УДК 517.9
DOI 10.53403/9785951505071_2022_473

АНАЛИЗ РЕЗУЛЬТАТОВ РАСПАРАЛЛЕЛИВАНИЯ ГРАДИЕНТНОГО АЛГОРИТМА РЕШЕНИЯ СИСТЕМ НЕЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ

В. Г. Суфиянов, А. В. Новиков

Ижевский государственный технический университет им. М.Т. Калашникова, Ижевск

Целью данной работы – распараллеливание градиентного алгоритма решения СНАУ на GPU и анализ результатов решения СНАУ в зависимости от размерности задачи.

Ключевые слова: СНАУ, СЛАУ, распараллеливание градиентногго алгоритма, ArrayFire.

Многие практические задачи сводятся к решению систем нелинейных алгебраических уравнений (СНАУ). Например, задача о химически равновесном составе продуктов горения топлива [1] решается в процессе численного моделирования процессов горения топлива в ракетных двигателях. При этом необходимо многократно решать уравнения в каждой ячейке сетки и на каждом временном слое, что при большом числе ячеек ($>10^6$) является проблемой, так как требуется значительные вычислительные мощности.

Основным методом решения систем нелинейных уравнений является градиентный метод методом Ньютона, в котором на каждой шаге итерации решается система линейных алгебраических уравнений (СЛАУ). Метод Ньютона обладает вычислительной неустойчивостью при большом значении числа обусловленности матрицы Якоби. Сходимость алгоритма и время решения системы нелинейных уравнений зависит от выбора начального приближения [2]. Наряду с градиентными методами используются и другие методы, например, метод простой итерации, генетические алгоритмы [3] и другие, которые также обладают рядом недостатков.

Одним из способов повышения скорости решения систем СНАУ является распараллеливание вычисления на многопроцессорных ЭВМ. Современное параллельное программирование для высокопроизводительных систем подразумевает многопоточность для многоядерных процессоров и ускорителей, которые могут быть реализованы, например, с помощью технологий MPI, OpenMP, OpenCL для центральных процессоров (CPU) и CUDA, OpenCL, OpenACC для графических процессоров (GPU).

В данной работе использовалась универсальная библиотека ArrayFire с открытым исходным кодом на языке C++, которая упрощает процесс разработки программного обеспечения, ориентированного на параллельные и массивно-параллельные архитектуры, включая центральные процессоры, графические процессоры и другие устройства. В библиотеке ArrayFire реализованы возможности решения задач линейной алгебры на параллельных системах, например, арифметические действия с матрицами и векторами, транспонирование, нахождении обратных матриц, вычисление определителей, решение СЛАУ и др. Одной из важных особенностей библиотеки ArrayFire является возможность написания одного кода программы на языке C++ для различных платформ CUDA и OpenCL.

Проведено исследование времени решения СНАУ градиентным методом на тестовом примере в зависимости от размерности задачи. В результате проведенных вычислений показано, что распараллеливание, с использованием библиотеки ArrayFire, позволяет значительно сократить время решения СНАУ с использованием GPU. «Узким» местом при решение этих задач является процедура

передачи данных из оперативной памяти на вычислительные устройства. Время на передачу данных в некоторых случаях сопоставимо с решением самой задачи. Решением данной проблемы является сокращение передачи данных из оперативной памяти на вычислительные устройства за счет вычислений векторов и матриц непосредственно на GPU.

1. Разработка градиентного алгоритма решения СНАУ

Рассмотрим систему нелинейных уравнений в общем виде

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0, \\ f_2(x_1, x_2, \ldots, x_n) = 0, \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0, \end{cases} \quad (1)$$

где $x_1, x_2, \ldots, x_n$ – искомые переменные; n – размерность задачи.

Перепишем систему (1) в векторном виде

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_n(\mathbf{x}) \end{pmatrix} = \mathbf{0}, \quad (2)$$

где $\mathbf{f}(\mathbf{x})$ – вектор-функция; $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ – вектор искомых переменных; $\mathbf{0} = (0, 0, \ldots, 0)^T$ – нулевой вектор.

При нахождении достаточно хорошего начального приближения к решению системы уравнений (1) эффективным методом решения задачи является метод Ньютона, который можно представить в виде

$$\mathbf{x}^{k+1} = \mathbf{F}(\mathbf{x}^k)$$

с начальным приближением

$$\mathbf{x}^0 = (x_1^0, x_2^0, \ldots, x_n^0)^T,$$

где вектор-функция $\mathbf{F}(\mathbf{x})$ имеет вид

$$\mathbf{F}(\mathbf{x}) = \mathbf{x} - \mathbf{J}^{-1}\mathbf{f}(\mathbf{x}); \quad (3)$$

$\mathbf{J} = \left[\dfrac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}\right]$ – матрица первых производных (матрица Якоби) функций $\mathbf{f}(\mathbf{x})$ системы нелинейных уравнений (2) по элементам вектора $\mathbf{x}$.

Итерационный процесс представляется в виде:

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \left[\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}\right]^{-1}_{\mathbf{x}=\mathbf{x}^t} \mathbf{f}(\mathbf{x}^t). \quad (4)$$

Согласно [4] система (4) представляется в виде системы линейных алгебраических уравнений (СЛАУ):

$$\left[\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}^t} \Delta^{t+1} = -\mathbf{f}(\mathbf{x}^t), \quad (5)$$

где $\Delta^{t+1} = \mathbf{x}^{t+1} - \mathbf{x}^t$ – разность между значениями приближениями на текущей и предыдущей итерациях. Решение СЛАУ (5) находится методом Гаусса, LU– или QR-разложения, итераций Зейделя и др. Таким образом, после вычисления разности Δ^{t+1} текущее приближение определяется по формуле

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \Delta^{t+1}.$$

Одно из следующих условий позволяет завершить итерационный процесс:

– число итераций $t > \mathrm{MaxIter}$, где $\mathrm{MaxIter}$ – максимальное число итераций;

– норма приращения $\left\|\Delta^{t+1}\right\| < \varepsilon_x$, где ε_x – минимальное значение приращения аргумента;

– норма вектор функции $\left\|\mathbf{f}\left(\mathbf{x}^{t+1}\right)\right\| < \varepsilon_f$, где ε_f – минимальное отклонение функции от нулевого вектора.

Ускорение решения систем уравнений на GPU требует представления системы уравнений в векторно-матричном виде. В общем случае решение системы нелинейных уравнений (2) достаточно сложно привести к векторно-матричному виду, т.к. требуется вычислять нелинейные функции и матрицу Якоби. В случае СНАУ представим функции в виде

$$f_i(x_1,x_2,\ldots,x_n) = \sum_{j=1}^{k} a_{ij} \cdot \varphi_j(x_1,x_2,\ldots,x_n) - b_i, \tag{6}$$

где $\varphi_j(x_1,x_2,\ldots,x_n) = x_1^{p_{j1}} \cdot x_2^{p_{j2}} \cdot \ldots \cdot x_n^{p_{jn}}$ – слагаемые функции, соответствующие произведению искомых переменных в степени p_{jl}, $l = 1,2,\ldots,n$; k – количество различных слагаемых φ_j в СНАУ; a_{ij}, b_i – константы СНАУ, $i = 1,2,\ldots,n$, $j = 1,2,\ldots,k$.

Представим константы СНАУ в векторно-матричном виде:

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nk} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \tag{7}$$

Кроме того, представим степени слагаемых функций в виде матрицы

$$\mathbf{P} = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k1} & p_{k2} & \cdots & p_{kn} \end{pmatrix}. \tag{8}$$

Матрицу множителей функций φ_j представим в виде:

$$\mathbf{Y} = \mathbf{X}^{\mathbf{P}} = \begin{pmatrix} x_1^{p_{11}} & x_2^{p_{12}} & \ldots & x_n^{p_{1n}} \\ x_1^{p_{21}} & x_2^{p_{22}} & \ldots & x_n^{p_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{p_{k1}} & x_2^{p_{k2}} & \ldots & x_n^{p_{kn}} \end{pmatrix}, \tag{9}$$

тогда вектор $\mathbf{y}$ значений функций $y_j = \varphi_j(x_1,x_2,\ldots,x_n)$, примет вид:

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{pmatrix} = prod_2(\mathbf{Y}) = \begin{pmatrix} x_1^{p_{11}} \cdot x_2^{p_{12}} \cdot \ldots \cdot x_n^{p_{1n}} \\ x_1^{p_{21}} \cdot x_2^{p_{22}} \cdot \ldots \cdot x_n^{p_{2n}} \\ \vdots \\ x_1^{p_{k1}} \cdot x_2^{p_{k2}} \cdot \ldots \cdot x_n^{p_{kn}} \end{pmatrix}, \tag{10}$$

где $prod_2(\mathbf{Y})$ – оператор, соответствующий произведению элементов матрицы $\mathbf{Y}$ по строкам.

В результате получим векторно-матричное представление для вычисления вектор-функции $\mathbf{f}(\mathbf{x})$:

$$\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{y} - \mathbf{b}. \tag{11}$$

Аналогичным образом запишем матрицу Якоби:

$$\mathbf{J} = \left[\frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}\right] = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \ldots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \ldots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \frac{\partial f_n(\mathbf{x})}{\partial x_2} & \ldots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{pmatrix}, \tag{11}$$

где элементы матрицы Якоби можно представить в виде:

$$\frac{\partial f_i(\mathbf{x})}{\partial x_j} = \sum_{l=1}^{k} a_{il} \cdot \frac{\partial \varphi_l((\mathbf{x}))}{\partial x_j}. \tag{12}$$

Вычисление производных слагаемых можно представить в следующем виде

$$\frac{\partial \varphi_l(\mathbf{x})}{\partial x_j} = \begin{cases} 0, & p_{lj} = 0, \\ p_{lj} \cdot x_1^{p_{l1}} \cdot x_2^{p_{l2}} \ldots \cdot x_j^{p_{lj}-1} \cdot \ldots \cdot x_n^{p_{ln}}, & p_{lj} \neq 0. \end{cases} \tag{13}$$

Следующее представление производной (13) позволяет сократить затраты на ее вычисление

$$\frac{\partial \varphi_l(\mathbf{x})}{\partial x_j} = \begin{cases} 0, & p_{lj} = 0, \\ p_{lj} \cdot \dfrac{\varphi_l(\mathbf{x})}{x_j}, & p_{lj} \neq 0, \end{cases} \tag{14}$$

где $y_l = \varphi_l(\mathbf{x})$ – известное значение слагаемого, но, в этом случае, возникают сложности при вычислении функций при $x_j \approx 0$ и необходимо устанавливать дополнительные условия проверки точности вычислений.

В результате каждый столбец матрицы Якоби можно вычислить в векторно-матричном виде:

$$\mathbf{J}(:, j) = \mathbf{A} \cdot \begin{pmatrix} \dfrac{\partial \varphi_1(\mathbf{x})}{\partial x_j} \\ \dfrac{\partial \varphi_2(\mathbf{x})}{\partial x_j} \\ \vdots \\ \dfrac{\partial \varphi_k(\mathbf{x})}{\partial x_j} \end{pmatrix}. \tag{15}$$

где $\mathbf{J}(:, j)$ – соответствует j -ому столбцу матрицы Якоби.

Таким образом, получаем полностью векторно-матричный алгоритм решения СНАУ методом Ньютона, который реализуется программными средствами распараллеливания алгоритмов, например, для GPU – это технологии CUDA или OpenCL.

2. Анализ результатов решения СНАУ на GPU

Рассмотрим тестовый пример для решения СНАУ вида

$$\begin{cases} x_1^2 + x_2^2 + \ldots + x_n^2 - n = 0, \\ x_1 - x_2 = 0, \\ x_2 - x_1 = 0, \\ \ldots \\ x_{n-1} - x_n = 0. \end{cases} \tag{16}$$

где размерность задачи $n \geq 2$.

Система (16) имеет два точных решения:

$$\mathbf{x}_1^* = \begin{pmatrix} -1 \\ -1 \\ \vdots \\ -1 \end{pmatrix}, \quad \mathbf{x}_2^* = \begin{pmatrix} +1 \\ +1 \\ \vdots \\ +1 \end{pmatrix}. \tag{17}$$

В случае $n = 2$ решение можно представить графически (рис. 1).

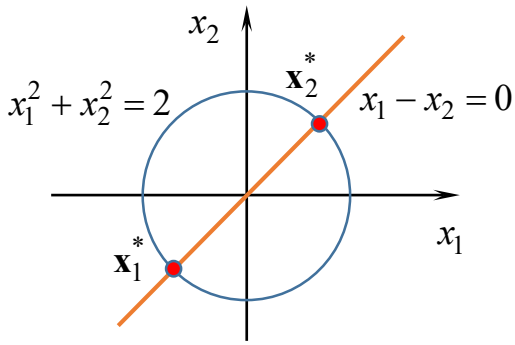

Рис. 1. Решение СНАУ для тестового примера при n=2

На рис. 1 представлено очевидные решения $\mathbf{x}_1^* = (-1, -1)$ и $\mathbf{x}_2^* = (+1, +1)$ системы двух уравнений, первое представляет собой окружность радиуса $\sqrt{2}$ с центром в начале координат и прямую, проходящую через начало координат.

Константы тестовой СНАУ в векторно-матричном представлении имеют вид

$$\mathbf{A} = \left(\begin{array}{cccccc|cccccc} 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -1 & \cdots & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 1 & -1 & 0 & 0 & 0 & \cdots & 0 & 0 \end{array}\right), \quad \mathbf{b} = \begin{pmatrix} -n \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}. \tag{17}$$

Матрица степеней слагаемых функций записывается в виде

$$\mathbf{P} = \left(\begin{array}{cccc} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \\ \hline 2 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2 \end{array}\right). \tag{18}$$

В качестве начального приближения принималось значение:

$$\mathbf{x}^0 = \begin{pmatrix} 2 \\ 3 \\ \vdots \\ n+1 \end{pmatrix}. \tag{18}$$

Решение тестовой СНАУ осуществлялось в операционной системе Windows 10 64 bit. В качестве интегрированной среды разработки использовался MS Visual Studio 2019. Программа была написана на языке программирования C++ с использованием библиотеки ArrayFire v. 3.8.0 и технологии OpenCL 2.0.

Тестирование программы осуществлялось на 4 ядерном CPU Intel Core i7 2600 с частотой 3 400 МГц, объем оперативной памяти 16 Гб. В качестве вычислительного устройства использовался

графический процессор AMD Radeon VII с числом ядер – 3840 и базовой частота 1 400 МГц, объем графической память HBM2 составила 16 Gb.

В качестве условия выхода из итераций задавались два условия: максимальное число итераций задавалось равным $\mathrm{MaxIter} = 100$ и минимальное значение приращение аргумента $\varepsilon_x = 10^{-5}$.

Тестовая задача решалась при заданной размерности $n = 2, 10, 20, \ldots, 500$. Зависимость числа итераций от размерности решаемой тестовой СНАУ представлена на рис. 2.

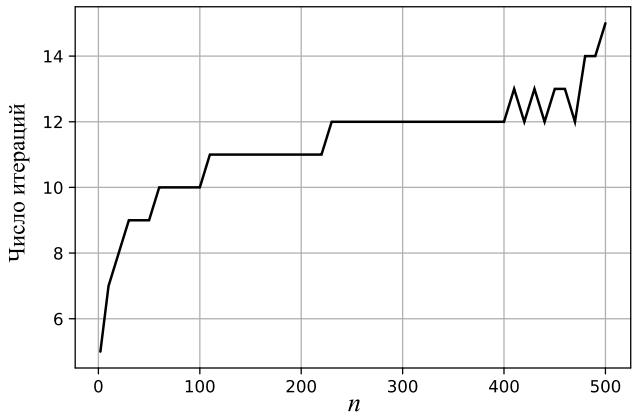

Рис. 2. Число итераций при решения тестовой СНАУ
в зависимости от размерности задачи n

Как видно из рис. 2, число итераций, требуемое для достижения заданной точности нелинейно растет с увеличением размерности тестовой задачи. Зависимость погрешности в логарифмических координатах от числа итераций для рассматриваемой тестовой задачи размерности $n = 500$ представлена на рис. 3.

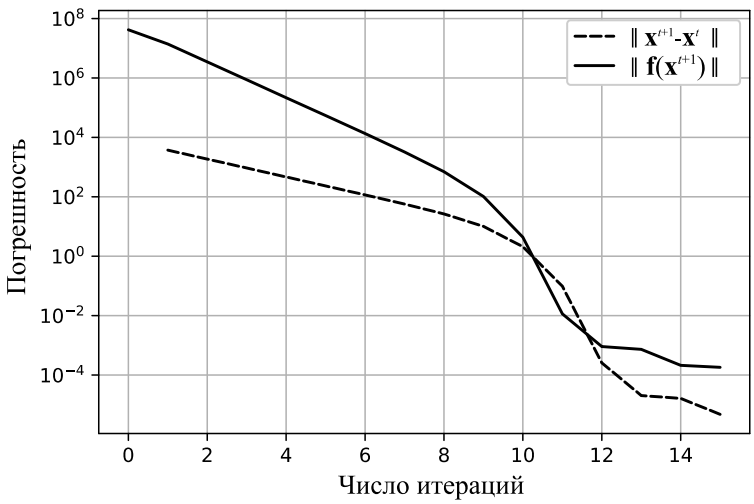

Рис. 3. Сходимость к точному решению от числа итераций
для тестовой СНАУ при $n = 500$

Как видно из рис. 3, значение приращения аргумента $\left\|\Delta^{t+1}\right\| = \left\|\mathbf{x}^{t+1} - \mathbf{x}^t\right\|$ и норма вектора функций $\left\|\mathbf{f}\left(\mathbf{x}^{t+1}\right)\right\|$ имеет одинаковый нелинейный характер убывания.

Зависимость времени решения тестовой СНАУ на CPU представлены на рис. 4 и на GPU – на рис. 5.

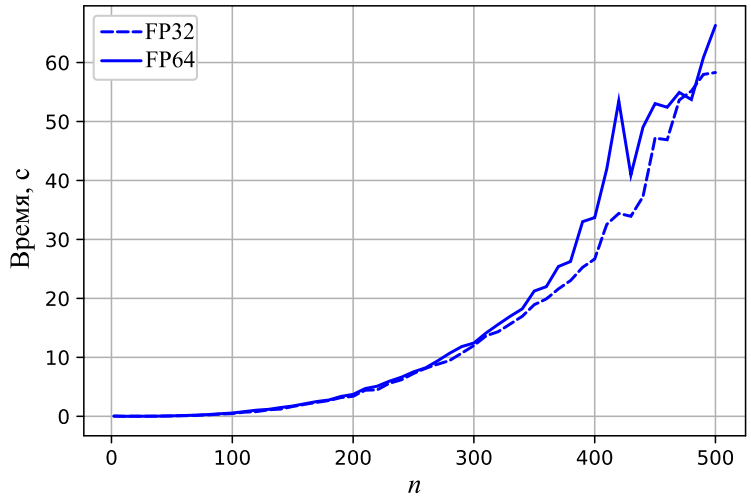

Рис. 4. Зависимость времени расчетов на CPU от размерности задачи с одинарной (FP32) и двойной точностью (FP64)

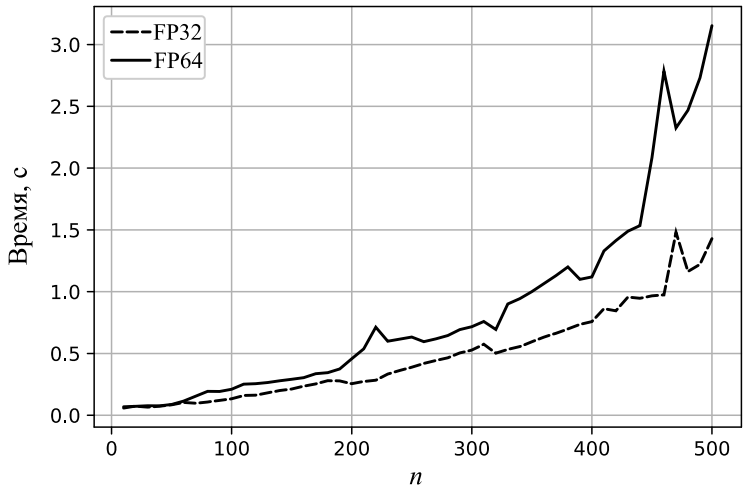

Рис. 5. Зависимость времени расчетов на GPU от размерности задачи с одинарной (FP32) и двойной точностью (FP64)

Из представленных выше рис. 4 и 5 видно, что время решения тестовой задачи растет нелинейно с увеличением размерности задачи. Сравнение рис. 4 и 5 показывает, значительное уменьшение времени решения тестовой СНАУ на GPU по сравнению с решением на CPU. Так, максимальное время расчета с одинарной точностью (FP32) на CPU составило 58,32 с, а на GPU – 1,43 с, т. е. время расчета уменьшилось в 40,8 раз. Максимальное время расчета с двойной точностью (FP64) на CPU составило 66,31 с, а на GPU – 3,15 с, т. е. время расчета уменьшилось в 21,0 раз.

Таким образом, можно сделать вывод, что значительный прирост производительности достигается за счет использования GPU с большим объемом памяти и большим числом вычислительных ядер. При этом для рассматриваемых вычислительных средств решение задачи с одинарной точностью (FP32) почти в 2 раза превосходит эффективность решения задачи с двойной точностью (FP64).

Анализ временных затрат на загрузку исходных данных на вычислительные устройства показал, что они весьма незначительные по сравнению со временем решения задачи.

Заключение

Таким образом, разработан обобщенный алгоритм решения системы СНАУ на графических процессорах с использованием библиотеки ArrayFire© и технологии параллельного программирования OpenCL. Исследование скорости сходимости алгоритма показало, что с увеличением размерности задачи увеличивается число итераций, а также наблюдается нелинейная сходимость к решению СНАУ.

Проведено исследование временных затрат на решение в зависимости от размерности задачи на тестовом примере для задач различной размерности. Проведенный анализ временных затрат показал, что распараллеливание алгоритма на GPU позволяет уменьшить время решения СНАУ с одинарной точностью (FP32) в 40 раз по сравнению со временем решения на CPU и в 21 раз – при решении с двойной точностью (FP64).

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 20-01-00072

Литература

1.Алемасов В. Е., Ваничев А. П., Гришин С. Д., Ильинский В. А., Дрегалин А. Ф., Тишин А. П., Илларионов Н. В. Термодинамические и теплофизические свойства продуктов сгорания; под ред. Глушко В. П. В 5 томах. Т. 1. М.: ВИНИТИ АН СССР, 1971. С. 266.

2.Суфиянов В. Г., Новиков А. В. Распараллеливание градиентного алгоритма решения систем нелинейных уравнений с использованием библиотеки ArrayFire // Сборник материалов XII Всероссийского совещания, посвященного 100-летию Калашникова М. Т. (Ижевск, 16-189 октября 2018). С. 80–87.

3.Попова Е. В. Решение систем нелинейных алгебраических уравнений гибридными генетическими алгоритмами // Интеллектуальные системы в производстве. 2011. № 2 (18). С. 33–40

4.Ортега Дж., Рейнболдт В. Итерационные методы решения нелинейных систем уравнений со многими неизвестными. –М.: Мир, 1975. –С. 560 .

EVALUATION OF THE EFFECT OF PARALLELING THE GRADIENT SOLVER FOR SYSTEMS OF NONLINEAR ALGEBRAIC EQUATIONS

V. G. Sufiyanov, A. V. Novikov

Kalashnikov Izhevsk State Technical University, Izhevsk

The objective of this work is to parallelize the gradient solver for GPU-assisted computations of systems of nonlinear algebraic equations and to evaluate the results of such computations as a function of problem dimensionality.

Key words: system of nonlinear algebraic equations, system of linear algebraic equations, parallelization of gradient solver, ArrayFire.