ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ БЛОКА МОДУЛЯ NEWT, ПРЕДНАЗНАЧЕННОГО ДЛЯ СЧЕТА СИСТЕМ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ МЕТОДОМ ПЕРЕМЕННЫХ НАПРАВЛЕНИЙ НА АРИФМЕТИЧЕСКИХ УСКОРИТЕЛЯХ

С. О. Черных, Р. А. Королев, А. С. Сухих

ФГУП «РФЯЦ-ВНИИЭФ», г. Саров Нижегородской обл.

Введение

Одним из направлений развития вычислительной техники является продвижение векторных процессоров (блоки векторной арифметики SSE, графические процессоры GPU) [1]. Использование GPU в качестве арифметических ускорителей (АрУ) заметно снижает стоимость и энергоемкость единицы реальной производительности вычислительной системы.

Методы решения систем линейных алгебраических уравнений (СЛАУ) с разреженными матрицами состоят из построения предобусловливателя и итерационного уточнения решения. Этап итерационного уточнения включает операции умножения исходной матрицы на вектор, решение СЛАУ с предобусловливателем, вычисление скалярного произведения и обновление вектора. Две последние операции векторизуются естественным образом. Достаточно эффективно может быть реализовано на АрУ умножение матрицы на вектор. Основную трудность составляет вычисление предобусловливателя и решения СЛАУ с предобусловливателем.

Начиная с середины прошлого века, с момента появления вычислительных машин, для решения различных классов задач было разработано огромное число различных предобусловливателей. Последние годы в счетных методиках используется и развивается, главным образом, метод неполного *LU*-разложе-ния (ILU), имеющий высокую эффективность и устойчивость. К сожалению, нам пока непонятно, как эффективно векторизовать метод ILU для АрУ. Известные нам публикации также не обещают заметного преимущества в использовании АрУ для метода ILU [2].

В качестве предобусловливателя, эффективно использующего АрУ, был выбран метод переменных направлений [3]. Этот метод уступает по скорости сходимости методу ILU. Вместе с тем, метод переменных направлений имеет значительно более быструю итерацию по сравнению с методом ILU, лучше векторизуется и распараллеливается на машинах с общей и распределенной памятью, а для его вычисления требуется ничтожно малое процессорное время.

Настоящая реализация метода переменных направлений ограничивает класс доступных ей задач регулярными методиками. Для тестирования решателя в рамках программы расчета теплопроводности были подготовлены три модельных тестовых задачи для уравнения теплопроводности в трехмерной геометрии. В работе представлены сравнения длительности счета СЛАУ и его доли в общем балансе затрат процессорного времени счета задач, с использованием и без использования АрУ.

Авторы работы выражают признательность В. В. Южакову за активную информационную поддержку и С. В. Чеботарю за помощь в работе.

1. Адаптация модуля NEWT для счета на АРУ

1.1. Описание алгоритма счета модуля NEWT, адаптированного под АрУ

Общая схема решения СЛАУ вида $Ax = f$ с разреженными матрицами A итерационными методами включает два этапа:

- вычисление предобусловливателя ($P \approx A$);
- итерационное уточнение решения.

На этапе итерационного уточнения итерационный алгоритм применяется к решению предобусловленной СЛАУ, эквивалентной исходной системе, например, вида $P^{-1}Ax = P^{-1}f$. Хорошая аппроксимация исходной матрицы предобусловливателем обеспечивает быструю сходимость итерационного алгоритма за счет хорошей обусловленности и/или благоприятного взаимного расположения собственных значений итерируемой матрицы вида $P^{-1}A$. В качестве предобусловливателя, допускающего эффективную реализацию на АрУ, выбран метод переменных направлений (ADI). Для итерационного уточнения решения был запрограммирован итерационный алгоритм BiCGstab.

1.2. Метод переменных направлений

Предположим, что требуется решить СЛАУ $Ax = f$ с матрицей вида $A = D + A_x + A_y + A_z$, где матрица D является диагональной, а матрицы A_x, A_y и A_z являются двухдиагональными, имеют нулевую главную диагональ и равноудаленные от главной диагонали ненулевые матричные элементы. Матрица с таким

портретом расположения ненулевых матричных элементов порождается, в частности, при аппроксимации оператора диффузии в трехмерной геометрии на регулярной сетке с использованием семиточечного разностного шаблона.

Предобусловливатель метода ADI можно записать в следующем виде [3]:

$$A = D + A_x + A_y + A_z \approx P =$$

$$= \left(\omega^{-1}D + A_x\right)\omega D^{-1}\left(\omega^{-1}D + A_y\right)\omega D^{-1}\left(\omega^{-1}D + A_z\right), \quad (1)$$

где ω – вещественный параметр. Матрицы вида $\left(\omega^{-1}D + A_x\right)$ являются трехдиагональными, и системы линейных уравнений с ними решаются методом *LU*-разложения, эквивалентным методу прогонок.

С помощью переупорядочения строк и столбцов каждая из матриц вида $\left(\omega^{-1}D + A_x\right)$, $\left(\omega^{-1}D + A_y\right)$ и $\left(\omega^{-1}D + A_z\right)$ может быть представлена в виде блочно-диагональной ленточной матрицы с шириной ленты, равной 3. При вычислении *LU*-разложения вида $\left(\omega^{-1}D + A_x\right) = L_x D_x U_x$, где матрица D_x является диагональной, а матрицы L_x и U_x являются нижней и верхней треугольными двухдиагональными с единичной главной диагональю, разложение каждого блока можно выполнять независимо. Матрицы разложения L_x и U_x также являются блочно-диагональными, поэтому процедура решения СЛАУ вида $L_x u = f$ и $U_x v = g$ может быть реализована как решение нескольких независимых линейных систем. Это обстоятельство справедливо также для матриц $\left(\omega^{-1}D + A_y\right)$ и $\left(\omega^{-1}D + A_z\right)$. Блочно-диагональная структура матриц вида $\left(\omega^{-1}D + A_x\right)$, $\left(\omega^{-1}D + A_y\right)$ и $\left(\omega^{-1}D + A_z\right)$ лежит в основе реализованного нами способа распараллеливания процедуры решения СЛАУ с использованием предобусловливателей вида (1). Во всех расчетах методом переменных направлений полагалось ω = 1.

1.3. Итерационный алгоритм BiCGstab

Для итерационного уточнения решения на языке CUDA был реализован стабилизированный вариант метода бисопряженных градиентов алгоритм BiCGstab. Приведем вариант записи алгоритма BiCGstab в случае, когда оператор *P* используется как левый предобусловливатель.

Алгоритм BICGSTAB:

1. $r_0 = P^{-1}\left(b - Ax_0\right)$; r_0^* – произвольный вектор.
2. $p_0 = r_0$.
3. Цикл j = 0,1, … до сходимости выполняем шаги 4–12.
4. $p_j^* = P^{-1}Ap_j$.
5. $\alpha_j = \left(r_j, r_0^*\right)\left(p_j^*, r_0^*\right)$.
6. $s_j = r_j - \alpha_j p_j^*$.
7. $s_j^* = P^{-1}As_j$.
8. $\omega_j = \left(s_j^*, s_j\right)/\left(s_j^*, s_j^*\right)$.
9. $x_{j+1} = x_j + \alpha_j p_j + \omega_j s_j$.
10. $r_{j+1} = s_j - \omega_j s_j^*$.
11. $\beta_j = \dfrac{\left(r_{j+1}, r_0^*\right)}{\left(r_j, r_0^*\right)}\dfrac{\alpha_j}{\omega_j}$.
12. $p_{j+1} = r_{j+1} + \beta_j\left(p_j - \omega_j p_j^*\right)$.
13. Конец цикла.

Тестовые задачи решались с точностью, определяемой ограничением на меру невязки вида $\mu \le 10^{-8}$. Мера невязки *r* вычислялась по формуле

$$\mu = \max_i \frac{|r_i|}{d_i}, \quad r_i = f_i - \sum a_{ij}x_j,$$

$$d_i = |f|_i + \sum |a_{ij}||x_j|. \quad (2)$$

В общей схеме алгоритма решения СЛАУ на этапе вычисления предобусловливателя вычисляются и запоминаются треугольные двухдиагональные матрицы *LU*-разложения трехдиагональных матриц вида $\left(\omega^{-1}D + A_x\right)$, $\left(\omega^{-1}D + A_y\right)$ и $\left(\omega^{-1}D + A_z\right)$. На этапе итерационного уточнения решаются СЛАУ с вычисленными заранее треугольными двухдиагональными матрицами.

1.4. Описание реализации счета на АрУ

Модель применения АрУ в модуле NEWT подобна счету задачи с сопроцессором. Применение АрУ ограничено счетом трудоемких для ЦП участков кода. Логика и управление ходом программы остается за универсальным процессором.

Для адаптации всего модуля NEWT к счету на АрУ необходимо было:

- интегрировать фрагменты кода для АрУ с кодом Фортрана;
- найти наиболее трудоемкие участки кода в модуле NEWT;
- распараллелить и оптимизировать эти участки для счета на АрУ;
- добавить данные, обрабатываемые АрУ, в систему обмена данными между функциями в коде Фортрана в модуле NEWT.

Использование схемы Fortran–C–CUDA обусловлено несколькими причинами. Адаптация модуля NEWT под АрУ началась до появления CUDA Fortran. Кроме этого, как показали предварительные тесты, применение CUDA Fortran значительно снижает эффективность от применения АрУ.

1.4.1. Связь Fortran−C−CUDA

Интеграция фрагментов кода для АрУ в модуль NEWT предполагает вызов из кода Фортрана функций, адаптированных под АрУ. На момент начала работ по адаптации модуля NEWT под АрУ компилятор CUDA поддерживал вызовы функций CUDA только из кода Си. Таким образом, необходимо было настроить эффективную связь Fortran–C–CUDA. Си используется как интерфейс между Фортраном и функциями CUDA.

Для вызовов функций Си из кода Фортрана был применен стандартный подход с созданием статической библиотеки на Си и подключением et к программе на Фортране. Функции статической библиотеки доступны из любого места программы на Фортране.

Таблица 1

Пример связи Fortran–C–CUDA

Фортран	CALL CU_CUDA_CALL(count, arrayA)

Си	extern "C" void CU_CUDA_CALL(int &count, long &arrayA) {
	int trd = BLOCK_SIZE, grd =
	= count/BLOCK_SIZE;
	dim3 threads(trd), grid(grd);
	CUDA_CALL<<<grid, threads>>>(count, (double *)arrayA);
	}

CUDA	__global__ void CUDA_CALL (int count, double *arrayA) { … }

1.4.2. Методы распараллеливания функций модуля NEWT для счета на АрУ

При решении СЛАУ на АрУ происходит последовательный вызов распараллеленных и адаптированных под АрУ функций. У адаптированных под АрУ функций разная степень распараллеливания, однако, можно выделить две основных группы:

- мелкозернистое распараллеливание – количество параллельно исполняемых потоков АрУ в функции равно размерности исходной системы (например, функции умножения матрицы на вектор, скалярного произведения векторов, суммирования и т. д.);
- крупнозернистое распараллеливание – функции последовательно работают «вдоль» пространственных направлений, количество исполняемых потоков в функции равно числу независимых подсистем «вдоль» данного направления (например, функции декомпозиции, вычисление предобусловливателя и т. д.).

Характеристикой, которая кардинально влияет на производительность задачи на АрУ, является количество исполняемых потоков. Большое количество потоков скрывает латентность доступа к памяти АрУ и более плотно загружает ядра.

Мелкозернистый параллелелизм позволяет эффективно использовать АрУ. Таким образом, чем больше потоков и степень параллелелизма функции, тем большее ускорение достигается на АрУ.

1.4.3. Функции модуля NEWT, адаптированные для счета на АрУ

Основная часть работ по адаптации счета модуля NEWT под АрУ – это выделение наиболее трудоемких для универсального процессора участков кода, их распараллеливание и оптимизация для счета на АрУ.

Наиболее затратные функции: расчет предобусловливателя, умножение матрицы на вектор и вычисление скалярного произведения. Оптимизации этих функций было уделено основное время. Доля основных адаптированных функций в общем времени расчета СЛАУ варьируется от 30 до 80 %.

Адаптировать к АрУ только наиболее «тяжелые» для ЦП функций нецелесообразно. В таком случае потребуются обмены «АрУ − оперативная память ЦП» после каждого расчета такой «тяжелой» функции, что существенно увеличит длительность счета модуля NEWT. Одно из требований эффективного счета на АрУ − минимизация обменов между оперативной памятью ЦП и АрУ. Идеальный сценарий – на все вычисления одна загрузка исходных значений и одна выгрузка результата. Для реализации этого сценария адаптированы все функции модуля NEWT, которые работают с данными, находящимися в памяти АрУ, даже если степень их распараллеливания невелика. Таким образом, обмены между оперативной памятью ЦП и АрУ сводятся к загрузке начальных данных и выгрузке результата после решения СЛАУ.

Адаптированные функции можно разделить на три группы:

- основные счетные функции – с наибольшей долей в решении СЛАУ;
- сервисные функции – с долей в решении СЛАУ, не превышающей 15 %;
- функции работы с АрУ (инициализация АрУ, создание, освобождение, копирование массивов на АрУ, загрузка исходных данных и конвертирование в подходящий для использования на АрУ формат, выгрузка результата).

1.4.3.1. Основные счетные функции

Декомпозиция по осям *X, Y, Z* – *LU*-разложение трехдиагональных матриц в произведение треугольных двухдиагональных матриц. Для параллельного вычисления исходные данные разбиваются на независимые подсистемы. *LU*-разложение использует метод крупнозернистого распараллеливания, где каждая подсистема вычисляется отдельным потоком АрУ. Для согласованного доступа к памяти АрУ входные данные по осям *X*, *Y*, *Z* приводятся к виду, указанному на рис. 1. Таким образом, все потоковые процессоры АрУ обращаются к непрерывной области памяти. Функция выполняется только при инициализации и не повторяется на итерациях.

Рис. 1. Распределение матричных элементов в массиве для согласованного доступа к памяти: а − элементы подсистем; б − расположение элементов для согласованного доступа

Все перечисленные ниже операции выполняются на итерациях.

Умножение матрицы на вектор – для распараллеливания было применено мелкозернистое распараллеливание, где каждый поток АрУ вычисляет свой элемент результирующего массива. Загрузка АрУ и, как следствие, количество блоков и потоков ограничены только размером результирующего массива, что позволяет эффективно использовать ресурсы АрУ.

```
// Счетная область 64×64. DiagInds состоит из: −64 −1 0 +1 +64
int i = blockDim.x * blockIdx.x + threadIdx.x;
if ( i < iBtmRow ) {
newtDOUBLE v0 = 0, v2 = 0, vs;
// Проверка на выход за границы массива. Если выходят, то присваиваем нулевое значение
v0 = A0[i] * load_from_texmem (texMatMult, DiagInds [0] + i);          // −64
iv2 = A2[i]* load_from_texmem (texMatMult, DiagInds [1] + I + 1−1);    // −1
// Производим операцию над основным блоком
vs = A1[i] * load_from_texmem (texMatMult, DiagInds [0]*(−1) + i) +    // +64
A4[i] * load_from_texmem (texMatMult, DiagInds [1] + i +1) +           // 0  общий индекс
A3[i] * load_from_texmem (texMatMult, DiagInds [1] + i + 1 + 1);       // +1
// Складываем результат
x [i] = vs + v0 + v3;  // (64+0+1) + (−64) + (−1)
}
```

Высокая эффективность счета на АрУ во многом зависит от доступа к памяти. Матричные элементы хранятся по диагоналям, что позволяет организовать для них согласованный доступ (см. рис. 2).

Как показано на рис. 3, некоторые данные, участвующие в вычислении одного потока, используются повторно в других потоках. Для этого случая наиболее эффективно использовать текстурную память, позволяющую КЭШировать повторяющиеся запросы [4].

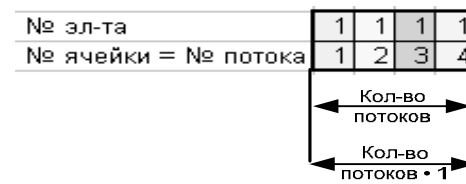


Рис. 2. Распределение элементов в массиве матричных элементов для согласованного доступа к памяти функции умножения матрицы на вектор

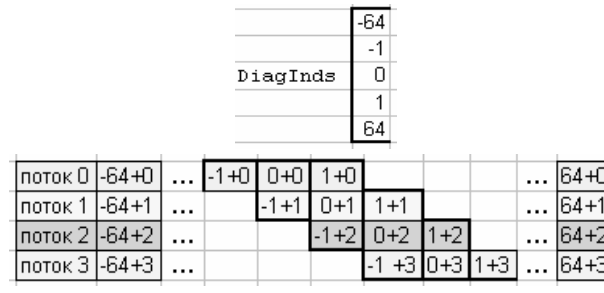


Рис. 3. Вверху: массив диагональных индексов матрицы, порождаемой задачей на счетной области размером 64×64 и пятиточечной разностной схемой. Внизу: схема повторного использования данных потоковыми процессорами

Решение систем с предобусловливателем состоит из последовательного вызова процедур решения СЛАУ с двухдиагональными матрицами по каждому пространственному направлению. Распараллеливание функции решения систем с предобусловливателем осуществляется с помощью крупнозернистого параллелелизма. Каждая подсистема − один поток на АрУ. Шаблон доступа к памяти процедуры решения систем с предобусловливателем идентичен описанному выше шаблону функции декомпозиции.

На этом шаге счета не требуется переупорядочивания данных для согласованного доступа. Результат выполнения предыдущей процедуры записан в соответствии с требованиями согласованного доступа.

Вычисление скалярного произведения (*Ddot*) и обновление вектора (*Daxpy*) − функции входят в состав библиотеки BLAS. Используются адаптированные под АрУ аналоги из библиотеки NVIDIA CUBLAS [5], поставляемой вместе с CUDA.

1.4.4. Методы обмена данными между адаптированными под АрУ функциями в модуле NEWT

Модуль NEWT является сложным программным продуктом. Для адаптации под счет на АрУ требовалось внести много изменений в отлаженный и взаимосвязанный код. Изменения касаются расширения количества массивов, создания новых переменных и флагов, а также «протягивания» до оптимизируемых функций новых данных «сквозь» весь модуль NEWT.

Для минимизации трудозатрат на модификацию модуля NEWT, придания коду гибкости и внесения минимальных модификаций в стабильную программу был применен новый подход. Уже описанные в

коде Фортран переменные используются как контейнеры для хранения указателей на данные, перенесенные в адресное пространство АрУ.

Для реализации подхода с подменой в массиве Фортран указателя на адрес области памяти в АрУ были найдены методы:

- создания массивов Фортран с указателями на структуры, загруженные в АрУ. Такие массивы идентичны созданным в Фортране, за исключением адреса начальной области памяти, который ссылается на память АрУ. Это позволяет не менять код, написанный на Фортране (проверки массивов и методы передачи массивов между функциями);
- получения из массива-контейнера, указателя на память АрУ для передачи в функции-интерфейсы Си.

2. Постановка и результаты задач на правильность решения

В первую очередь проверялась правильность решения на АрУ тестовых задач, имеющих точное решение. Для исследования как точности получаемого решения в программе расчета теплопроводности, так и эффективности модуля NEWT были подготовлены следующие модельные тестовые задачи:

- задача об остывании куба;
- задача о бегущей тепловой волне;
- стационарная двухобластная задача с функцией источника.

Постановки задач были взяты из [4]. Задачи представлялись на ортогональных счетных сетках с одинаковым разбиением по пространственным направлениям. Оператор диффузии аппроксимировался семиточечной разностной схемой.

Результаты в параллельном режиме должны совпадать (до заданной точности) с результатами, полученными в последовательном режиме счета. Серии расчетов проводились в следующих постановках:

- в последовательном режиме счета (далее − 1 CPU);
- в параллельном режиме счета с использованием АрУ в модуле NEWT (далее − CPU+GPU).

2.1. Задачи об остывании куба

Задача расчета поля температур, формирующегося при остывании куба единичного размера $0 \le x \le 1$; $0 \le y \le 1$; $0 \le z \le 1$, внутри которого задана температура $T(0, x, y, z) = 1$, а на всех границах во все последующие моменты поддерживается нулевая граничная температура $T_{гр}(t) = 0$ при $t \ge 0$.

Внутри куба содержится идеальный газ с уравнением состояния $E = T$ и коэффициентом теплопроводности, равным 1. Процесс распространения тепла в такой среде описывается трехмерным уравнением теплопроводности:

$$\frac{\partial T}{\partial t} = \frac{\partial^2 t}{\partial x^2} + \frac{\partial^2 T}{\partial^2 y} + \frac{\partial^2 T}{\partial^2 z}.$$

Точное решение этой задачи находится методом Фурье и имеет вид

$$T(x, y, t) = \psi(x,t)\psi(y,t)\psi(z,t);$$

$$\psi(x,t) = 4\sum_{k=0}^{\infty} \frac{e^{\pi^2(2k+1)^2 t}}{\pi(2k+1)} \sin \pi(2k=1)x.$$

Расчет на точность проводился на ортогональной сетке размером 70×70×70 (строк, столбцов и листов соответственно) до момента времени t = 0,08 с шагом по времени τ = 0,0005.

На рис. 4 показано сравнение численного и аналитического решений на момент времени t = 0,08 для результатов, полученных в последовательном (1 CPU) параллельном и режимах, при работе модуля NEWT на графических ускорителях (CPU+GPU).

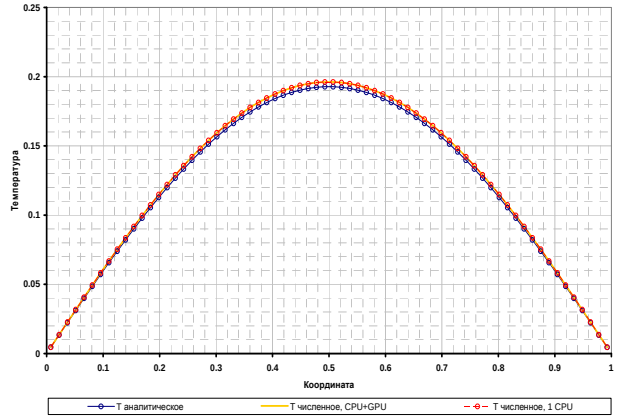
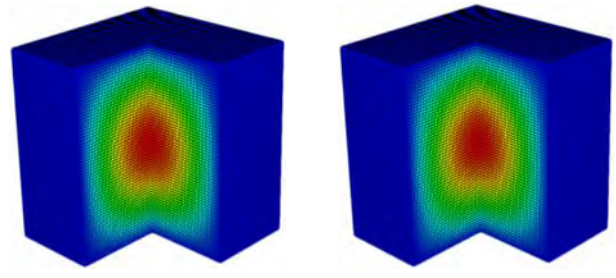


Рис. 4. Профиль температуры $T(t, x = 0{,}5; y = 0{,}5; z)$ для задачи об остывании куба

На рис. 5 приведены поля температур, полученные в различных режимах счета, на момент времени t = 0,08 для данной задачи.



а б

Рис. 5. Поле температур на момент времени t = 0,08 для задачи об остывании куба: а − 1 CPU; б − CPU+GPU

Численные эксперименты показали, что результаты, полученные как в последовательном, так и в параллельном режиме счета, совпадают (вплоть до заданной точности), что подтверждает правильность работы созданных программ.

2.2. Задача о бегущей волне

Задача расчета движения тепловой волны внутри единичного куба ($0 \le x \le 1$; $0 \le y \le 1$; $0 \le z \le 1$), заполненного идеальным газом ($\varepsilon = T$, $\rho = 1$), с коэффициентом теплопроводности $\chi = \chi_0 T^{\beta}$. В начальный момент времени газ имеет нулевую температуру. На одной из граней куба задано граничное условие $T_{\text{гр}}(t) = \left(\dfrac{\beta c^2}{\chi_0}\right)^{1/\beta}$.

У этой задачи существует автомодельное аналитическое решение, представляющее собой бегущую волну, которая распространяется вдоль оси y со скоростью c:

$$T(t,y) = \begin{cases} \left[\dfrac{\beta c}{\chi_0}(ct - y)\right]^{1/\beta}, & y < ct; \\ 0, & y \ge ct. \end{cases}$$

Для расчетов были взяты следующие значения параметров:

$$\chi_0 = 6, \quad \beta = 3, \quad c = 4.$$

Расчет на точность проводился на ортогональной сетке размером 70×70×70 (строк, столбцов и листов соответственно) до момента времени $t = 0{,}2$ с шагом по времени $\tau = 0{,}01$.

На рис. 6 показано сравнение численного и аналитического решений на конечный момент времени для результатов, полученных в последовательном режиме (1 CPU) и при работе модуля NEWT на АрУ (CPU+GPU).

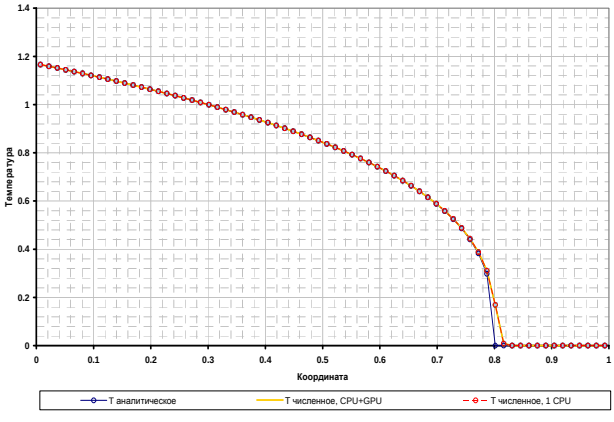
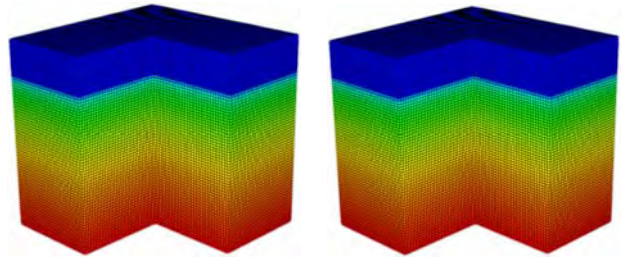


Рис. 6. Профиль температуры $T(t, x = 0{,}5, y, z = 0{,}5)$ для задачи о бегущей волне

На рис. 7 приведены поля температур, полученные в различных режимах счета, на конечный момент времени для данной задачи.

Как показали результаты численных экспериментов, результаты, полученные как в последовательном, так и в параллельном режиме счета, совпадают, что подтверждает правильность работы созданных программ.



а б

Рис. 7. Поле температур на момент времени $t = 0{,}2$ в задаче о бегущей волне: а − 1 CPU; б − CPU+GPU

2.3. Стационарная двухобластная задача с функцией источника

Шар радиуса $r = 2$ разделен сферой радиуса $r = 1$ на области I и II, заполненные разными, холодными в начальный момент времени веществами. В области I задано постоянное по времени энерговыделение $W = \dfrac{\partial E}{\partial T} = 6$. Сфера $r = 2$ является свободной границей. С течением времени в двухслойной сферической системе устанавливается стационарный тепловой режим с профилем температуры, подлежащим расчету. На рис. 8 приведена геометрия данной задачи.

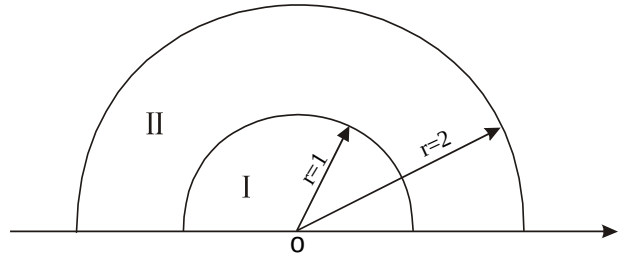


Рис. 8. Сечение геометрии стационарной двухобластной задачи с функцией источника

Уравнение теплопроводности для данного случая имеет вид $\rho_i \dfrac{\partial E}{\partial t} = \dfrac{1}{r^2}\dfrac{\partial}{\partial r}\left(r^2 \kappa_i \dfrac{\partial T}{\partial r}\right), \quad i = 1, 2.$

Параметры веществ в областях I и II определяются соотношениями

$$\rho_1 = 1; \quad E_1 = T; \quad \kappa_1 = \frac{\ell_1 C}{3} 4\sigma T^3; \quad \ell_1 = 1{,}82216 \cdot 10^{-4},$$

$$\rho_2 = 1; \quad E_2 = T; \quad \kappa_2 = \frac{\ell_1 C}{3} 4\sigma T^3; \quad \ell_1 = 1{,}45772 \cdot 10^{-3} T^4.$$

Расчет на точность проводился на ортогональной сетке размером 50×50×50 (строк, столбцов и листов соответственно) до момента времени $t = 1{,}6$ (именно на этот момент задача выходит на стационар) с шагом по времени $\tau = 0{,}01$.

На рис. 9 показано сравнение численного и аналитического решений на конечный момент времени для результатов, полученных в последовательном режиме (1 CPU) и при работе модуля NEWT на АрУ (CPU+GPU).

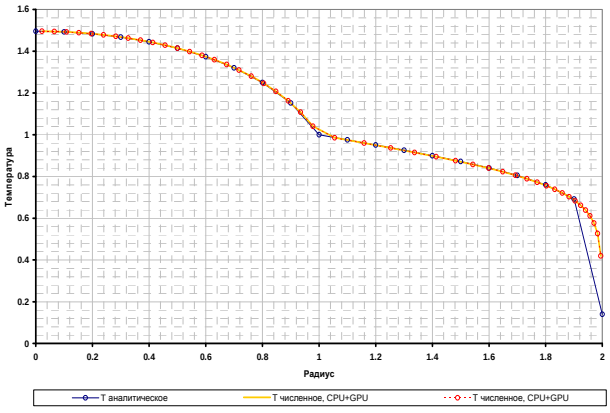
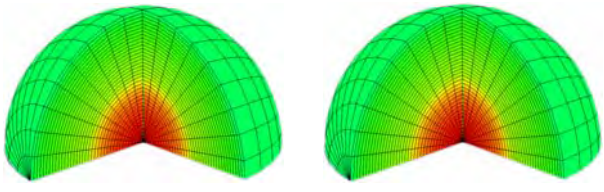


Рис. 9. Профиль температуры $T(t, x = 0.5, y, z = 0.5)$ для стационарной двухобластной задачи с функцией источника

На рис. 10 приведены поля температур, полученные в различных режимах счета, на конечный момент времени для данной задачи.



а б

Рис. 10. Поле температур на конечный момент времени стационарной двухобластной задачи с функцией источника: а − 1 CPU; б − CPU+GPU

Как видно из графиков и приведенных полей температур, результаты, полученные как в последовательном, так и в параллельном режиме счета, совпадают, что подтверждает правильность работы созданных программ.

3. Результаты расчетов

3.1. Состав программно-аппаратных средств

Тестовые расчеты проводились на вычислительной системе с процессором AMD Opteron 2431 2,41 ГГц и памятью 32 ГБ DDR2 и АрУ Tesla C2050.

Таблица 2

Спецификация АрУ Tesla C2050

Параметр	Значение
Графический процессор Количество ядер, шт. Тактовая частота ГП, ГГц	GF100 14 1,15
Оперативная память АрУ: − тип; − емкость АрУ, Гбайт; − максимальная пропускная способность в режиме ECC on, ГБайт/с	GDDR5 2,64 86
Теоретическая пиковая производительность с двойной точностью АрУ, ГФлоп/с	515,2

Результаты, полученные на АрУ, сравнивались с результатами вычисления тех же задач на одном ядре ЦП сервера. Модуль NEWT компилировался в 64-разрядном режиме (x64) компилятором Intel Visual Fortran, v.11.1 в режиме оптимизации -O2. Тестирование на АрУ проводилось со включенным режимом ECC (error-correcting code), что повышает стабильность вычислений, одновременно снижая пропускную способность памяти АрУ примерно на 20 %.

3.2. Результаты экспериментов

3.2.1. Расчеты для задачи остывания бесконечного куба квадратного сечения

Общий анализ результатов экспериментальных расчетов показал, что коэффициент ускорения от применения АрУ в модуле NEWT, в задаче остывания куба, в рамках программы расчета теплопроводности, напрямую зависит от размерности задачи. Чем больше размерность задачи, тем большее ускорение может быть достигнуто от применения АрУ (см. табл. 3). Ускорение счета СЛАУ на АрУ, относительно одного ядра ЦП, может достигать до 10,5 раз, а ускорение всей задачи − до 3,24 раза.

Таблица 3

Результаты расчетов для задачи остывания куба квадратного сечения

	CPU		GPU		CPU/GPU
	%	sec	%	sec	ускорение
20×20×20					
Задача теплопроводности		2,1		3,2	0,66
Решение СЛАУ в рамках задачи	61,5	1,3	73,5	2,3	0,57
50×50×50					
Задача теплопроводности		132,4		104	1,28
Решение СЛАУ в рамках задачи	33,1	43,8	14	14,2	3,08
70×70×70					
Задача теплопроводности		123,8		38,2	3,24
Решение СЛАУ в рамках задачи	72	89,1	22,3	8,5	10,48

3.2.2. Результаты расчетов для задачи о бегущей тепловой волне

Ускорение счета СЛАУ на АрУ относительно одного ядра ЦП может достигать до 9,21 раз, а ускорение всей задачи − до 3,33 раза.

Таблица 4

Результаты расчетов для задачи о бегущей тепловой волне

	CPU		GPU		CPU/GPU
	%	sec	%	sec	ускорение
20×20×20					
Задача теплопроводности		15,60		20,90	0,75
Решение СЛАУ в рамках задачи	51,10	8,00	62,30	13,00	0,62
50×50×50					
Задача теплопроводности		895,30		422,00	2,12
Решение СЛАУ в рамках задачи	66,60	596,50	28,90	122,10	4,89
70×70×70					
Задача теплопроводности		3911,20		1413,20	2,77
Решение СЛАУ в рамках задачи	71,70	2802,60	21,50	304,30	9,21

3.2.3. Расчеты для стационарной двухобластной задачи с функцией источника

Как и в предыдущих задачах коэффициент ускорения от применения АрУ для задачи Тихомирова в рамках программы расчета теплопроводности напрямую зависит от размерности задачи. Чем больше размерность задачи, тем большее ускорение может быть достигнуто от применения АрУ (см. табл. 5). Ускорение счета СЛАУ на АрУ относительно одного ядра ЦП может достигать до 12,33 раз, а ускорение всей задачи − до 7,45 раза.

Сравнение времени счета показало, что задачи малой размерности (20×20×20) на АрУ считаются в полтора-два раза медленнее, чем на ЦП. Это объясняется тем, что в этих задачах общее число потоков (= 20×20 = 400) слишком мало для эффективной загрузки конвейеров.

Использование АрУ заметно ускоряет счет задач большой размерности. Решение СЛАУ в задаче расчета поля температур остывающего куба и решение СЛАУ в задаче о бегущей тепловой волне размерностью 70×70×70 считаются на АрУ в 9−10 раз быстрее по сравнению с последовательной версией модуля

Таблица 5

Стационарная двухобластная задача с функцией источника

	CPU		GPU		CPU/GPU
	%	sec	%	sec	ускорение
20×20×20					
Задача теплопроводности		985,4		1277,9	0,77
Решение СЛАУ в рамках задачи	67,20	661,9	75,10	959,4	0,69
50×50×50					
Задача теплопроводности		10525,6		2344,0	4,49
Решение СЛАУ в рамках задачи	89,80	9456,4	60,50	1419,2	6,66
70×70×70					
Задача теплопроводности		153964,4		20662,2	7,45
Решение СЛАУ в рамках задачи	94,20	145073,0	56,90	11763,6	12,33

Заключение

Версия модуля NEWT, имеющая в своем составе реализацию метода переменных направлений на языке CUDA для счета на одном АрУ, подключена к программе решения уравнения теплопроводности.

Для всех трех тестируемых задач получено, что распределение температур на АрУ совпадает с распределением, полученным на процессоре общего назначения NEWT. Доля затрат модуля NEWT в общем балансе времени счета каждой из этих двух задач снизилась примерно с 70 до 20 %, а время счета задач уменьшилось примерно в 3 раза. При счете стационарной задачи Тихомирова размерностью 70×70×70 использование АрУ уменьшило время решения СЛАУ в 12 раз, снизило долю решателя с 94 до 60 %, а общее время счета задачи уменьшило в 7,5 раза.

Литература

1. Цилькер Б. Я., Орлов С. А. Организация ЭВМ и систем. М.: Питер, 2004.

2. Naumov M. Parallel Solution of Sparse Triagular Linear Systems in the Preconditioned Iterative Methods on the GPU. NVIDIA Technical Report NVR-2011-001, June 2011.

3. Owe Axelsson. Iterative Solution Methods. Cambridge University Press, NY, 1994, 1996.

4. Бондаренко Ю. А., Воронин Б. Л., Горев В. В. и др. Описание набора тестов для методик и программ, предназначенных для решения двумерных задач теплопроводности // Вопросы атомной науки и техники. Сер. Математическое моделирование физических процессов. 1992. Вып. 2. С. 14−20.

5. NVIDIA CUDA Compute Unified Device Architecture CUBLAS CUDA Basic Linear Algebra Subprograms Library. Programming Guide Ver. 4.0. 2/2011 NVIDIA Corporation, 2011.