УДК 004.942

РЕАЛИЗАЦИЯ МЕТОДА РЕШЕНИЯ ДВУМЕРНОГО УРАВНЕНИЯ ТЕПЛОПРОВОДНОСТИ НА ГИБРИДНОЙ АРХИТЕКТУРЕ (CPU+GPU)

В. О. Анисов, Э. М. Вазиев, Д. А. Ушаков*
(ФГУП "РФЯЦ-ВНИИТФ им. академ. Е. И. Забабахина", г. Снежинск Челябинской обл.)

Представлен один из способов организации совместных вычислений на гибридной архитектуре: универсальный процессор (CPU) и графический ускоритель (GPU). На данной архитектуре с использованием двухуровневого распараллеливания реализован численный метод для решения двумерного уравнения теплопроводности. Вычисления на распределенной памяти (MPI) организованы с использованием геометрической декомпозиции. Для вычислений на общей памяти используется OpenMP на универсальном процессоре, CUDA — на графическом ускорителе. Распределение объема вычислений между CPU и GPU основано на оценке возможностей подсистем памяти используемых устройств, так как их пропускная способность является основным фактором, ограничивающим скорость вычислений. При балансировке вычислений также учитываются накладные расходы на организацию вычислений. Особое внимание уделено реализации метода сопряженных градиентов, так как большую часть времени решения уравнения теплопроводности занимает решение систем линейных уравнений.

Ключевые слова: гибридные вычисления, теплопроводность, метод сопряженных градиентов.

Введение

Современные тенденции построения суперкомпьютеров связаны с гибридными вычислительными архитектурами. Так, в ноябрьской редакции "Top500" 2019 года [1] шесть суперкомпьютеров из десяти первых строк рейтинга построены на гибридной архитектуре, т. е. вместе с центральным процессором (CPU) на вычислительном узле также присутствует сопроцессор, который призван ускорить вычисления. Зачастую в качестве сопроцессоров выступают графические ускорители (GPU), которые содержат тысячи вычислительных ядер. Следовательно, используемые алгоритмы должны хорошо работать на таких устройствах, иметь простую логику и хорошо распараллеливаться.

При моделировании на суперкомпьютерах процесса распространения тепла его можно описать уравнением теплопроводности. В случае его численного решения с использованием неявной разностной схемы уравнение теплопроводности сводится к решению систем линейных алгебраических уравнений (СЛАУ) вида $A\mathbf{x} = \mathbf{b}$. Одним из способов решения такой системы является итерационный метод сопряженных градиентов [2].

В рамках данной работы рассмотрена консервативная конечно-объемная схема. Описан метод сопряженных градиентов, приведены две версии (MPI+OpenMP и MPI+CUDA) его параллельной реализации на распределенной памяти (MPI) и общей памяти (OpenMP или CUDA). Распараллеливание на распределенной памяти выполнено с использованием геометрической декомпозиции

*Ушаков Денис Александрович, научный сотрудник,
e-mail: ushakovda@vniitf.ru

исходной сетки на параобласти. Каждая параобласть рассчитывается своим MPI-процессом, который производит моделирование теплопроводности либо на ядрах CPU (версия MPI+OpenMP), либо на GPU (версия MPI+CUDA). Обе версии реализованы так, что для любого MPI-процесса не имеет значения, какую из них используют остальные MPI-процессы. Вычисления, при которых используются обе версии параллельной реализации, будем называть *гибридными вычислениями*.

При различной мощности вычислительных устройств необходимо распределить работу между ними так, чтобы не происходило простоя оборудования. При геометрической декомпозиции исходной сетки объем работы определяется размером параобласти, и тем самым задача сводится к определению размеров параобластей для каждого из вычислительных устройств. Долю вычислений на одном GPU от полного объема вычислений на CPU в рамках одного узла будем называть *коэффициентом балансировки*. В работе показано, что существует единственный коэффициент балансировки, при котором достигается максимальная производительность гибридных вычислений. Также представлен простой алгоритм оценки оптимального коэффициента балансировки.

Основное содержание работы разбито на пять разделов. В разд. 1 коротко рассматривается схема решения нелинейного уравнения теплопроводности. В разд. 2 описывается используемая двухуровневая схема распараллеливания численного решения уравнения теплопроводности. В разд. 3 описывается алгоритм метода сопряженных градиентов для решения возникающих СЛАУ, представлена оптимизация, связанная с сокращением обращений к памяти и уменьшением накладных расходов на обмены. В разд. 4 подробно описаны организация гибридных вычислений и их балансировка. В разд. 5 приведены результаты численных экспериментов. В заключение сделаны выводы о проделанной работе.

1. Схема решения нелинейного уравнения теплопроводности

Рассматривается процесс распространения тепла в двумерной области W, ограниченной гладкой замкнутой поверхностью Σ. В области W решается уравнение теплопроводности в цилиндрической системе координат с осевой симметрией или в декартовых координатах на плоскости, записанное в потоковой форме:

$$\begin{aligned} &\rho\frac{\partial\varepsilon}{\partial t} + \operatorname{div}\mathbf{F} = \rho Q; \\ &\mathbf{F} + D\operatorname{grad}T = 0. \end{aligned} \tag{1}$$

Здесь ρ и $T(\mathbf{r},t)$ — плотность и температура вещества; $\varepsilon(\rho,T)$ — удельная внутренняя энергия; $D(\rho,T)$ — коэффициент теплопроводности; $Q(\mathbf{r},t)$ — удельное энерговыделение; $\mathbf{F}$ — тепловой поток. Данная система дополняется уравнением состояния $\varepsilon = \varepsilon(\rho,T)$, и для нее в области W ставится смешанная краевая задача

$$T^0 = T(\mathbf{r},t)|_{t=0}; \quad \alpha(\mathbf{r},t)\,T - \beta(\mathbf{r},t)\,\mathbf{F}\cdot\mathbf{n} = \mu(\mathbf{r},t), \quad \mathbf{r}\in\Sigma,$$

где $\mathbf{n}$ — внешняя нормаль к поверхности Σ; $\alpha,\beta,\mu = \text{const}$.

Покроем область W неструктурированной сеткой, состоящей из произвольных треугольных и четырехугольных ячеек. Обозначим центры ячеек индексом i, грани (ребра) ячеек — индексом s. Множество всех ячеек сетки обозначим Ω. Упорядоченное множество ребер ячейки i обозначим $E(i)$. В центрах ячеек определены: температура T_i; внутренняя энергия ε_i; масса ячейки m_i; объем ячейки V_i; плотность ячейки ρ_i; удельное энерговыделение Q_i. В центре ребра определены: внешняя единичная нормаль $\mathbf{n}_s$ к данному ребру рассматриваемой ячейки; длина ребра $\Delta\ell_s$; расстояние от оси симметрии до центра ребра R_s. При решении уравнения теплопроводности в декартовой системе координат $R_s = 1$ для всех ребер.

Проинтегрируем первое уравнение (1) по объему. Далее перейдем к интегрированию по поверхности и, используя неявный подход, получим следующее разностное уравнение для каждой ячейки $i\in\Omega$:

$$\rho_i\frac{\varepsilon_i^{n+1}-\varepsilon_i^n}{\tau} + \frac{1}{V_i}\sum_{s\in E(i)} R_s\Delta\ell_s\mathbf{n}_s\cdot\mathbf{F}_s^{n+1} = \rho_i Q_i^n. \tag{2}$$

Используя линеаризацию внутренней энергии по температуре

$$\varepsilon^{\nu+1} = \varepsilon\left(\rho, T^{n+1}\right)^{\nu+1} = \varepsilon^{\nu} + \frac{\partial \varepsilon^{\nu}}{\partial T^{\nu}}\left(T^{\nu+1} - T^{\nu}\right),$$

где ν — индекс итерации по нелинейности, уравнение (2) можно переписать в следующем виде:

$$m_i \left(\frac{\partial \varepsilon}{\partial T}\right)_i^{\nu} T_i^{\nu+1} + \tau \sum_{s \in E(i)} R_s \Delta \ell_s \mathbf{n}_s \cdot \mathbf{F}_s^{\nu+1} = m_i \left[\tau Q_i^n + \left(\frac{\partial \varepsilon}{\partial T}\right)_i^{\nu} T_i^{\nu} + (\varepsilon_i^n - \varepsilon_i^{\nu})\right]. \tag{3}$$

Не вдаваясь в детали вычисления значений потоков $\mathbf{F}_s^{\nu+1}$, отметим, что они имеют вид $\mathbf{F}_s^{\nu+1} = \mathbf{k}\left(T_i^{\nu+1} - T_k^{\nu+1}\right)$, где $\mathbf{k} = \mathbf{k}(\mathbf{r}, D)$ — некоторый коэффициент; $T_k^{\nu+1}$ — температура в соседней ячейке, имеющей общий узел с рассматриваемой (на рис. 1 красным цветом выделена рассматриваемая ячейка, желтым — соседние ячейки, которые учитываются при вычислении потоков).

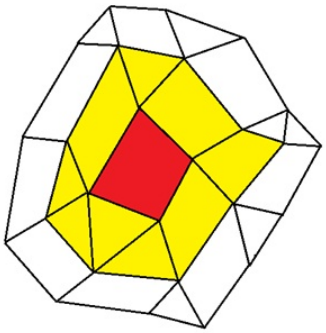

Рис. 1. Вычислительный шаблон схемы

Для определения значений температур в центрах ячеек необходимо решить систему уравнений (3), записанных для каждой ячейки. Эта система может быть представлена в виде

$$A\mathbf{x} = \mathbf{b}, \tag{4}$$

где матрица A имеет разреженную структуру. Аппроксимация исходных уравнений выполнена таким образом, что данная матрица является симметричной и положительно определенной.

При проведении расчета на структурированной четырехугольной сетке среднее количество ненулевых элементов в строке матрицы равно девяти. Номера строк и столбцов ненулевых элементов определяются номерами ячеек в исходной сетке. Поскольку матрица имеет разреженную структуру, авторы выбрали способ ее хранения в формате CSR (Compressed Sparse Row) [3].

На практике решение СЛАУ занимает от 80 до 95 % всего времени моделирования, поэтому эффективность реализации алгоритма решения СЛАУ в конечном итоге определяет скорость вычислений для всей схемы. В силу этого не будем касаться деталей реализации самой схемы, а подробнее рассмотрим метод решения СЛАУ, описав сначала используемую схему распараллеливания.

2. Схема распараллеливания

Исходная сетка состоит из треугольных и четырехугольных ячеек, причем большинство ячеек являются четырехугольными. Для расчета рассматриваемого класса задач достаточно 10^6—10^8 ячеек.

Для распараллеливания на распределенной памяти (MPI) используется геометрическая декомпозиция. Исходная сетка разбивается на параобласти. Каждая параобласть рассчитывается своим MPI-процессом. Вычисления для каждой параобласти распараллелены на общей памяти (OpenMP или CUDA). Используется автоматическая векторизация кода средствами компилятора.

На рис. 2, *а* красной линией выделена граница между двумя параобластями. На сетке отмечены три типа ячеек: внутренние (белого цвета), обменные (синего цвета), виртуальные (серого цвета). На рис. 2, *б* красной линией выделена главная диагональ матрицы, квадратными маркерами отмечены ненулевые элементы ячеек, каждый столбец соответствует своей ячейке: внутренней (черной), обменной (синей), виртуальной (серой).

Геометрическая декомпозиция сетки приводит к тому, что исходная матрица A представляется в виде набора лент, каждая лента матрицы содержит некоторый уникальный набор строк матрицы A и хранится на отдельном MPI-процессе (см. рис. 2, *б*). Для согласований решений между процессами используется технология виртуальных ячеек. Так как исходя из уравнений (3) для используемой

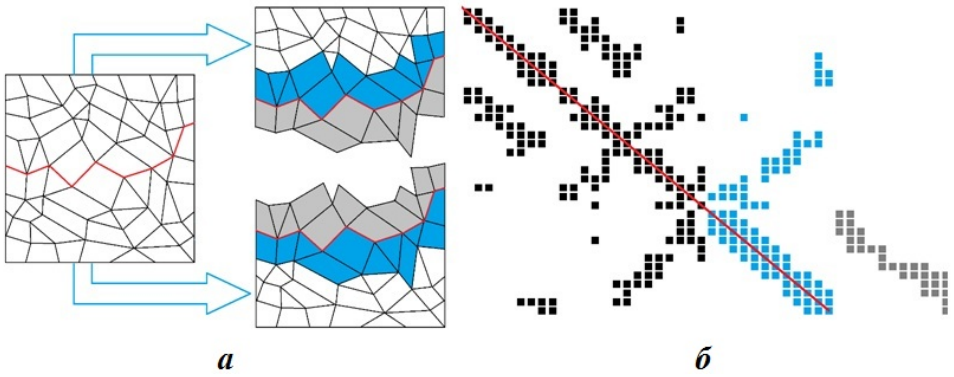

Рис. 2. Пример геометрической декомпозиции на две параобласти: *а* — сетки параобластей, *б* — портрет матрицы A для верхней параобласти

схемы необходим, как минимум, один слой соседних ячеек вокруг каждой ячейки, то размер виртуальной области имеет ширину в одну ячейку (см. рис. 2, *а*). Следовательно, каждая параобласть, помимо своих локальных ячеек (синие и белые на рис. 2, *а*), содержит еще и виртуальные ячейки; количество таких ячеек можно оценить примерно как $4\sqrt{N_{cells}}$, где N_{cells} — число ячеек в параобласти. Однако каждый MPI-процесс обновляет значения только для локальных ячеек, поэтому в начале каждой итерации по нелинейности необходимо получить все значения для виртуальных ячеек, которые вычисляются на других MPI-процессах. Ячейки, из которых отправляются данные, назовем обменными.

3. Метод сопряженных градиентов

Для решения полученной линейной системы (4), поскольку матрица A является симметричной и положительно определенной, используется метод сопряженных градиентов.

Алгоритм для параллельной реализации метода можно записать следующим образом:

$$
\begin{aligned}
&\mathbf{r}_j = \mathbf{b} - A\mathbf{x}_0 \\
&\mathbf{p}_j = \mathbf{r}_j \\
&\rho_{j+1} = \mathbf{r}_j \cdot \mathbf{r}_j \\
&\rho_j = 1 \\
&while\ \left(\rho_{j+1} < \varepsilon^2\right) \\
&\qquad \mathbf{q}_j = A\mathbf{p}_j \\
&\qquad \alpha = \rho_j / \left(\mathbf{p}_j \cdot \mathbf{q}_j\right) \\
&\qquad \mathbf{x}_{j+1} = \mathbf{x}_j + \alpha\mathbf{p}_j \\
&\qquad \mathbf{r}_{j+1} = \mathbf{r}_j - \alpha\mathbf{q}_j \\
&\qquad \rho_{j+1} = \mathbf{r}_{j+1} \cdot \mathbf{r}_{j+1} \\
&\qquad \mathbf{p}_{j+1} = \left(\rho_{j+1}/\rho_j\right)\ \mathbf{p}_j + \mathbf{r}_{j+1}
\end{aligned}
$$

В силу геометрической декомпозиции в алгоритме метода появляется необходимость в обменах между MPI-процессами. Синим цветом выделены вычисления, после которых необходимо произвести обмен частичными суммами для получения скалярного произведения векторов. Во время обмена происходит синхронизация всех MPI-процессов. После операции, выделенной красным цветом, происходит изменение компонент вектора $\mathbf{p}$, которые соответствуют обменным ячейкам для текущего MPI-процесса и виртуальным ячейкам для соседних MPI-процессов. Следовательно, перед операцией $A\mathbf{p}$ необходимо обновить соответствующие компоненты вектора $\mathbf{p}$. Во время этого обмена также происходит множество синхронизаций для соседствующих MPI-процессов.

Решение СЛАУ представляется в виде набора векторных и матрично-векторных операций, которые можно реализовать с помощью циклов и двойных циклов соответственно. При вычислениях на CPU данные циклы распараллеливаются с помощью OpenMP, а при вычислениях на GPU необходимые математические операции уже присутствуют в библиотеках CUDA, таких как Cublas [4], Cusparse [5]. Поэтому формулы линейной алгебры реализуют в виде отдельных функций, таких как умножение матрицы на вектор, скалярное произведение двух векторов и т. д.

Для рассматриваемого метода можно получить небольшой выигрыш, если отойти от этого правила и реализовать внутри одного цикла функции для выражений, выделенных в табл. 1 одним цветом. Данный подход позволяет сократить число обращений к памяти. Так, для набора, обозначенного красным цветом, при вычислении скалярного произведения нет необходимости считывать повторно значения компонент вектора $\mathbf{q}$, так как они еще находятся в регистрах процессора после умножения строки матрицы на вектор-столбец. Для набора зеленого цвета после определения нового значения вектора невязки можно сразу же начинать вычисление скалярного произведения. Для "синего набора" можно сэкономить на повторном обращении к вектору $\mathbf{p}$.

Проведем анализ данного алгоритма для вариантов с объединением вычислений (модифицированная версия) и без него (базовая версия), используя roofline-модель [6]. Будем анализировать только итерационную часть алгоритма, считая, что число итераций много больше единицы. Также предполагаем, что у вычислительной системы отсутствует кэш-память. Если считать, что число ненулевых элементов в строках матрицы одинаково и равно L, то получим значения для объема считанных/записанных данных и количества арифметических операций, приходящихся на обработку одной строки матрицы, указанные в табл. 1.

Таблица 1

Анализ итерационной части метода сопряженных градиентов для двух вариантов: с объединением вычислений (модифицированная версия) и без него (базовая версия)

Выражение	Считанные/записанные данные, байт		Количество арифметических операций
	базовая версия	модифицированная версия	
$\mathbf{q}_j = A\mathbf{p}_j$	$20L + 12$	$20L + 20$	$2L - 1$
$\alpha = \rho_j / \mathbf{p}_j \cdot \mathbf{q}_j$	16		2
$\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha\mathbf{q}_j$	24	24	2
$\rho_{j+1} = \mathbf{r}_{j+1} \cdot \mathbf{r}_{j+1}$	8		2
$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha\mathbf{p}_j$	24	40	2
$\mathbf{p}_{j+1} = (\rho_{j+1}/\rho_j)\,\mathbf{p}_j + \mathbf{r}_{j+1}$	24		2
Всего:	$20L + 108$	$20L + 84$	$2L + 9$

Из табл. 1 видно, что при объединении вычислений количество обращений в память для итерационной части метода сопряженных градиентов снижается на 9 % при $L = 9$ (соответствует четырехугольной сетке) и на 7 % при $L = 13$ (соответствует треугольной сетке).

Согласно roofline-модели *арифметическая интенсивность* (AI) *алгоритма* — это отношение количества арифметических операций к количеству прочитанных/записанных байт данных. На рис. 3 представлена арифметическая интенсивность для базовой и модифицированной версий, а также показана AI при $L \to \infty$, равная 0,1.

При известной арифметической интенсивности алгоритма согласно roofline-модели можно оценить производительность, получаемую на вычислительном устройстве, как $\min(AI \cdot BW; PeakPerf)$, где BW — пропускная способность памяти данного устройства, $PeakPerf$ — его пиковая производительность. Для современных вычислительных устройств, как правило, $AI \cdot BW$ много меньше, чем $PeakPerf$. Отсюда производительность метода сопряженных градиентов есть $AI \cdot BW$, и в этом случае принято говорить, что алгоритм ограничен пропускной способностью памяти. Подтверждают этот факт результаты набирающего в последнее время популярность теста HPCG [7]. Следовательно, насколько сокращено количество обращений в память при объединении вычислений, настолько же увеличивается производительность.

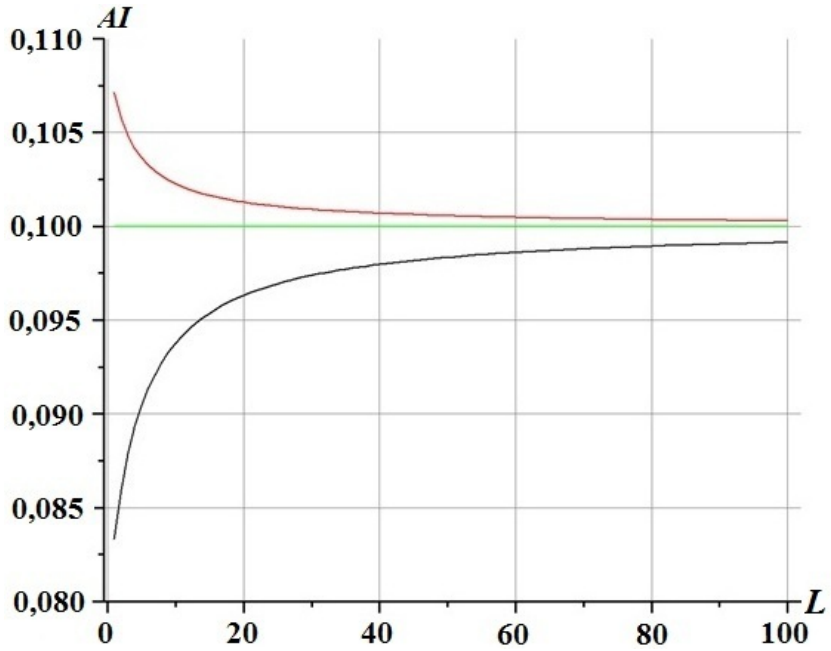

Рис. 3. Сравнение зависимостей арифметической интенсивности от количества ненулевых элементов в строке матрицы: —— — базовая версия алгоритма; —— — модифицированная версия; —— — асимптота, арифметическая интенсивность алгоритма при $L \to \infty$

Расчеты, проводимые на суперкомпьютерах, выполняются в многопользовательском режиме. Обычно ресурсы вычислительных узлов предоставляются в эксклюзивное пользование, а коммуникационная среда, связывающая вычислительные узлы в единое целое, остается общей для всех пользователей, работающих на суперкомпьютере. Поэтому время на обмен сообщениями между узлами может сильно различаться в зависимости от загруженности коммуникационной среды. Обычно для уменьшения влияния загруженности среды обмены "прячут" за вычислениями.

Сборку частичных сумм для скалярных произведений в рассматриваемом случае не скрыть. Скрыть можно обмены соответствующими компонентами вектора $\mathbf{p}$. Для этого пронумеруем ячейки параобласти в следующем порядке: внутренние, обменные (в количестве k штук) и виртуальные. С такой нумерацией ячеек при умножении матрицы на вектор последние k строк будут содержать в последних столбцах ненулевые элементы, которые относятся к виртуальным ячейкам (см. рис. 2, *б*). В этом случае операцию $\mathbf{q}_j = A\mathbf{p}_j$ можно разделить на два этапа: 1) инициализация асинхронного обмена соответствующими компонентами вектора $\mathbf{p}$ и на фоне данного обмена вычисление $\mathbf{q}_j = A\mathbf{p}_j$ для внутренних ячеек; 2) по окончании обмена вычисление $\mathbf{q}_j = A\mathbf{p}_j$ для обменных ячеек.

Напомним, что решение СЛАУ занимает от 80 до 95 % всего времени моделирования процесса теплопроводности. В силу этого при вычислениях на GPU не стоит реализовывать на CUDA только решение СЛАУ. Необходимо, чтобы на CUDA была реализована вся рассматриваемая схема: частичная реализация имеет большие накладные расходы, связанные с тем, что время на загрузку СЛАУ и выгрузку решения в/из графического ускорителя может быть соизмеримо со временем решения СЛАУ. Практика показала, что на сетке с 1 млн ячеек потеря производительности при частичной реализации (только решение СЛАУ) может составить до 22 %, а на сетке с 78 млн ячеек — до 41 %. По трудозатратам частичная реализация значительно проще, так как легко выполняется с использованием библиотек Cublas, Cusparse и не требует освоения CUDA. Полная же реализация всей схемы не может быть представлена в виде набора функций, которые присутствуют в библиотеках CUDA (например, заполнение СЛАУ). Поэтому требуется освоение CUDA и архитектуры графических видеокарт для написания собственных функций, которые могут быть запущены на ускорителях.

4. Гибридные вычисления

При вычислениях только на GPU необходимо использовать одно ядро CPU, которое выполняет только управленческую работу: выделяет/освобождает память на GPU, загружает/выгружает данные в/из GPU, делегирует вычислительные задачи GPU. Остальные ядра CPU не используются. Поэтому было решено загрузить вычислениями и оставшиеся, незадействованные, ядра CPU. Данный режим работы будем называть гибридными вычислениями.

Единственная дополнительная информация, которую каждый MPI-процесс хранит при распараллеливании на распределенной памяти, — виртуальные ячейки и ранки MPI-процессов, от которых необходимо принять значения в данных ячейках, а также ранки MPI-процессов, которым необходимо передать значения в обменных ячейках. В силу этого каждый MPI-процесс не зависит от того, какую реализацию (архитектуру) используют другие MPI-процессы.

Таким образом, для организации гибридных вычислений необходимы две отдельные полноценные реализации, чтобы одна производила вычисления только на CPU, а вторая — только на GPU. Для удобства запуска и из-за наличия повторяющегося кода данные реализации можно объединить в одну программу. Входным параметром программы является режим вычислений: на CPU, на GPU или гибридный режим счета. Для гибридных вычислений также необходимо задать количество MPI-процессов, которые будут использовать вычисления на GPU (остальные MPI-процессы будут определены для вычислений на CPU). Пример задействования ядер при гибридных вычислениях показан на рис. 4, где всего имеется четыре MPI-процесса (используемые ядра каждого выделены одним цветом).

Зачастую каждый вычислительный узел представляет собой NUMA (Non-Uniform Memory Access) систему; например, на рис. 4 вычислительный узел состоит из двух NUMA-узлов. При вычислениях на таких системах для максимальной скорости доступа к оперативной памяти выполняемые процессы и их потоки должны использовать только оперативную память своего NUMA-узла. Таким образом, лучший вариант — разбить вычислительный узел на отдельные NUMA-узлы и для каждого NUMA-узла назначить ядра процессора для CPU- и GPU-режимов счета. В данной работе авторы не рассматривали режим, в котором один MPI-процесс управляет несколькими GPU асинхронно.

Вследствии того, что производительности GPU и CPU могут быть различны, необходимо определить объем вычислений, приходящийся на GPU и CPU, для сбалансированности времени вычислений на этих устройствах. Поставим задачу следующим образом. Пусть имеется сетка из N ячеек и вычислительное поле, состоящее из K одинаковых узлов. Тогда на каждый узел приходится N/K ячеек. Далее в рамках каждого узла данное количество ячеек распределяется между MPI-процессами, из которых R_{GPU} и R_{CPU} процессов производят вычисления на GPU и CPU соответственно. Пусть на один MPI-процесс, производящий вычисления на GPU, приходится N_{GPU} ячеек, а на один MPI-процесс на CPU — N_{CPU} ячеек. Введем соотношение между количеством ячеек, рассчитываемых на одном GPU, и суммарным количеством ячеек на MPI-процессах, выполняемых на CPU ($R_{CPU}N_{CPU}$):

$$N_{GPU} = \lambda R_{CPU} N_{CPU},$$

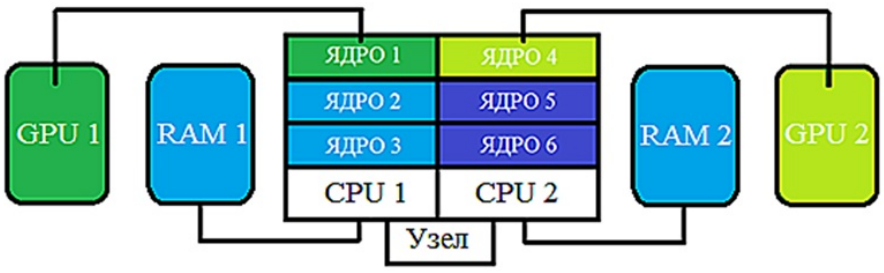

Рис. 4. Схема задействования вычислительного узла при гибридных вычислениях

где λ назовем коэффициентом балансировки. Полное количество ячеек равно

$$N = K(R_{GPU}N_{GPU} + R_{CPU}N_{CPU}).$$

Отсюда, используя введенное соотношение, можно выразить N_{CPU} и N_{GPU} через коэффициент балансировки λ:

$$N_{GPU} = \frac{N}{K}\frac{\lambda}{\lambda R_{GPU}+1}; \quad N_{CPU} = \frac{N}{K}\frac{1}{R_{CPU}}\frac{1}{\lambda R_{GPU}+1}. \tag{5}$$

Значение параметра λ должно определяться исходя из главной цели — максимальной производительности гибридных вычислений, точнее производительности при решении СЛАУ, поскольку оно занимает 80—95 % всего времени. Обозначим данную производительность, зависящую от коэффициента балансировки, как $Perf_{hybrid}(\lambda)$.

Исследуем поведение функции $Perf_{hybrid}(\lambda)$. Пусть $T_{CPU}(\lambda)$ и $T_{GPU}(\lambda)$ — время решения СЛАУ (не учитывая время на обмены) на MPI-процессе, который производит вычисления на CPU и GPU соответственно. Очевидно, что при увеличении N_{GPU} время $T_{GPU}(\lambda)$ монотонно возрастает. При $\lambda \to 0$ из формул (5) следует, что $N_{GPU} \to 0$, $N_{CPU} \to \dfrac{N}{KR_{CPU}}$. При $\lambda \to +\infty$ возникает противоположная картина: $N_{GPU} \to \dfrac{N}{KR_{GPU}}$, $N_{CPU} \to 0$. Значит, если λ увеличивается, то $T_{GPU}(\lambda)$ монотонно возрастает, а $T_{CPU}(\lambda)$ монотонно убывает. Время решения СЛАУ (не учитывая время на обмены) при гибридных вычислениях есть $T_{hybrid}(\lambda) = \max(T_{GPU}(\lambda), T_{CPU}(\lambda))$, и эта функция имеет глобальный минимум в момент равенства $T_{GPU}(\lambda_{opt}) = T_{CPU}(\lambda_{opt})$. Производительность гибридных вычислений $Perf_{hybrid}(\lambda)$ обратно пропорциональна времени $T_{hybrid}(\lambda) + Ov_{hybrid}(\lambda)$, где $Ov_{hybrid}(\lambda)$ — время накладных расходов на организацию обменов; предположим, что оно равно некоторой константе Ov_{hybrid}. Так как $T_{hybrid}(\lambda)$ имеет глобальный минимум при $\lambda = \lambda_{opt}$, то $Perf_{hybrid}(\lambda)$ имеет глобальный максимум при оптимальном коэффициенте балансировки $\lambda = \lambda_{opt}$.

Произведем оценку λ_{opt}. Предположим, что задействуются все ядра вычислительного узла и MPI-процессы на CPU используют одинаковое количество ядер. Было показано, что максимальная производительность гибридных вычислений достигается при равенстве времен решения СЛАУ на MPI-процессах с вычислениями на CPU и GPU, которые можно выразить через соответствующий объем обрабатываемых данных и производительность:

$$T_{GPU}(\lambda) = T_{CPU}(\lambda) \quad \to \quad \frac{N_{GPU}Bytes_{SLE}}{Perf_{GPU}} = \frac{R_{CPU}N_{CPU}Bytes_{SLE}}{Perf_{CPUs}},$$

где $Perf_{GPU}$, $Perf_{CPUs}$ — производительность решения СЛАУ на одном GPU и всех CPU соответственно; $Bytes_{SLE}$ — средний объем обрабатываемых данных, приходящийся на одну строку при решении СЛАУ. Преобразовав данное равенство, получим $N_{GPU} = R_{CPU}\lambda_b N_{CPU}$, где $\lambda_b = Perf_{GPU}/Perf_{CPUs}$ — теоретическая оценка оптимального коэффициента балансировки λ_{opt}. Подставив коэффициент λ_b в (5), получим

$$N_{GPU} = \frac{N}{K}\frac{Perf_{GPU}}{Perf_{GPU}R_{GPU} + Perf_{CPUs}}; \quad N_{CPU} = \frac{N}{K}\frac{1}{R_{CPU}}\frac{Perf_{CPUs}}{Perf_{GPU}R_{GPU} + Perf_{CPUs}}.$$

Напомним, что производительность метода сопряженных градиентов ограничена пропускной способностью памяти. Пусть BW_{CPUs} — пропускная способность оперативной памяти вычислительного узла, BW_{GPU} — пропускная способность глобальной памяти одного графического ускорителя. Тогда можно произвести замену $Perf_{GPU} = AI \cdot BW_{GPU}$, $Perf_{CPUs} = AI \cdot BW_{CPUs}$, после чего получим $\lambda_b = BW_{GPU}/BW_{CPUs}$ и размеры параобластей

$$N_{GPU} = \frac{N}{K}\frac{BW_{GPU}}{BW_{GPU}R_{GPU} + BW_{CPUs}}; \quad N_{CPU} = \frac{N}{K}\frac{1}{R_{CPU}}\frac{BW_{CPUs}}{BW_{GPU}R_{GPU} + BW_{CPUs}}.$$

Заметим, что для получения оценок пропускной способности памяти необходимо ее измерить, так как зачастую реальная пропускная способность памяти ниже теоретической.

Важно, что при получении данной оценки был введен ряд допущений, из-за чего λ_b может отличаться от реальной λ_{opt}. Поэтому на практике необходимо выполнять поиск λ_{opt} в окрестностях λ_b.

5. Вычислительные эксперименты

Для вычислительных экспериментов была выбрана задача о мгновенном энерговыделении в точке [8]. Использовался один вычислительный узел с двумя 18-ядерными процессорами Intel Xeon Gold 6240 2.6 ГГц и восемью графическими ускорителями NVIDIA GPU Tesla V100 32GB. Суперкомпьютер "УРАН" [9] в составе ИММ УрО РАН оборудован одним узлом с такой архитектурой.

Будем исследовать зависимость производительности гибридных вычислений $Perf_{hybrid}(\lambda, N)$ от коэффициента балансировки λ и размера сетки N. Также для анализа дополнительно построим нижнюю и верхнюю границы производительности, которые зависят от размера сетки и не зависят от λ. В качестве нижней границы возьмем производительность вычислений только на GPU ($Perf_{GPU}(N)$), которая позволит судить о целесообразности гибридных вычислений. В качестве верхней границы возьмем сумму производительностей вычислений только на CPU и GPU ($Perf_{GPU+CPU}(N)$), которая показывает предельно возможную производительность гибридных вычислений. Будем рассматривать количество $GPUs$ используемых графических ускорителей от одного до четырех. Использование восьми GPU рассматривать не имеет смысла, поскольку теоретическое ускорение гибридных вычислений в этом случае мало:

$$\frac{BW_{CPUs}}{BW_{CPUs} + BW_{GPU}GPUs} = \frac{1}{1 + 8\lambda_b} = \frac{1}{26{,}6} = 0{,}038 = 3{,}8\,\%.$$

Перед выполнением расчетов построим график эффективности E использования GPU в зависимости от размера сетки, которая показывает, какой процент от максимальной производительности можно получить для данного размера сетки (рис. 5). Из рисунка следует, что максимальную производительность можно получить, если на каждый GPU приходится 19 млн ячеек.

На рис. 6, 7 для одного, двух и четырех GPU показаны производительности $Perf_{hybrid}(\lambda, N)$, $Perf_{GPU}(N)$, $Perf_{GPU+CPU}(N)$ относительно $Perf_{GPU}(N = 19)$ при $\lambda \in [0, 5]$ на сетках с 19, 78 и 312 млн ячеек. В подрисуночных подписях используются обозначения в виде N-$TYPE$, где $N = 19, 78, 312$ — размер сетки (в млн ячеек), $TYPE$= HYBRID, GPU, GPU+CPU — режим счета.

Исследуем точность предложенной оценки $\lambda_b = 3{,}2$ оптимального коэффициента балансировки λ_{opt}, при котором достигается максимальная производительность $Perf_{hybrid}(\lambda_{opt})$. На основе данных из рис. 6, 7 найдены λ_{opt} — оптимальный коэффициент балансировки и ΔP — разница между относительными производительностями для $Perf_{hybrid}(\lambda_{opt})$ и $Perf_{hybrid}(\lambda_b)$ (табл. 2).

Из табл. 2 видно, что отклонение оценки коэффициента балансировки от его оптимального значения составляет от нуля до 10,4 % с разницей в производительности от нуля до 1,4 %. Также можно

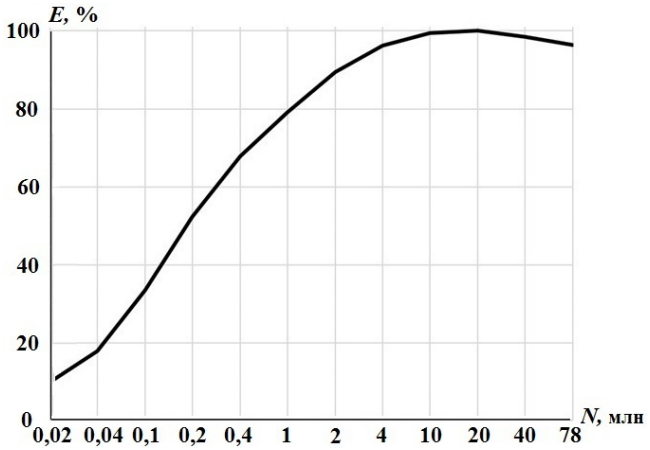

Рис. 5. Эффективность использования GPU в зависимости от размера сетки (в млн ячеек)

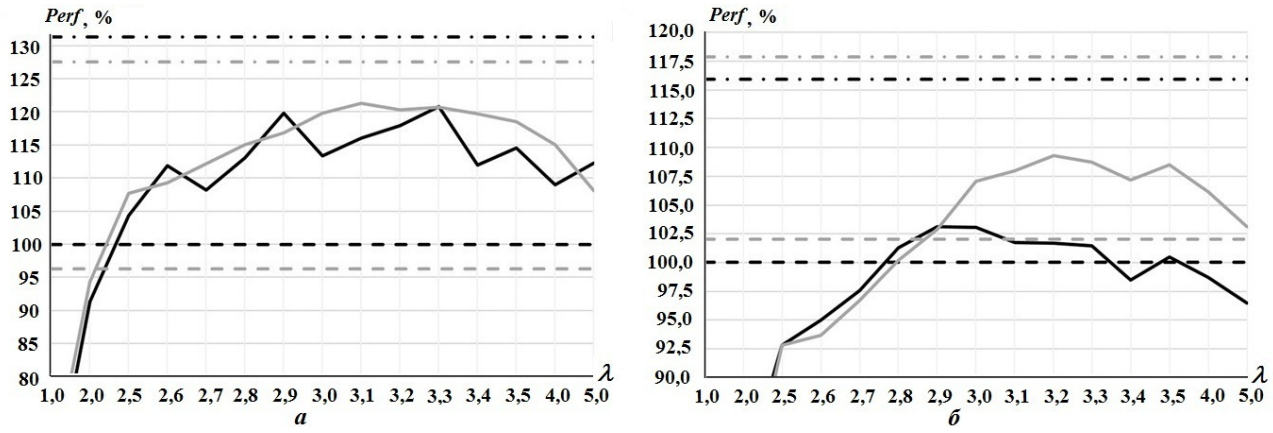

Рис. 6. Относительная производительность решения СЛАУ в зависимости от коэффициента балансировки при использовании одного (*а*) и двух (*б*) GPU на сетках с 19 и 78 млн ячеек: —— — 19-HYBRID; —— — 78-HYBRID; – – — 19-GPU; – – — 78-GPU; – · – — 19-GPU+CPU; – · – — 78-GPU+CPU

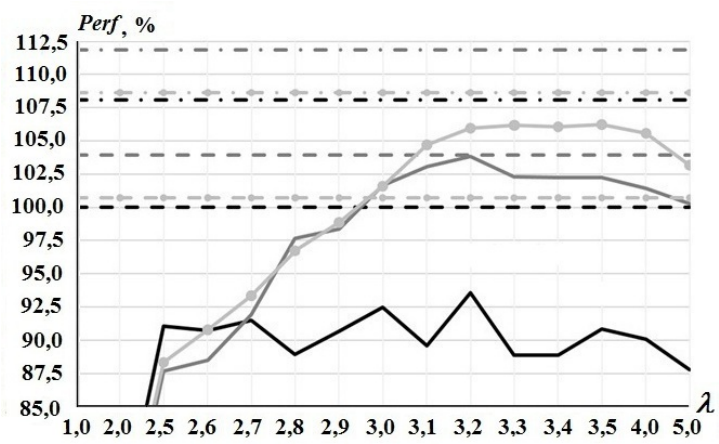

Рис. 7. Относительная производительность решения СЛАУ в зависимости от коэффициента балансировки при использовании четырех GPU на сетках с 19, 78, 312 млн ячеек: —— — 19-HYBRID; —— — 78-HYBRID; ●——● — 312-HYBRID; – – — 19-GPU; – – — 78-GPU; ●– –● — 312-GPU; – · – — 19-GPU+CPU; – · – — 78-GPU+CPU; ●– · ●– — 312-GPU+CPU

Таблица 2

Разница между оценкой оптимального коэффициента балансировки и его реальным значением

Кол-во ячеек сетки, млн	1 GPU			2 GPU			4 GPU		
	λ_b	λ_{opt}	ΔP, %	λ_b	λ_{opt}	ΔP, %	λ_b	λ_{opt}	ΔP, %
19	3,2	3,0	−1,0	3,2	2,9	−1,4	3,2	3,2	0,0
78	3,2	3,1	−0,9	3,2	3,2	0,0	3,2	3,2	0,0
312	–	–	–	–	–	–	3,2	3,3	−0,2

заметить, что чем больше размер сетки и количество задействованных GPU, тем точнее оценка. По данным результатам можно сделать вывод, что предложенная в работе оценка оптимального коэффициента балансировки достаточно точная и ее можно использовать на практике.

Из рис. 6, 7 также найдены максимальная эффективность и ускорение гибридных вычислений (табл. 3). Видно, что при фиксировании размера сетки и увеличении количества используемых GPU ускорение гибридных вычислений падает с 18,2 до -6, 4% для 19 млн ячеек, с 25,9 до -0,1 %

Таблица 3

Максимальное ускорение и эффективность распараллеливания гибридных вычислений

Количество ячеек сетки, млн	Количество используемых GPU	Ускорение гибридных вычислений относительно вычислений только на GPU, %	Эффективность гибридных вычислений, %
19	1	+18,2	100
	2	+3,1	86
	4	−6,4	75
78	1	+25,9	100
	2	+7,1	89
	4	−0,1	82
312	4	+5,4	100

для 78 млн ячеек, а на сетке с 312 млн ячеек равняется 5,4 %. Заметим, что не при всех параметрах производительность $Perf_{hybrid}$ выше производительности $Perf_{GPU}$. Наблюдается следующая картина: при увеличении числа используемых GPU растет минимальный размер сетки, при котором гибридные вычисления оказываются быстрее, чем вычисления только на GPU. Объяснить это могут следующие факторы, влияющие на $Perf_{hybrid}$:

- производительность повышается с использованием незадействованных ядер процессора;
- производительность понижают накладные расходы, которые выражаются в увеличении времени обмена и рассинхронизации вычислений между MPI-процессами;
- производительность может повышаться/понижаться за счет изменения эффективности использования GPU.

Время обмена возрастает в силу увеличившегося числа MPI-процессов (которые вводятся, чтобы использовать незадействованные ядра процессора), что непосредственно влияет на стоимость обменных операций. Также из-за увеличения числа MPI-процессов размер параобласти, приходящейся на каждый MPI-процесс, становится меньше, но размер обменных данных изменяется незначительно, следовательно, доля вычислений уменьшается, что отрицательно влияет на долю вклада каждого MPI-процесса в общую производительность $Perf_{hybrid}$. Уменьшение размера параобласти также влияет на эффективность использования GPU, которая представлена на рис. 5. Поэтому для расчетов 78-HYBRID с одним GPU (см. рис. 6) и 312-HYBRID с четырьмя GPU (см. рис. 7) эффективность использования GPU во время гибридных вычислений возрастает по сравнению с вычислениями только на GPU, что положительно сказывается на производительности. Для остальных расчетов эффективность, наоборот, падает, что отрицательно влияет на производительность. Так, например, при использовании четырех GPU наклон касательной графика эффективности использования GPU для расчета 19-HYBRID больше, чем для 78-HYBRID, поэтому потеря эффективности на сетке с 19 млн ячеек будет больше потери на сетке с 78 млн ячеек. Эффективность использования CPU, которая зависит от кэш-памяти процессора, можно принять за константу: поскольку даже для расчета HYBRID-19 с четырьмя GPU при $\lambda = 5$, когда размер параобласти на MPI-процессах с CPU минимальный среди всех расчетов, объем данных при решении СЛАУ в 2,5 раза больше размера кэш-памяти процессора, то кэш-память никак не влияет на скорость вычислений.

В итерационной части алгоритма решения СЛАУ существует три момента, когда может произойти рассинхронизация вычислений. Для наглядности приведем алгоритм метода сопряженных градиентов с операциями обмена, которые могут вызывать рассинхронизацию:

$$
\begin{aligned}
&SendRecv[\mathbf{p}_j]\\
&\mathbf{q}_j = A\mathbf{p}_j\\
&Allreduce[\mathbf{p}_j \cdot \mathbf{q}_j]\\
&\alpha = \rho_j/(\mathbf{p}_j \cdot \mathbf{q}_j)
\end{aligned}
$$

$$\mathbf{r}_{j+1} = \mathbf{r}_j - \alpha \mathbf{q}_j$$
$$Allreduce[\mathbf{r}_{j+1} \cdot \mathbf{r}_{j+1}]$$
$$\rho_{j+1} = \mathbf{r}_{j+1} \cdot \mathbf{r}_{j+1}$$
$$\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha \mathbf{p}_j$$
$$\mathbf{p}_{j+1} = (\rho_{j+1}/\rho_j)\ \mathbf{p}_j + \mathbf{r}_{j+1}.$$

Напомним, что все обмены (*SendRecv*, *AllReduce*) синхронные, т. е. каждый MPI-процесс при обмене дожидается полного его завершения и только после этого продолжает счет. Но во избежание рассинхронизации каждый MPI-процесс должен еще, закончив вычисления перед обменом, ждать завершения вычислений другими MPI-процессами, участвующими в этом обмене.

Для гибридных вычислений существуют две группы MPI-процессов: первая производит вычисления на GPU, вторая — на CPU. В рамках одной группы размер параобласти одинаковый, поэтому будем предполагать, что MPI-процессы одной группы завершают вычисления одновременно. Следовательно, рассинхронизация вычислений может возникнуть только между MPI-процессами разных групп и напрямую зависит от размера параобластей в рамках каждой группы, который определяется по формулам (5) исходя из выбранного коэффициента балансировки λ.

Из формул (5) следует прямая зависимость размера параобласти от коэффициента балансировки λ у MPI-процессов на GPU и обратная зависимость — у процессов на CPU. Отсюда если $\lambda < \lambda_{opt}$, то размеры параобластей MPI-процессов на GPU настолько малы, что они всегда заканчивают вычисления быстрее MPI-процессов на CPU, но, с другой стороны, им всегда необходимо простаивать во время обменов. Поэтому у MPI-процессов на GPU уменьшается вклад в общую производительность $Perf_{hybrid}$ при уменьшении λ, а у процессов на CPU, наоборот, увеличивается. При $\lambda > \lambda_{opt}$ происходит обратное. Так, в расчете 78-HYBRID с двумя GPU при уменьшении λ, начиная с 3,2, MPI-процессы на GPU будут простаивать, следовательно, уменьшится их доля вклада в общую производительность $Perf_{hybrid}$. А поскольку пиковая производительность GPU больше пиковой производительности CPU, то $Perf_{hybrid}$ уменьшается в разы быстрее, чем при увеличении λ, начиная с 3,2. Здесь увеличивается простой MPI-процессов с CPU, следовательно, уменьшается их доля вклада в общую производительность $Perf_{hybrid}$.

Из проведенного анализа становится ясно, что на производительность гибридных вычислений влияет множество факторов, которые необходимо учитывать, чтобы они были производительнее, чем вычисления только на GPU. Можно выделить три основных условия:

1) должна обеспечиваться сбалансированность вычислений, которая достигается выбором оптимального коэффициента балансировки. Точное значение коэффициента заранее не известно, но для его определения, как показали вычислительные эксперименты, можно использовать предложенную оценку;
2) объем вычислений, приходящийся на вычислительное устройство, должен быть оптимальным в смысле эффективности его использования;
3) коммуникационная среда должна обеспечивать высокую скорость обменов и обладать достаточной полнотой.

Влияние третьего условия можно устранить с помощью сокрытия обменов на фоне вычислений. Сокрытие обменов также сильно уменьшит накладные расходы на организацию гибридных вычислений, следовательно, увеличит их производительность. Так, в разд. 3 было показано, что можно полностью скрыть обмен виртуальными ячейками умножением матрицы на вектор. Однако обмен частичными суммами при вычислении скалярных произведений, как отмечалось там же, скрыть не удается, так как этого не позволяет сделать сам алгоритм метода сопряженных градиентов: результат скалярного произведения необходимо использовать сразу. Следовательно, остается найти более подходящий метод решения СЛАУ, алгоритм которого будет устроен так, чтобы все обмены можно было скрыть на фоне вычислений. Например, можно использовать модифицированный метод сопряженных градиентов [10].

Заключение

Для современных архитектур (Intel, AMD, Nvidia) показано, что производительность метода сопряженных градиентов ограничена пропускной способностью подсистемы памяти. Предпочтение в выборе архитектуры должно отдаваться устройствам с большей пропускной способностью подсистемы памяти.

Представленный способ уменьшения запросов к памяти для итерационной части метода сопряженных градиентов позволяет получить увеличение производительности на 9—13 % при проведении расчетов на сетках из треугольных и четырехугольных ячеек.

Показано, что зависимость производительности гибридных вычислений от коэффициента балансировки имеет глобальный максимум. Для оценки оптимального коэффициента балансировки предложен простой метод, основанный на возможностях подсистемы памяти.

Продемонстрирована эффективность гибридного режима вычислений. Эффективность составляет 75—89 %. Полученное максимальное ускорение гибридных вычислений относительно вычислений только на GPU составило 25,9 % для одного, 7,1 % для двух и 5,4 % для четырех GPU.

В настоящее время, исходя из архитектуры суперкомпьютеров рейтинга [1], пропускная способность памяти GPU в среднем в 4—9 раз выше, чем пропускная способность CPU, а количество GPU больше двух. Поэтому перед организацией гибридных вычислений, чтобы определить их целесообразность, необходимо произвести теоретическую оценку возможного максимального ускорения относительно вычислений только на GPU при известных пропускных способностях устройств и количестве графических ускорителей ($GPUs$) по формуле

$$\frac{BW_{CPUs}}{BW_{CPUs} + BW_{GPU}GPUs}.$$

Из вычислительных экспериментов ясно, что метод сопряженных градиентов не всегда позволяет получить производительность гибридных вычислений, близкую к теоретической, поскольку при вычислении скалярных произведений обмены частичными суммами невозможно совершать на фоне счета. В дальнейшем предполагается использовать модификацию метода сопряженных градиентов из статьи [10], который не имеет данного недостатка.

Список литературы

1. TOP500: Top-Ranked Systems // The TOP500 Project. November 2019. www.top500.org/lists/2019/11.
2. *Баландин М. Ю., Шурина Э. П.* Методы решения СЛАУ большой размерности. Нижний Новгород: НГТУ, 2000.
   Balandin M. Yu., Shurina E. P. Metody resheniya SLAU bolshoy razmernosti. Nizhniy Novgorod: NGTU, 2000.
3. *Писсанецки С.* Технология разреженных матриц. М.: Мир, 1988.
   Pissanetski S. Tekhnologiya razrezhennykh matrits. M.: Mir, 1988.
4. Cublas library. User guide // Cuda Toolkit Documentation. NVIDIA Corpotation, 2013. www.docs.nvidia.com/cuda/cublas.
5. Cusparse library. User guide // Cuda Toolkit Documentation. NVIDIA Corpotation, 2013. www.docs.nvidia.com/cuda/cusparse.
6. *Williams S, Waterman A, Petterson D.* Roofline: an insightful visual performance model for multicore architectures // Communications of the ACM. 2009. Vol. 52, No 4. P. 65—76.
7. HPCG Benchmark. www.hpcg-benchmark.org.
8. *Зельдович Я. Б., Райзер Ю. П.* Физика ударных волн и высокотемпературных гидродинамических явлений. М.: Наука, 1966.
   Zeldovich Ya. B., Rayzer Yu. P. Fizika udarnykh voln i vysokotemperaturnykh gidrodinamicheskikh yavleniy. M.: Nauka, 1966.

9. ИММ УрО РАН. Вычислительный кластер "УРАН". www.parallel.uran.ru/node/3.
IMM UrO RAN. Vychislitelny klaster "URAN". www.parallel.uran.ru/node/3.
10. *Cornelis J., Cools S., Vanroose W.* The Communication-Hiding Conjugate Gradient Method with Deep Pipelines. www.arxiv.org/abs/1801/04728.

Статья поступила в редакцию 08.07.20.

IMPLEMENTATION OF A METHOD FOR SOLVING THE 2D HEAT CONDUCTION EQUATION ON THE HYBRID ARCHITECTURE (CPU + GPU) / V. O. Anisov, E. M. Vaziev, D. A. Ushakov* (FSUE "Acad. E. I. Zababakhin RFNC-VNIITF", Snezhinsk, Chelyabinsk region).

The paper presents a way of organizing the computation process on a hybrid-architecture computer: a general-purpose processor (CPU) and a graphics accelerator (GPU). A numerical method for solving the 2D heat conduction equation was implemented on the architecture with two-level parallelism. Domain decomposition was used for computations on a distributed-memory (MPI). Computations on a shared memory were performed using OpenMP and CUDA for CPU and GPU, respectively. Load balancing between CPU and GPU is based on the evaluation of their memory performance as the main limiting factor for speedup. Computing overheads are also taken into account for load balancing. Special attention is paid to the implementation of the conjugate gradient method, because linear equations take most of the time required for solving the heat conduction equation.

Keywords: hybrid computing, heat conduction, conjugate gradient method.

*Ushakov Denis Aleksandrovich, scientist,
e-mail: ushakovda@vniitf.ru