современных установок с GPU время вычислений оказывается больше, чем время пересылок. Это означает, что производительность вычислений оценивается как $PERF_{CG} \approx K \cdot Perf_{kern}$, т. е. определяется производительностью ядра и линейно масштабируется с ростом количества сопроцессоров.

Данная оценка является лишь ориентировочной оценкой сверху – в предположении, что вычисления и пересылки выполняются с максимальной производительностью, и никаких накладных расходов нет. Кроме того, оценка получена для базовой операции, а не для алгоритма в целом. Можно ожидать, что реальная производительность составит 60–70 % от этого теоретического максимума – если вычисления полностью совмещаются с пересылками, и нет дополнительных затратных операций на CPU, таких как MPI-обмены.

В качестве примера получим теоретические оценки для гибридного узла с четырьмя GPU GeForce TITAN. На классе C, при выборе $H = 64$, его производительность, разработанного в НИИСИ РАН ядра SpMV на одном GPU, составила примерно 11,57 Гфлоп/с – возьмем эту величину в качестве $Perf_{kern}$. Подставляя значения всех остальных параметров в (5), получим, что время вычислений на несколько порядков превышает время пересылок. Та же картина имеет место и для других классов задачи. Значит, производительность вычислений теоретически оценивается как $4Perf_{kern}$. В табл. 2 эта теоретическая оценка сравнивается с лучшими результатами, полученными в результате замеров референсных кодов на CPU.

Таблица 2

CG: сравнение оценки для GPU с результатами на CPU

Класс задачи	Теоретическая оценка для всей установки (4 x TITAN)	Референсный код на CPU (2 x Xeon E5-2690v2)
S	46280 Мфлоп/с	2260 Мфлоп/с
W		13500 Мфлоп/с
A		26500 Мфлоп/с
B		10100 Мфлоп/с
C		9900 Мфлоп/с

Мы видим, что реализация метода CG на гибридных архитектурах оправданна. Преимущество особенно заметно для больших классов, для которых на CPU задача перестает помещаться в кэш, и производительность вычислений падает.

Узким местом реализации на GPU является производительность вычислений в ядре, которая определяется производительностью подсистемы памяти: при умножении разреженной матрицы, в том числе в формате SlELL, на вектор элементы входного вектора считываются из глобальной памяти в случайном порядке; такая схема доступа является неэффективной. В НИИСИ РАН был разработан новый формат упаковки матрицы, который был назван Framed ELLpack (FrELL). Для матрицы в этом формате в ходе вычислений в ядре доступ к входным данным остается случайным, но не в глобальной памяти, а в локальной, поэтому происходит быстрее. Минусом данного формата является большее количество дополнительных нулей, по сравнению с SlELL.

Поскольку теоретическая оценка производительности алгоритма CG показала, что его реализация на гибридной установке имеет смысл, группой гетерогенных вычислений была разработана реализация данного алгоритма на OpenCL. При этом использовался следующий простой прием: «подмена» частей референсного кода NPB на FORTRAN вызовами процедур на C для планировщика, которые организуют загрузки и выгрузки данных на сопроцессорах и вызов ядер в необходимой последовательности.

На первом этапе весь алгоритм был реализован на одном GPU, для небольших классов задач. Замеры производительности, проведенные для этой реализации на различных GPU, показали, что, начиная с класса B, производительность вычислений на GPU в среднем в 1,5–2 раза выше, чем производительность референсной OpenMP-процедуры на CPU.

На втором этапе алгоритм был реализован для нескольких сопроцессоров на одном узле. Также была добавлена возможность обрабатывать часть матрицы на CPU – таким образом, была получена гетерогенная процедура, которая задействует гибридный узел целиком.

Кроме того, было написано ядро для матрицы в формате FrELL и соответствующая версия процедуры SpMV для планировщика. Чтобы получить максимально возможную производительность полученной реализации, для различных GPU были выбраны оптимальные параметры для обоих форматов упаковки матрицы, путем полного перебора по всем допустимым значениям. Результаты замеров показали, что новый формат FrELL дает некоторый выигрыш производительности, наиболее существенный на задаче класса A. В среднем выигрыш невелик: в данном формате приходится выполнять большее количество избыточных вычислений с дополнительными нулями, чем в формате SlELL. Для последних GPU от AMD, в частности, FirePro, выигрыша нет, поскольку в последнее время велись разработки по аппаратному ускорению случайного доступа к глобальной памяти.

На рис. 3 приводятся два графика с результатами замеров производительности полученной гетерогенной реализации алгоритма CG, для двух видов GPU и различного их количества. Для GeForce TITAN в каждом случае представлены лучшие из пар результатов – для матрицы в формате SlELL и FrELL; для FirePro w9100 все результаты – для матрицы в формате SlELL.

Из графиков следует, что использование GPU оправданно при обработке больших объемов данных; маленьких объемов недостаточно, чтобы задействовать все ресурсы и возможности параллелизма графического ускорителя.

На больших классах задач реализация НИИСИ РАН показывает хорошую масштабируемость: например, на классе D, на GeForce TITAN, масштабируемость близка к линейной, с коэффициентом 0,8. Помимо масштабируемости, реализация показывает заметно бóльшую производительность вычислений, чем референсный код на CPU:

– ~5400 Мфлоп/с на двух современных CPU с OpenMP;

– 28500 Мфлоп/с на восьми современных GPU.

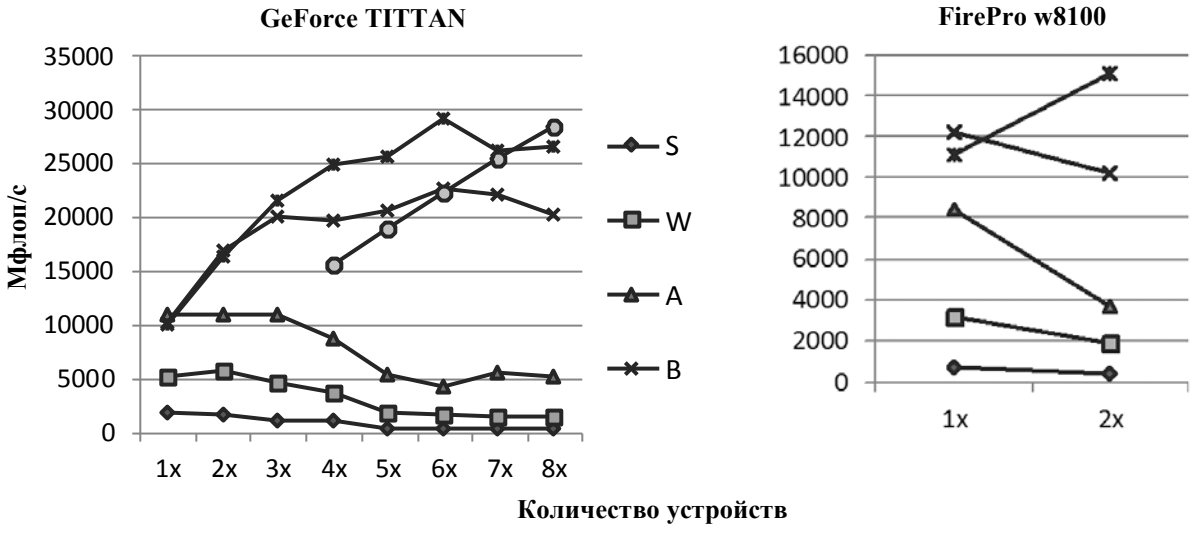

Рис. 3. Гетерогенная процедура CG: масштабируемость на узле

Также были проведены замеры для гетерогенной процедуры, использующей совместно CPU (два Xeon E5-2660) и GPU (четыре AMD Radeon 7970). Замеры показали, что на маленьких классах задач разделение работы между CPU и GPU невыгодно, так как накладные расходы отнимают существенную долю от общего времени вычислений. На больших классах делегирование части работы CPU дает дополнительный выигрыш в скорости вычислений – до 35 %. Естественным образом, этот выигрыш тем больше, чем меньше сопроцессоров используется. Гетерогенная процедура, запущенная только на CPU, показала ту же производительность, что и референсная реализация.

Результаты, полученные для гетерогенной реализации алгоритма NPB CG, подтверждают теоретический прогноз: использование GPU в вычислениях по данному алгоритму оправданно. В настоящий момент группа гетерогенных вычислений продолжает работу над реализацией алгоритма CG.

8. NPB MG: многосеточный метод

Третий алгоритм – MG, многосеточный метод. Задача тестирует регулярную передачу данных на маленьких и больших дистанциях. Многосеточный метод в настоящее время широко применяется для решения многих задач, в частности, для предобуславливания СЛАУ с разреженными матрицами. В NPB рассматривается простая версия алгоритма: классический геометрический многосеточный метод.

Формулировка задачи: найти приближенное решение уравнения Пуассона $\nabla^2 u = v$ на сетке $N \times N \times N$, с периодическими граничными условиями. Алгоритм решения полностью описан в тексте NPB. Начальное приближение – нулевое. Для каждого класса задается размер сетки N и другие параметры задачи.

Алгоритм, в сущности, представляет собой последовательность трилинейных операторов на сетке (далее Tlin). Tlin переводит трехмерный массив в трехмерный массив, каждое новое значение в точке получается в общем случае как сумма 27 соседних старых значений с коэффициентами. При вычислении значений на границе требуется учитывать периодические граничные условия: $u_N = u_1$, $u_0 = u_{N-1}$ – соответственно, на входе необходимы значения с противоположной грани обрабатываемого трехмерного массива.

Замеры для референсных кодов показали, что ~90 % общего времени вычислений по алгоритму занимает вычисление невязки на самой мелкой сетке, то есть для массива порядка $N \times N \times N$. Поэтому в качестве базовой операции мы рассмотрим упрощенный Tlin на самой мелкой сетке: все

коэффициенты ненулевые, и сетка остается неизменной. Опишем схему вычислений для последовательности таких Tlin, где выходной массив на каждом шаге является входным для следующего шага.

При обработке на нескольких сопроцессорах массив делится на блоки, каждый сопроцессор обрабатывает свой блок: вычисляет Tlin для его точек. При этом для вычисления значений в граничных точках блока требуется иметь на каждом сопроцессоре 6 дополнительных граничных плоскостей (по одной для каждой грани блока. Ядро на сопроцессоре получает на вход блок с дополнительными границами и вычисляет Tlin, давая на выходе новый блок. Перед вычислением следующего Tlin в ходе алгоритма необходимо произвести обмен граничными плоскостями с соседними сопроцессорами.

Для совмещения вычислений в ядре с пересылками работа разделяется на два этапа: вычисление граничных точек блока (за время T_{CB}) и вычисление внутренних точек (T_{CI}), одновременно с обменом граничными плоскостями с другими сопроцессорами (T_{SB}). Время работы процедуры MG определяется максимальным из двух: T_{CI} или T_{SB}. Пусть размер одного блока – $N_1 \times N_2 \times N_3$, тогда количество граничных точек для каждого блока составляет по порядку $2(N_1N_2 + N_2N_3 + N_1N_3)$. На вычисление значения в каждой точке требуется 30 арифметических операций. Отсюда получаются следующие формулы для оценки времени:

$$T_{SB} = \frac{4K(N_1N_2 + N_2N_3 + N_1N_3)}{BW}, \quad T_{CI} = \frac{30N_1N_2N_3}{Perf_{kern}}. \tag{8}$$

Если для конкретной установки $T_{SB} > T_{CI}$, то общее время вычислений по алгоритму оценивается по формуле

$$T_{MG} \approx T_{CB} + T_{SB} = \left(\frac{60}{Perf_{kern}} + \frac{4K}{BW}\right)(N_1N_2 + N_2N_3 + N_1N_3). \tag{9}$$

Если же $T_{SB} \le T_{CI}$, то

$$T_{MG} \approx \frac{30N^3}{K \cdot Perf_{kern}}. \tag{10}$$

Как правило, для гибридных установок рассматриваемого типа имеет место вторая ситуация. В этом случае производительность вычислений по алгоритму определяется производительностью ядра и оценивается по формуле $PERF_{CG} \approx K \cdot Perf_{kern}$.

Получим оценки для гибридного узла с четырьмя GeForce TITAN. В НИИСИ РАН было разработано ядро на OpenCL для вычисления Tlin на одном ускорителе. Его производительность на классе B составила примерно 128,42 Гфлоп/с – примем эту величину за $Perf_{kern}$. Из формул (8) получаем для всех классов, что $T_{SB} \le T_{CI}$. Значит, производительность вычислений теоретически оценивается как $4Perf_{kern}$. В табл. 3 эта теоретическая оценка сравнивается с лучшими результатами, полученными при замерах референсных OpenMP-кодов на CPU.

Таблица 3

MG: сравнение оценки для GPU с результатами на CPU

Класс задачи	Теоретическая оценка для всей установки (4 x TITAN)	Референсный код на CPU (2 x Xeon E5-2690v2)
S	513680 Мфлоп/с	4300 Мфлоп/с
W		18150 Мфлоп/с
A		19100 Мфлоп/с
B		19500 Мфлоп/с
C		17000 Мфлоп/с

Отсюда можно сделать вывод, что реализация метода MG на гибридных архитектурах оправданна и обещает многократно бóльшую производительность, чем процедура на CPU. Для алгоритма MG группой гетерогенных вычислений также была разработана реализация на OpenCL.

Сначала, как и для алгоритма CG, была реализована процедура для одного ускорителя. Замеры производительности этой процедуры на различных GPU показали, что уже на классе W производительность вычислений на GPU сопоставима с производительностью на CPU, а для бóльших классов использование GPU дает преимущество в 4–5 раз.

От реализации на K сопроцессорах можно ожидать, в лучшем случае, производительность в K раз больше, чем на одном таком же сопроцессоре. Вычислительные ядра Tlin в ходе дальнейших реализаций не претерпели существенных изменений: все изменения касались оптимизации обработки границ и обменов данными.

Реализация алгоритма для нескольких сопроцессоров разрабатывалась в несколько этапов. Сначала была реализована синхронная процедура, осуществляющая вычисления, загрузки/выгрузки данных и MPI-обмены последовательно. Затем было организовано разбиение массива на части, что позволило совмещать вычисления с пересылками и обменами: процедура для планировщика формирует граф зависимостей между различными операциям, а само совмещение осуществляется аппаратными средствами. Также были разработаны специальные ядра для более эффективной обработки границ. Наконец, для получения максимальной производительности, для каждого сопроцессора, ядра Tlin и размера массива были подобраны оптимальные параметры ядра (количество рабочих групп, нитей и т. п.).

Чтобы выяснить, насколько падает производительность при добавлении к вычислениям в ядре пересылок данных и MPI-обменов, полученная гетерогенная процедура была запущена на одном сопроцессоре. Замеры показали, что в сумме время на пересылки и MPI-обмены приблизительно равно времени вычислений в ядре, и это время не полностью скрывается за вычислениями. Для GeForce TITAN потери составили порядка 20 %. Для FirePro w9100 потери выше – в среднем около 50 %.

Гетерогенная процедура MG может выполняться как на одном узле, так и на распределенной установке. На рис. 4 приводятся два графика с результатами замеров производительности полученной гетерогенной реализации алгоритма MG, на одном узле, для двух видов GPU и различного их количества.

Реализация показывает хорошую масштабируемость: линейную для класса C, с коэффициентом примерно 0,5. Также очевиден и абсолютный выигрыш от использования GPU, по сравнению с референсной реализацией – например, на классе C:

– ~17000 Мфлоп/с на двух современных CPU с OpenMP;

– 161700 Мфлоп/с на восьми современных GPU.

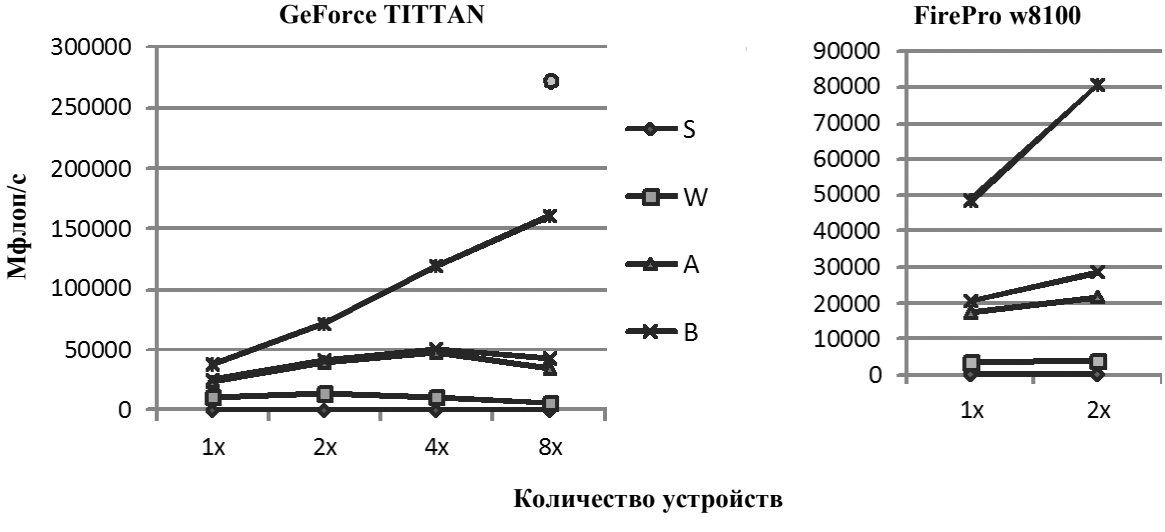

Рис. 4. Гетерогенная процедура MG: масштабируемость на узле

Диаграмма на рис. 5 показывает результаты запусков гетерогенной процедуры на мини-суперЭВМ НИИСИ РАН – вычислительном кластере из нескольких узлов. Здесь «np» – число процессов в терминологии MPI, совпадает с общим количеством сопроцессоров на всех узлах. С увеличением количества узлов при постоянном общем количестве сопроцессоров возрастает время на передачи данных, однако, сокращается время на дополнительные операции на CPU. В зависимости от общего количества сопроцессоров, тот или иной фактор является определяющим, однако, в целом производительность на распределенной установке незначительно отличается от производительности на одном узле. На классе D производительность составляет:

– ~18 Гфлоп/с на двух современных CPU с OpenMP;

– до 276 Гфлоп/с на восьми современных GPU.

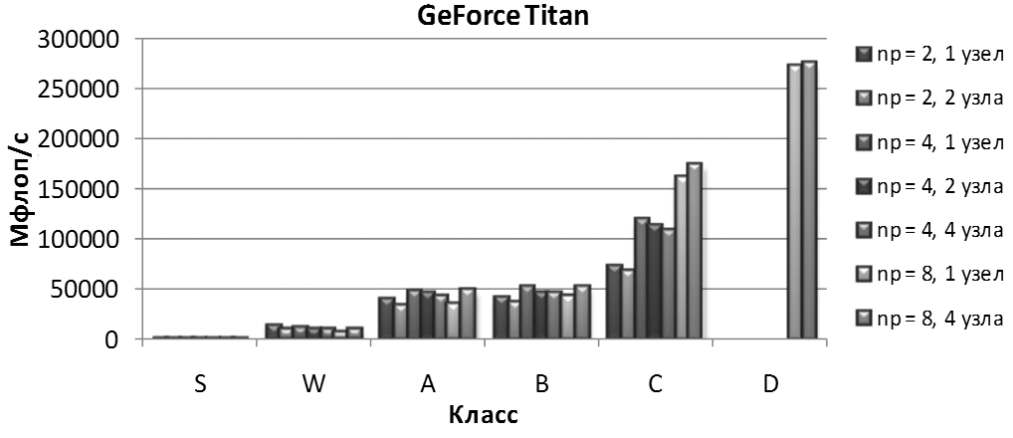

Рис. 5. Гетерогенная процедура MG: результаты на вычислительном кластере НИИСИ

Чтобы показать масштабируемость задачи на большее количество сопроцессоров, приведем на рис. 6 результаты запусков на суперЭВМ К100. Здесь точное значение зависит от взаимного расположения узлов, на которых исполняется задача, поэтому приводятся средние значения по 10 запускам. Параметр «ppn» – в терминологии MPI, количество сопроцессоров на одном узле. Задача класса D масштабируется до 64 ускорителей:

– 342 Гфлоп/с на 16 GPU;

– 1,035 Тфлоп/с на 64 GPU.

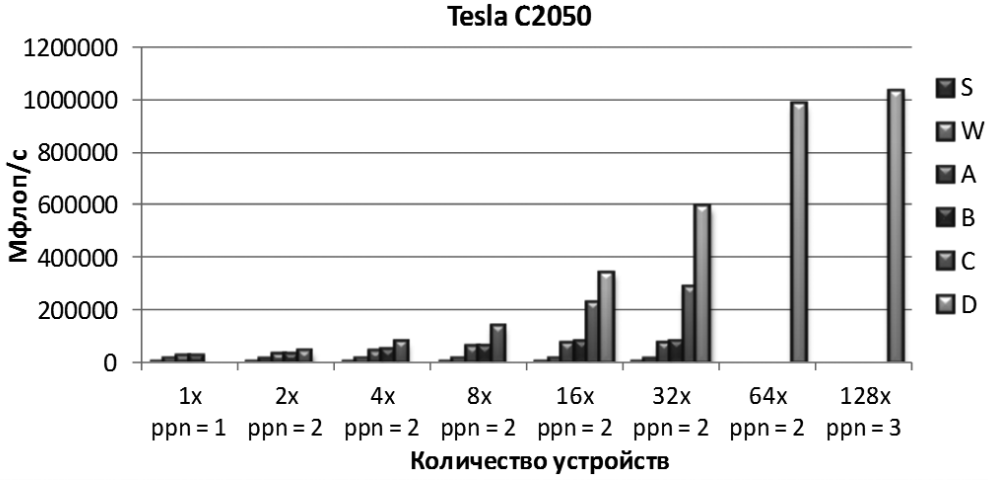

Рис. 6. Гетерогенная процедура MG: результаты на К100

В целом, по результатам всех замеров, имеют место две закономерности:

1. Для задачи каждого класса существует пиковое количество сопроцессоров (зависящее от конкретного вида сопроцессоров), до которого производительность вычислений возрастает линейно, а при дальнейшем увеличении числа сопроцессоров роста практически нет. Чем больше класс задачи, тем больше пиковое количество. Это объясняется тем, что необходим некоторый минимальный объем данных, чтобы задействовать все вычислительные мощности каждого ускорителя.

2. На одном и том же количестве ускорителей производительность возрастает с ростом класса задачи. Причина та же, что и в первом случае.

Производительность гетерогенной реализации алгоритма была также замеряна на CPU. Она оказалась ниже, чем для референсного кода. Это означает, что переносимая гетерогенная реализация алгоритма возможна, но требует доводки под конкретную архитектуру.

Результаты, полученные для гетерогенной реализации НИИСИ РАН, подтверждают теоретический прогноз: использование GPU в вычислениях по алгоритму MG оправданно.

9. Выводы

Мы рассмотрели гибридные установки общего вида и реализацию на них, теоретическую и реальную, для трех алгоритмов из NPB. Для этих алгоритмов:

– FT – гетерогенная реализация неэффективна;

– CG – гетерогенная реализация оправданна, дает заметный выигрыш в производительности для больших классов задач; узкое место – ядро SpMV;

– MG – гетерогенная реализация оправданна, дает значительный выигрыш в производительности.

По результатам проделанных исследований, реализаций и замеров можно сделать следующие выводы:

I. Использование гибридных установок выгодно для широкого класса задач естествознания.

II. Для большинства алгоритмов производительность вычислений определяется производительностью подсистемы памяти.

III. Необходимое условие высокой эффективности – сокрытие пересылок данных за вычислениями.

IV. Персональные мини-суперЭВМ являются неплохой альтернативой совместному использованию больших суперЭВМ.

V. Переносимые реализации требуют «доводки» под конкретную аппаратуру.

Литература

1. Khronos OpenCL Working Group. The OpenCL Specification: version 1.2. 14.11.2011. [Electronic resource]. URL Mode of access: http://www.khronos.org/registry/cl/

2. Институт Прикладной Математики им. М. В. Келдыша РАН. Web-сервер. [Электронный ресурс]. URL Режим доступа: http://www.kiam.ru

3. Богданов П. Б., Ефремов А. А., Горобец А. В., Суков С. А. Применение планировщика для эффективного обмена данными на суперкомпьютерах гибридной архитектуры с массивно-параллельными ускорителями // Выч. мет. программирование. 2013. Т. 14. Вып. 4. С. 122–134.

4. Богданов П. Б., Горобец А. В., Суков С. А. Адаптация и оптимизация базовых операций газодинамического алгоритма на неструктурированных сетках для расчетов на массивно-параллельных ускорителях // Журнал выч. мат. и мат. физ. 2013. Т. 53, № 8. С. 1360–1373.

5. Горобец А. В., Суков С. А., Железняков А. О. и др. Расширение двухуровневого распараллеливания MPI + OpenMP посредством OpenCL для газодинамических расчетов на гетерогенных системах // Вестн. ЮУрГУ. Сер. Матем. моделирование и программирование. 2011. № 9. С. 76–86.

6. Efremov A. A., Bogdanov P. B. Programming infrastructure for heterogeneous computing and its applications // AMD DevGurus OpenCL Forum. 2012. [Electronic resource]. URL Mode of access: http://devgurus.amd.com/thread/159457.

7. Gorobets A. V., Soukov S. A., Bogdanov P. B. et al. Extension with OpenCL of the two-level MPI + OpenMP parallelization for large-scale CFD simulations on heterogeneous systems // Parallel Computational Fluid Dynamics (ParCFD). Barcelona, Spain. 2011.

8. Gorobets A. V., Soukov S. A., Efremov A. A., Bogdanov P. B. OpenCL GPU-oriented implementation for basic operations of CFD algorithms on unstructured meshes // Super-Computation and Computer Simulation (SCCS). Sarov, Russia. 2012.

9. Efremov A. A., Bogdanov P. B. Programming Infrastructure for Heterogeneous Computing Based on OpenCL and its Applications // Exascale Application and Software Conference (EASC). Edinburg, Scotland, UK. 2013.

10. Gorobets A. V., Soukov S. A., Bogdanov P. B. OpenCL implementation of basic operations for a high-order finite-volume polynomial scheme on unstructured hybrid meshes // Parallel Computational Fluid Dynamics (ParCFD). Changsha, Hunan, China. 2013.

11. Sudareva O. U., Efremov A. A., Bogdanov P. B. Heterogeneous programming methodology based on OpenCL framework // High Performance Computing (HPC-UA). Kyiv, Ukraine. 2013.

12. BLAS (Basic Linear Algebra Subprograms). [Electronic resource]. URL Mode of access: http://www.netlib.org/blas

13. Dongarra J., Dong T., Gates M. et al. MAGMA: a New Generation of Linear Algebra Libraries for GPU and Multicore Architectures: MAGMA 1.4 Release. Knoxville: University of Tennessee, 08.14.2013.

14. Petitet A., Whaley R. C., Dongarra J., A. Cleary. HPL – A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers: version 2.0. 10.09.2008. [Electronic resource]. URL Mode of access: http://www.netlib.org/benchmark/hpl/

15. Bailey D., Barszcz E., Barton J. et al. The NAS Parallel Benchmarks: RNR Technical Report RNR-94-007, March 1994.

16. NAS Parallel Benchmarks. [Electronic resource]. URL Mode of access: https://www.nas.nasa.gov/publications/npb.html

17. Cooley J. W., Tukey J. W. An algorithm for the machine calculation of complex Fourier series // Mathematics of Computation. 1965. Vol. 19, N 90. P. 297–301.

18. Takahashi D. An Implementation of Parallel 2-D FFT Using Intel AVX Instructions on Multi-core Processors. Algorithms and Architectures for Parallel Processing // Lecture Notes in Computer Science. 2012. Vol. 744. P. 197–205.

19. Monakov A., Lokhmotov A., Avetisyan A. Automatically tuning sparse matrix-vector multiplication for GPU architectures // High Performance Embedded Architectures and Compilers (HiPEAC). Pisa, Italy. January 25–27, 2010. P. 111–125.

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

ПРЯМОЕ ОДНОМЕРНОЕ ГАЗОДИНАМИЧЕСКОЕ МОДЕЛИРОВАНИЕ РАСПРОСТРАНЕНИЯ ВОЛН В ПЕРИОДИЧЕСКИХ ДВУХСЛОЙНЫХ СРЕДАХ И ВОЛНОВЫЕ УРАВНЕНИЯ С ДИСПЕРСИЕЙ

Ю. А. Бондаренко, В. Н. Софронов

Российский федеральный ядерный центр –
Всероссийский НИИ экспериментальной физики, г. Саров

Введение

Интерес к исследованию композитных материалов разных типов связан с их новыми полезными свойствами, которых можно добиться оптимальным выбором компонент и их объемного содержания. Например, в композитных материалах с помощью выбора частиц высокой прочности под-