УДК 539.3

С.С. ЩЕРБАКОВ, д-р физ.-мат. наук, проф.
заместитель Председателя[1]
профессор кафедры теоретической и прикладной механики[2]
E-mail: sherbakovss@mail.ru

М.М. ПОЛЕЩУК
аспирант[2]
E-mail: mikhailpaliashchuk@yandex.ru

[1]Государственный комитет по науке и технологиям Республики Беларусь, г. Минск, Республика Беларусь
[2]Белорусский государственный университет, г. Минск, Республика Беларусь

Поступила в редакцию 10.09.2019.

УСКОРЕНИЕ ГРАНИЧНО-ЭЛЕМЕНТНЫХ РАСЧЕТОВ С ПОМОЩЬЮ ГРАФИЧЕСКОГО АКСЕЛЕРАТОРА ДЛЯ ЭЛЕМЕНТОВ С НЕЛИНЕЙНЫМИ ФУНКЦИЯМИ ФОРМЫ

В работе рассмотрена реализация метода граничных элементов (МГЭ) для определения распределения потенциала в полуплоскости с использованием трех нелинейных функций формы. Также обеспечено ускорение соответствующих вычислений за счет их распараллеливания с использованием технологии NVidia CUDA. Построены сравнительные графики точности расчетов при использовании равномерных и нелинейных функций формы, а также графики зависимости точности от дискретизации поверхности полуплоскости. Представлена методика распараллеливания МГЭ на графическом процессоре. Построены зависимости времени вычислений, а также коэффициента ускорения от количества граничных элементов и расчетных узлов для последовательного и параллельных расчетов.

Ключевые слова: *МГЭ, ускорение вычислений, распределение потенциала, моделирование, графический акселератор, CUDA, распараллеливание*

Введение. В настоящее время большое количество задач в различных областях теоретических и прикладных наук решается с применением метода конечных элементов (МКЭ) или же пакетов инженерного моделирования, использующих данный метод. Подобные расчеты требуют больших вычислительных возможностей и могут длиться десятки часов. Применение МГЭ слабо распространено среди популярных пакетов моделирования в связи с его алгоритмической сложностью. Однако для ряда задач гранично-элементное моделирование с использованием технологии CUDA может быть эффективной альтернативой МКЭ [3–11].

Постановка задачи. Рассмотрим полуплоскость $A = \{(x, y): (x, y) \in R^2, y \le 0\}$. Прямая $y = 0$ является границей расчетной области, а граничным условием будет распределение потенциала

$$p^*(x) = p_0\sqrt{1-\frac{x^2}{b^2}} \qquad (1)$$

на отрезке поверхности $[-b, b]$, где $b \in R$ (рисунок 1) [16].

Целью данной работы является решение задачи об отыскании распределения потенциала под поверхностью полуплоскости, источником которого является потенциал на поверхности, с использованием распараллеливания вычислений. Для решения будем использовать метод функций влияния и сингулярное решение для неограниченного однородного полупространства в рамках решения непрямой задачи МГЭ.

Решение с применением нелинейных функций формы. Решение рассматриваемой задачи может быть получено с помощью МГЭ на основе задачи о действии сосредоточенного потенциала на полуплоскость [1–2]. Тогда поверхностное рас-

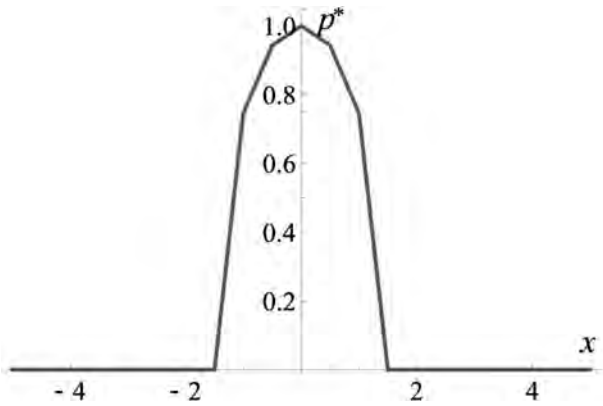


Рисунок 1 — Распределение граничного потенциала (граничное условие)
Figure 1 — Distribution of boundary potential (boundary condition)

пределение потенциала представимо в следующем виде:

$$p(x) = G(x,\xi) f(\xi), \qquad (2)$$

где

$$G = -\frac{1}{2\pi k} \ln\left|\frac{r}{r_0}\right|,$$

а константа r_0 такова, что $G = 0$ при $r = r_0$; f — функция формы.

Для применения МГЭ поверхность была разбита на N граничных элементов (ГЭ), полудлина которых равна h. Центры ГЭ расположены в точках x_i. При расчетах использовались три функции формы $f_1(x)$, $f_2(x)$, $f_3(x)$, которые распределялись на граничных элементах таким образом, как показано на рисунке 2.

Исходя из того, что (2) определена для точечного источника, то, чтобы получить влияние одного граничного элемента на точку наблюдения (x, y), требуется проинтегрировать (2) по длине граничного элемента [14, 15]:

$$IG_i^j(x,y) = p_i^j \int_{x_i-h}^{x_i+h} G(x-\xi, y) f_i^j(\xi) d\xi,$$

где $j = 1...3$ — номер функции на граничном элементе; $i = 1...N$ — номер граничного элемента; p_i^j — коэффициент функции формы; h — полудлина ГЭ; x_i — центр ГЭ.

Для соблюдения непрерывности распределения потенциала на поверхности в общих точках граничных элементов должно выполняться условие

$$p_i^3 = p_{i+1}^1. \qquad (3)$$

Тогда влияние одного ГЭ с представленными на нем функциями формы будет иметь вид:

$$SIG_i(x,y) = IG_i^1(x,y) + IG_i^2(x,y) + IG_i^3(x,y). \qquad (4)$$

Сумма (4) по всем граничным элементам даст значение потенциала в точке (x, y). Для нахождения коэффициентов (3) требуется составить и решить систему уравнений на границе области:

$$\sum_{i=1}^{N} SIG_i(x_q, 0) = p^*(x_q),$$
$$x_q \in \{x : a + ih,\ i = 1...2N-1\}, \qquad (5)$$

где a — левая граница первого ГЭ.

Найдя коэффициенты p_i^j и используя (4), можно получить распределение потенциала в точке (x, y).

Покажем, что использование нелинейных функций формы приводит к повышению точности

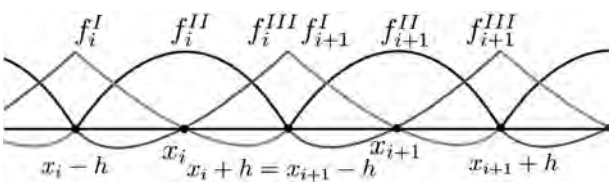


Рисунок 2 — Распределение нелинейных функций формы
Figure 2 — Distribution of nonlinear form functions

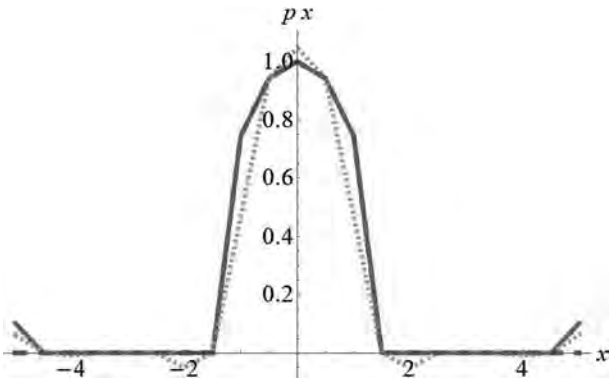


Рисунок 3 — Сравнение решений на поверхности:
пунктирная кривая — граничное условие; сплошная кривая — МГЭ с нелинейными функциями формы; кривая с мелким пунктиром — МГЭ с постоянными функциями формы
Figure 3 — Comparison of solutions on the surface:
dashed curve — boundary condition; solid curve — boundary element method with nonlinear form functions; dotted curve — boundary element method with constant form function

решения на границе, следовательно, и под поверхностью. Ниже приведены распределения потенциала на поверхности при использовании постоянных и нелинейных функций формы в сравнении с граничным условием (рисунок 3).

Относительная погрешность считалась в соответствии с приведенной ниже формулой:

$$e = \frac{p(x_i) - p^*(x_i)}{\max\left(p^*(x_i)\right)}. \qquad (6)$$

При использовании постоянной функции влияния погрешность составила 0,2 %, а при использовании нелинейных — 0,09 %.

Также стоит рассмотреть зависимость погрешности от дискретизации (рисунок 4). На рисунке 5 можно заметить, что при увеличении дискретизации наблюдается уменьшение шума в области, где граничное условие перестает быть нулевым, что влечет за собой снижение средней относительной погрешности.

Применение технологии CUDA для ускорения формирования матрицы взаимовлияния. Формирование матрицы взаимовлияния и вычисление распределения потенциала в полуплоскости являются процессами, которые состоят из независимых операций. Именно это свойство позволяет эффективно

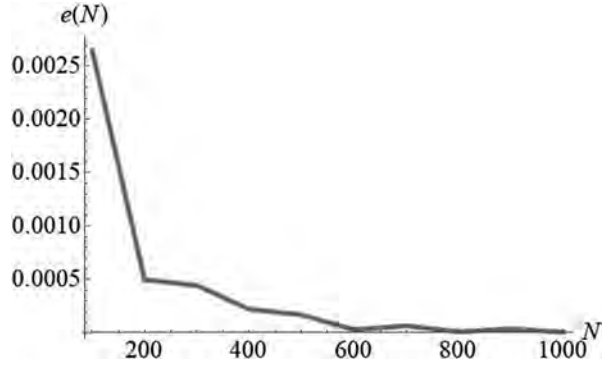


Рисунок 4 — Средняя погрешность
Figure 4 — Mean error

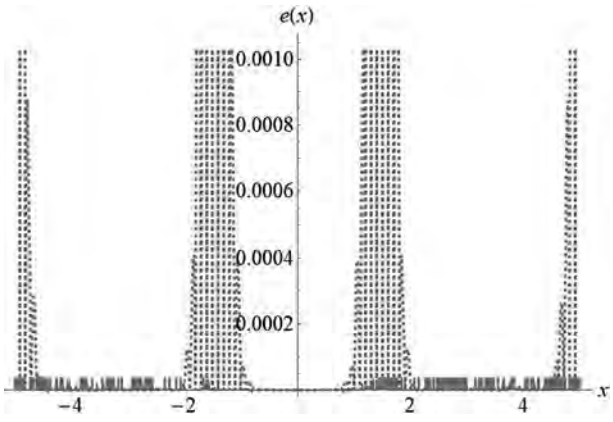


Рисунок 5 — Зависимость погрешности от координаты на границе расчетной области при $N = 100$
Figure 5 — Dependence of the error on the coordinate at the boundary of the calculated area at $N = 100$

применять CUDA и МГЭ. Таким образом, вычисляя каждую из таких операций в отдельном потоке, можно значительно сократить время расчетов.

Было проведено распараллеливание расчетов для разных дискретизаций поверхности $N = 100, 200, ... 1000$ и для разного количества расчетных узлов в полуплоскости $n = 100, 200, ... 1000$. Для параллельных расчетов применялся графический акселератор NVidia GTX 1070Ti, в то время как последовательный расчет проводился на процессоре Intel core i7-8700k. Следует отметить, что вычислительная сеть акселератора может максимально состоять из 65 535 блоков, в каждом из которых можно запустить максимум 1024 вычислительных потока, однако данные цифры могут отличаться на различных графических ядрах. Таким образом, имеется возможность запуска 67 107 840 параллельно работающих вычислительных потоков, что значительно превосходит возможности распараллеливания на центральных процессорах [12].

Рассмотрим систему уравнений (5) и запишем ее в матричном виде:

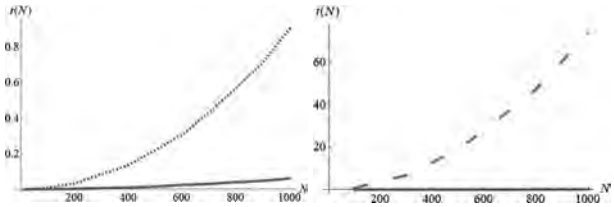
$$Ap = F, \tag{7}$$

где $A = \{a_{ij} : a_{ij} = SIG(x_i, 0)\}$ — матрица взаимовлияния; p — вектор неизвестных коэффициентов функций формы; $F = \{p^*(x_i)\}$ — множество значений потенциала на поверхности.

Стоит заметить, что на формирование такой системы требуется значительное количество времени, так как создается матрица, которая имеет размерность $N \times N$, где N — количество граничных элементов. Однако особенность этого процесса состоит в том, что каждый элемент этой матрицы можно вычислять независимо друг от друга, и, как следствие, этот процесс может быть эффективно распараллелен с помощью CUDA.

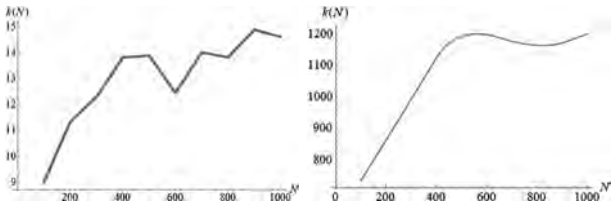
Создание $N \times N$ потоков, каждый из которых будет вычислять a_{ij}, заметно ускорило формирование матрицы взаимовлияния.

На рисунке 6 *a* приведена зависимость времени формирования этой матрицы от количества ГЭ



a *b*

Рисунок 6 — Время формирования матрицы взаимовлияния
Figure 6 — Time of formation of the matrix of mutual influence



a *b*

Рисунок 7 — Коэффициент ускорения: *a* — C++ относительно CUDA; *b* — Mathematica относительно CUDA
Figure 7 — Acceleration coefficient: *a* — C++ in relation to CUDA; *b* — Mathematica in relation to CUDA

для последовательной реализации (C++) и распараллеленной реализации алгоритма (CUDA). На рисунке 6 *b* представлена такая же зависимость для последовательной реализации на Wolfram Mathematica и реализации на CUDA. На рисунке 7 показаны зависимости коэффициентов ускорения от количества ГЭ.

Из рисунков 6 и 7 видно, что распараллеливание с помощью CUDA позволяет ускорить расчет матрицы взаимовлияния до 15 раз по сравнению с последовательным расчетом в C++ и до 1200 раз — в Mathematica.

Распараллеливание расчета распределения потенциала под поверхностью. Процесс вычисления потенциала в точке полуплоскости (x, y) также является независимым от вычисления потенциала в другой точке. Это позволяет создать n вычислительных потоков, где n — количество точек полуплоскости, для которой рассчитывается потенциал. Каждый такой поток будет вычислять потенциал в точке (x, y).

Исходя из проведенных расчетов, можно построить распределения потенциала на полуплоскости. Из рисунка 8 видно, что с увеличением количества расчетных узлов распределения потенциала становятся более гладкими, а при увеличении количества ГЭ это не наблюдается. Однако, как уже было сказано выше, при увеличении количества ГЭ повышается точность вычисления потенциала в точке.

Рассмотрим зависимости времени расчета потенциала (рисунок 9) и соответствующего коэффициента ускорения (рисунок 10) от количества расчетных узлов при фиксированном количестве граничных. Из рисунков 9 и 10 видно, что распараллеливание с помощью CUDA позволяет ускорить расчет распределения потенциала в по-

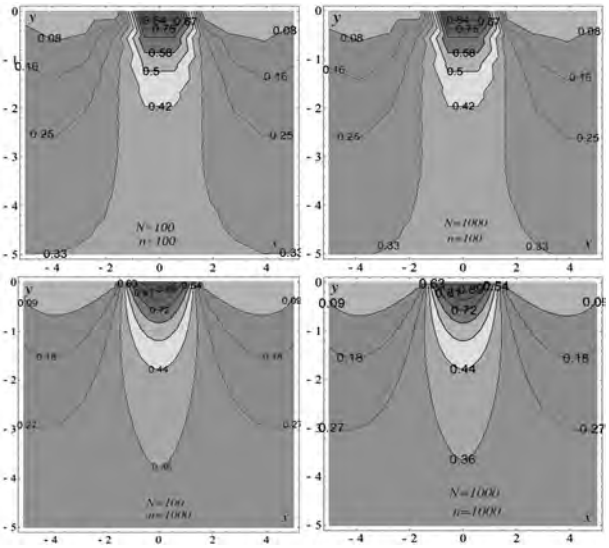


Рисунок 8 — Распределение потенциала в полуплоскости для разного количества граничных элементов и расчетных узлов
Figure 8 — Half-plane potential distribution for different number of boundary elements and nodal points

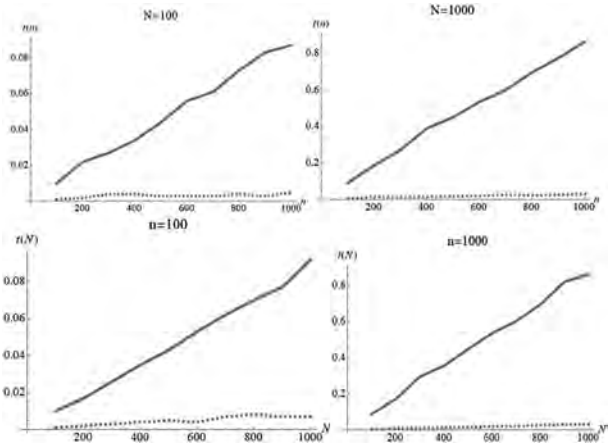


Рисунок 9 — Зависимости времени расчета распределения потенциала от количества граничных элементов и расчетных узлов
Figure 9 — Dependencies of computing time of potential distribution on the number of boundary elements and calculation points

луплоскости до 32 раз по сравнению с последовательным расчетом в C++.

Для более полного анализа полученные результаты можно представить в трехмерном виде (рисунки 11–15). В этом случает графики имеют вид поверхности, координатами которой являются количество расчетных узлов, количество граничных элементов и время расчета (коэффициент ускорения расчета) распределения потенциала для заданной дискретизации.

Из рисунков 11–15 видно, что распараллеливание с помощью CUDA позволяет ускорить расчет распределения потенциала в полуплоскости до 2800 раз по сравнению с последовательным расчетом в Mathematica.

Заключение. Был рассмотрен метод граничных элементов для нелинейных распределений функций влияния, а также особенности его реализа-

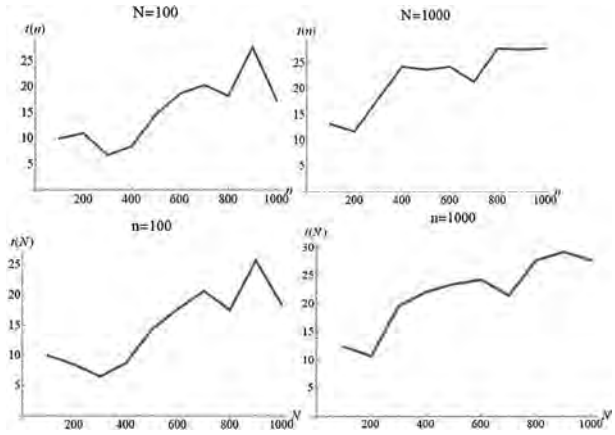


Рисунок 10 — Зависимости коэффициента ускорения расчета от количества граничных элементов и расчетных узлов
Figure 10 — Dependencies of computing acceleration coefficient on the number of boundary elements and calculation points

ции. Проведено распараллеливание расчетов МГЭ с помощью технологии CUDA на вычислительных потоках графического ускорителя. Последовательные вычисления проводились в пакете Wolfram Mathematica и в программе, написанной на C, а для распараллеливания была написана программа на языке C++. Были получены зависимости погрешности расчетов от количества граничных элементов. Построены зависимости времени последовательного и распараллеленного расчетов, а также коэффициентов ускорения вычислений от количества граничных элементов и количества

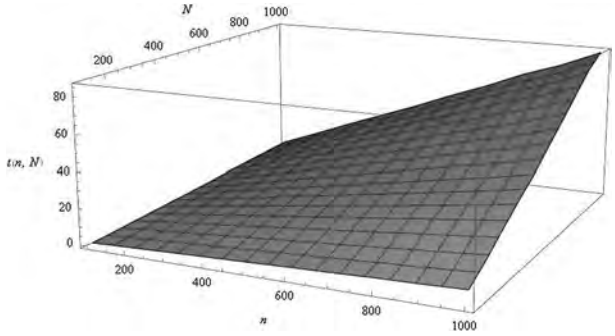


Рисунок 11 — Последовательная реализация (Wolfram Mathematica)
Figure 11 — Sequential implementation (Wolfram Mathematica)

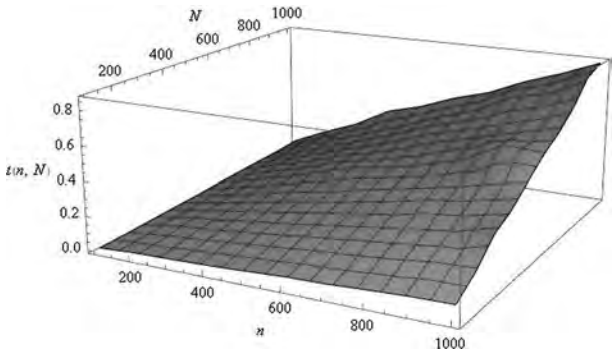


Рисунок 12 — Последовательная реализация (реализация на C++)
Figure 12 — Sequential implementation (implemetation on C++ platform)

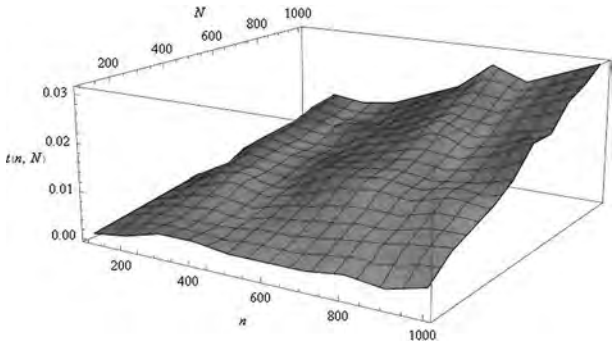


Рисунок 13 — Распараллеленная реализация
Figure 13 — Parallelized implementation

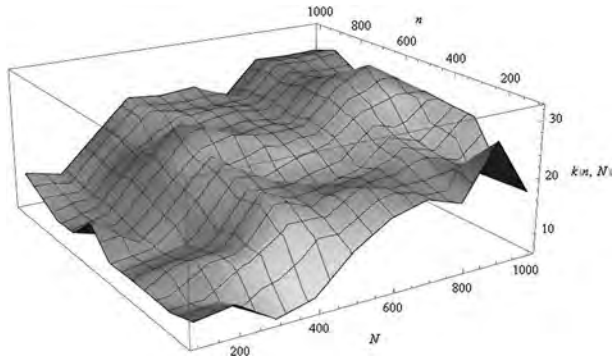


Рисунок 14 — Коэффициент ускорения для реализации на C++ относительно CUDA
Figure 14 — Acceleration coefficient for implementation on C++ platform in relation to CUDA

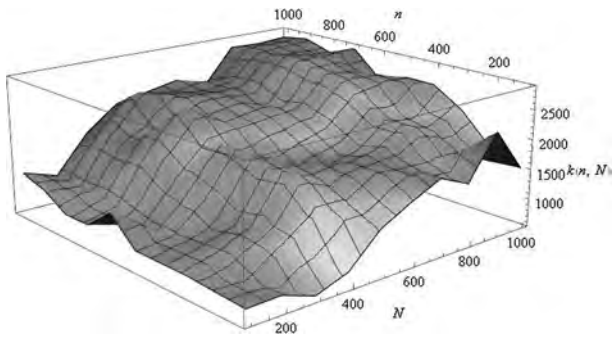


Рисунок 15 — Коэффициент ускорения для реализации на Wolfram относительно CUDA
Figure 15 — Acceleration coefficient for implementation on Wolfram platform in relation to CUDA

расчетных узлов под поверхностью полуплоскости. В результате распараллеливания относительно реализации на C достигнуто значительное (до 32 раз) ускорение расчета распределения потенциала в полуплоскости при сохранении его точности и до 2800 раз — для реализации на Wolfram Mathematica. Достигнуто ускорение соответственно до 15 и 1200 раз при формировании матрицы взаимовлияния граничных элементов. Были получены графики зависимости погрешности расчетов от количества граничных элементов. Построены зависимости времени последовательного и распараллеленного расчетов, а также коэффициентов ускорения вычислений от количества граничных элементов и количества расчетных узлов под поверхностью полуплоскости. Показано, что при распараллеливании вычислений коэффициент ускорения расчета растет быстрее при увеличении числа граничных элементов, чем при увеличении числа расчетных узлов.

Список литературы

1. Бенерджи, П. Метод граничных элементов в прикладных науках / П. Бенерджи, Р. Баттерфилд. — Мир, 1984. — 494 с.
2. Крауч, С. Методы граничных элементов в механике твердого тела / С. Крауч, А. Старфилд. — Мир, 1987. — 328 с.
3. Bramas, B. Optimization and parallelization of the boundary element method for the waveequation in time domain / B. Berenger // Distributed, Parallel and Cluster Computing. Universit'e de Bor-deaux, 2016.
4. Molina-Moya, J. An Iterative Parallel Solver in GPU Applied to Frequency Domain Linear Water Wave Problems by the Boundary Element Method [Electronic resource] / Jorge Molina-Moya, Alejandro Enrique Mart'inez-Castro, Pablo Ortiz // Department of Structural Mechanics and Hydraulic Engineering, University of Granada, Edificio Polit'cnico, Granada, Spain – 26.11.2018. — Mode of access: https://www.frontiersin.org/articles/10.3389/fbuil.2018.00069/full. — Date of access: 22.07.2019.
5. Hamada, T. GPU-accelerated boundary element method for Helmholtz equation in three dimensions / T. Hamada, T. Takahashi // Int. J. Numer. Meth. Eng. — 2009. — Vol. 80, No. 10. — C. 1295–1321.
6. An adaptive dual-information FMBEM for 3D elasticity and its GPU implementation / Y. Wang [et al.] // Eng. Anal. Boundary Elem. — 2013. — Vol. 37, No. 10. — Pp. 236–249.
7. D'Azevedo, E. On the effective implementation of a boundary element code on graphics processing units using an out-of-core LU algorithm / E. D'Azevedo, S. Nintcheu Fata // Eng. Anal. Boundary Elem. — 2012. — Vol. 36, No. 8. — Pp. 1246–1255.
8. Kelly, J.M. Numerical solution of the two-phase incompressible Navier-Stokes equations using a GPU-accelerated meshless method / J.M. Kelly, E.A. Divo, A.J. Kassab // Eng. Anal. Boundary Elem. — 2014. — Vol., 40. — Pp. 36–49.
9. Ma, Z. GPU computing of compressible flow problems by a meshless method with space-filling curves / Z. Ma, H. Wang, S. Pu // J. Comput. Phys. — 2014. Vol. 263. — Pp. 113–135.
10. Vatera K. Simple and efficient GPU parallelization of existing H-Matrix accelerated BEM code [Electronic resource] / K. Vatera, T. Betckeb, B. Dilba. — 2017. — Mode of access: https://arxiv.org/pdf/1711.01897.pdf. — Date of access: 22.07.2019.
11. Torkya, A.A. GPU acceleration of the boundary element method for shear-deformable bending of plates / A.A. Torkya, Y.F. Rashed // Eng. Anal. Boundary Elem. — 2017. — Vol.74. — Pp. 34–48.
12. Sanders, J. CUDA by Example: An Introduction to General-Purpose GPU Programming / J. Sanders, E. Kandrot. — Addison-Wesley Professional, 2010. — 312 p.

SHERBAKOV Sergei S., D. Sc. in Phys. and Math., Prof.
Deputy Chairman[1]
Professor of the Department of Theoretical and Applied Mechanics[2]
E-mail: sherbakovss@mail.ru

POLESTCHUK Mikhail M.
Ph. D. Student[2]
E-mail: mikhailpaliashchuk@yandex.ru

[1]State Committee on Science and Technology of the Republic of Belarus, Minsk, Republic of Belarus
[2]Belarusian State University, Minsk, Republic of Belarus

Received 10 September 2019.

ACCELERATION OF BOUNDARY-ELEMENT COMPUTING USING GRAPHICS ACCELERATOR FOR THE ELEMENTS WITH NONLINEAR FORM FUNCTIONS

The implementation of a boundary element method (BEM) using three nonlinear form functions designed for determination of half-plane potential distribution was considered in the current paper. The application of NVidia CUDA technology for parallel computing leading to the essential acceleration of computations was also performed. The accuracy of calculations performed with constant and nonlinear form functions was analyzed. The influence of the surface discretization on accuracy was studied. Parallelization technique for BEM using graphic processor was presented. Timings and acceleration coefficient dependencies on the number of boundary elements and calculation points were presented for sequential and parallel calculations.

Keywords: *boundary element method, acceleration of computing, potential distribution, modelling, CUDA, graphics accelerator, parallelization*

References

1. Banerjee P.K., Butterfield R. *Boundary element methods in engineering science*. London, McGraw-Hill, 1981. 452 p.
2. Crouch S.L., Starfield A.M. *Boundary element methods in solid mechanics*. London, George Allen & Unwin, 1983. 322 p.
3. Bramas B. *Optimization and parallelization of the boundary element method for the wave equation in time domain*. 2016. Available at: https://tel.archives-ouvertes.fr/tel-01306571/document (accessed 27 August 2019).
4. Molina-Moya J., Martinez-Castro A.E., Ortiz P. *An Iterative Parallel Solver in GPU Applied to Frequency Domain Linear Water Wave Problems by the Boundary Element Method*. 2018. Available at: https://www.frontiersin.org/articles/10.3389/fbuil.2018.00069/full (accessed 22 July 2019).
5. Hamada T., Takahashi T. GPU-accelerated boundary element method for Helmholtz equation in three dimensions. *International Journal for Numerical Methods in Engineering*, 2009, vol. 80, no. 10, pp. 1295–1321.
6. Wang Y., Wang Q., Wang G., Huang Y., Wang S. An adaptive dual-information FMBEM for 3D elasticity and its GPU implementation. *Engineering Analysis with Boundary Elements*, 2013, vol. 37, no. 10, pp. 236–249.
7. D'Azevedo E., Nintcheu Fata S. On the effective implementation of a boundary element code on graphics processing units using an out-of-core LU algorithm. *Engineering Analysis with Boundary Elements*, 2012, vol. 36, no. 8, pp. 1246–1255.
8. Kelly J.M., Divo E.A., Kassab A.J. Numerical solution of the two-phase incompressible Navier-Stokes equations using a GPU-accelerated meshless method. *Engineering Analysis with Boundary Elements*, 2014, vol., 40, pp. 36–49.
9. Ma Z., Wang H., Pu S. GPU computing of compressible flow problems by a meshless method with space-filling curves. *Journal of Computational Physics*, 2014, vol. 263, pp. 113–135.
10. Vatera K., Betckeb T., Dilba B. *Simple and efficient GPU parallelization of existing H-Matrix accelerated BEM code*. 2017. Available at: https://arxiv.org/pdf/1711.01897.pdf (accessed 22 July 2019).
11. Torkya A.A., Rashed Y.F. GPU acceleration of the boundary element method for shear-deformable bending of plates. *Engineering Analysis with Boundary Elements*, 2017, vol. 74, pp. 34–48.
12. Sanders J., Kandrot E. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010. 312 p.