ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра механіки, автоматизації та інформаційних технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Інформаційна технологія CUDA паралельних обчислень в задачі стаціонарної теплопровідності для плоскої прямокутної платівки»
(тема кваліфікаційної роботи українською мовою)

«CUDA information technology of parallel calculations in the problem of stationary thermal conductivity for a flat rectangular plate»
(тема кваліфікаційної роботи англійською мовою)

Виконав(ла): здобувач(ка) денної/заочної форми навчання
спеціальності 126 – Інформаційні системи та технології
(код, назва спеціальності)
Освітня програма Інформаційні системи та технології
(назва)
Нуждіна Марина Ігорівна
(прізвище, ім'я, по-батькові здобувача)

Керівник старший викладач Царенко О.П. _____
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)
Рецензент канд. фіз.-мат. наук, доцент Рачинська А.Л.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
_____
№ ___ від ____.____. 20___ р.

Завідувач(ка) кафедри
_____ Алла РАЧИНСЬКА
(підпис)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.20___ р.

Оцінка_____/____/_____
(за національною шкалою/шкалою ECTS/ бали)
Голова ЕК
_____ Володимир ВИЧУЖАНІН
(підпис)

Одеса 2023

АНОТАЦІЯ

У кваліфікаційній роботі розробляється тема «Інформаційна технологія CUDA паралельних обчислень в задачі стаціонарної теплопровідності для плоскої прямокутної пластинки».

Мета роботи – застосування технології паралельних обчислень CUDA задля наближеного розв’язування задачі стаціонарної теплопровідності для плоскої прямокутної пластинки.

В результаті виконання кваліфікаційної роботи було розроблено 5 проектів, призначених для розв’язування задачі стаціонарної теплопровідності для плоскої прямокутної пластинки.

Розробка проводилася у IDE “Visual Studio” із використанням інформаційної технології CUDA паралельних обчислень та мов програмування високого рівня C++ та C#.

Можливість застосування обраного підходу паралельних обчислень була перевірена на іншій модельній задачі, яка пов’язана із наближеним обчисленням невласних інтегралів першого роду, що містять функції Бесселя.

За допомогою програмної оболонки Surfer було побудовано у вигляді кольорової мапи ізотерм шукане поле температур, яке є кінцевим розв’язком задачі стаціонарної теплопровідності для плоскої прямокутної пластинки.

Розрахунки проводилися на мобільному комп’ютері ASUSTeK ROG Strix G713QR_G713QR, оснащеному ЦП AMD Ryzen 9 5900HX та ГП NVIDIA Geforce RTX 3070 Laptop GPU.

Кваліфікаційну роботу виконано на 54 сторінках. Робота містить 26 рисунків та 4 додатки.

ANNOTATION

The subject "CUDA information technology of parallel calculations in the problem of stationary thermal conductivity for a flat rectangular plate" is developed in the qualification work.

The purpose of the work is to use the CUDA parallel computing technology for the approximate solution of the problem of stationary thermal conductivity for a flat rectangular plate.

As a result of completing the qualification work, 5 projects were developed, designed to solve the problem of stationary thermal conductivity for a flat rectangular plate.

The development was carried out in the IDE "Visual Studio" using the CUDA information technology of parallel computing and high-level programming languages C++ and C#.

The possibility of applying the selected approach of parallel calculations was tested on another model problem, which is related to the approximate calculation of improper integrals of the first kind containing Bessel functions.

With the help of the Surfer software, the desired temperature field was constructed in the form of a color map of isotherms, which is the final solution to the problem of stationary thermal conductivity for a flat rectangular plate.

Calculations were performed on an ASUSTeK ROG Strix G713QR_G713QR mobile computer equipped with an AMD Ryzen 9 5900HX CPU and an NVIDIA Geforce RTX 3070 Laptop GPU.

The qualification work is completed on 54 pages. The work contains 26 figures and 4 appendices.

ЗМІСТ

Стор.

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

Скорочення

ГП — графічний процесор

ЦП — центральний процесор

CUDA — Compute Unified Device Architecture

Терміни

хост (host) — центральний процесор, який керує виконанням програми

пристрій (device) – відеоадаптер, який виступає в ролі співпроцесора центрального процесора

грид (grid) — об'єднання блоків, які виконуються на одному пристрої

блок (block) — об'єднання потоків, яке виконується повністю на одному потоковому мультипроцесорі

потік (thread) — одиниця виконання програми

ядро (kernel) — паралельна частина алгоритму, що виконується на гриді [1]

ВСТУП

За останні три десятиліття можна спостерігати стрімкий розвиток комп’ютерних технологій, який значно вплинув на розвиток науки, техніки та на численні аспекти нашої діяльності. Він докорінно змінив спосіб доступу до інформації та технологію виконання завдань.

Дуже значним є підвищення здатності комп’ютерів розв’язувати складні задачі, вирішення яких раніше потребувало багато часу та значних ресурсів. На відміну від перших комп’ютерів (або навіть комп’ютерів, випущених у вісімдесятих або дев’яностих роках минулого століття), сучасні комп’ютери здатні проводити триваліші та складніші обчислення, та обробляти більші обсяги різноманітних даних із безпрецедентною швидкістю.

Велику роль у цьому зіграла і поява паралельного програмування.

Паралельні обчислення — це одночасне виконання кількох дій для вирішення певної задачі. Паралельні обчислення передбачають поділ одної великої задачі на невеликі її частини, які можливо виконувати одночасно на кількох процесорах. Для виконання цих частин можна використовувати ядра ЦП, ГП або розподілені системи.

ГП, окрім відтворення комп’ютерної графіки, здатні виконувати широкий спектр інтенсивних обчислювальних задач.

ГП складаються з великої кількості ядер, які виконують одночасні математичні обчислення. ЦП в основному зосереджені на послідовному виконанні програм, тому графічні процесори обробляють складні задачі значно швидше.

Обчислювальна техніка розвивається від «центральної обробки» на ЦП до «спільної обробки» на ЦП і ГП. Корпорація NVIDIA винайшла архітектуру паралельних обчислень CUDA, яка зараз постачається в її графічних процесорах GeForce, ION, Quadro та Tesla [2].

Технологія CUDA надає розробникам інтерфейс програмування та інструменти, які дозволяють писати та оптимізувати паралельні програми.

Технологія CUDA дозволяє значно підвищити продуктивність обчислень за рахунок використання потужності ГП для прискорення найскладніших завдань, які виконуються на ПК.

Технологія CUDA набула широкого поширення у сферах, які потребують високопродуктивних обчислень, таких як наука або машинне навчання. Хоча для використання технології CUDA потребується наявність в ГП NVIDIA, що робить її недоступною для розробників, які не мають доступу до ПК, які містять необхідні ГП, CUDA є гідною альтернативою використанню кластерних систем, які за рахунок необхідності обміну даними між комп'ютерами у кластері можуть демонструвати менш ефективні результати виконання програм, або використанню суперкомп'ютерів, які в основному доступні державним установам, великим науково-дослідним установам та організаціям зі значними фінансовими ресурсами, та є недоступними для більшості розробників, студентів.

У нашому випадку була поставлена така задача: за допомогою методу Якобі знайти розв'язок задачі стаціонарної теплопровідності для плоскої прямокутної пластинки. Практична реалізація цього метода передбачає тривалі обчислення і має можливість розпаралелювання, що підходить для здійснення обчислень із використанням технології CUDA.

Метою роботи є, по-перше, засвоєння основ використання технології паралельних обчислень CUDA, та, по-друге, безпосереднє розв'язування задачі стаціонарної теплопровідності для плоскої прямокутної пластинки.

Всі програмні компоненти повинні бути об'єднаними в рамках загального робочого простору системи розробки MS Visual Studio, а мови програмування повинні бути C++ та C#.

Кожен чисельний алгоритм повинен бути налагоджений та підтверджений шляхом проведення верифікаційних обчислень на зрозумілих

аналітичних задачах та їх розв’язках. Такі алгоритми повинні реалізовуватися в вигляді бібліотечних компонентів.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Стаціонарне температурне поле для прямокутної пластинки

Розглянемо наступну крайову задачу для рівняння Лапласа для прямокутника із заданими сторонами, які позначені відповідними виносками:

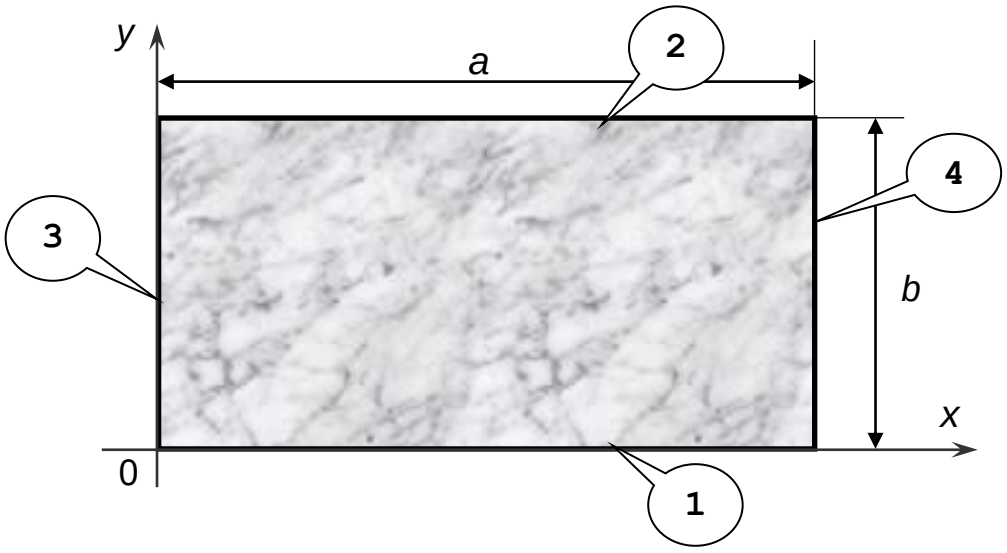

Рисунок 1.1 – Задача стаціонарної теплопровідності для прямокутника

Отже, нехай шукана температура $T(x,y)$ є різницею між температурою пластини $\tilde{T}(x,y)$ та температурою зовнішнього середовища T_o, тобто: $T(x,y) = \tilde{T}(x,y) - T_o$, де температура зовнішнього середовища задана певним чином, наприклад, $T_o = 20^o C$.

Математична постановка відповідної крайової задачі для рівняння Лапласа всередині прямокутної пластинки виглядає наступним чином:

$$\begin{cases} \Delta T = \dfrac{\partial^2 T}{\partial x^2} + \dfrac{\partial^2 T}{\partial y^2} = 0, \qquad 0 \le x \le a,\ \ 0 \le y \le b, \\ T(x,0) = t_1(x) = T_1 \cdot \sin\left(\dfrac{k_1 \pi}{a} x\right), \quad T(x,b) = t_2(x) = T_2 \cdot \sin\left(\dfrac{k_2 \pi}{a} x\right), \\ T(0,y) = t_3(y) = T_3 \cdot \sin\left(\dfrac{k_3 \pi}{b} y\right), \quad T(a,y) = t_4(y) = T_4 \cdot \sin\left(\dfrac{k_4 \pi}{b} y\right), \end{cases} \quad (1.1)$$

де a і b – розміри пластини в дециметрах (без сантиметрів);

T_1, T_2, T_3, T_4 – деякі задані значення температури T,

k_1, k_2, k_3, k_4 – параметри, які мають деякі задані цілочислові значення.

На сторонах прямокутника задані такі температурні функції, які задовольняють наступним умовам в вершинах прямокутника:

$$t_1(0) = t_3(0),\ t_1(a) = t_4(0),\ t_4(b) = t_2(a),\ \ t_2(0) = t_3(b). \qquad (1.2)$$

Аналітичний розв'язок задачі будується в наступному виді [1]:

$$T(x,y) = T_*(x,y) + \\ + \sum_{n=1}^{\infty} \left\{ \left[\bar{t}_{1,n} \cdot \frac{\operatorname{sh}\left(\frac{\pi n}{a}(b-y)\right)}{\operatorname{sh}\left(\frac{\pi n}{a} b\right)} + \bar{t}_{2,n} \cdot \frac{\operatorname{sh}\left(\frac{\pi n}{a} y\right)}{\operatorname{sh}\left(\frac{\pi n}{a} b\right)} \right] \cdot \sin\left(\frac{\pi n}{a} x\right) \right\} + , \qquad (1.3)$$

$$+ \sum_{n=1}^{\infty} \left\{ \left[\bar{t}_{3,n} \cdot \frac{\operatorname{sh}\left(\frac{\pi n}{b}(a-x)\right)}{\operatorname{sh}\left(\frac{\pi n}{b} a\right)} + \bar{t}_{4,n} \cdot \frac{\operatorname{sh}\left(\frac{\pi n}{b} x\right)}{\operatorname{sh}\left(\frac{\pi n}{b} a\right)} \right] \cdot \sin\left(\frac{\pi n}{b} y\right) \right\}$$

де $\bar{t}_{1,n},\ \bar{t}_{2,n},\ \bar{t}_{3,n},\ \bar{t}_{4,n}$ – коефіцієнти Фур'є-функцій, які дорівнюють:

$$\bar{t}_{1,n} = \frac{2}{a} \cdot \int_0^a \sin\left(\frac{\pi n}{a} x\right) \cdot \bar{t}_1(x) \cdot dx\ , \quad \bar{t}_{2,n} = \frac{2}{a} \cdot \int_0^a \sin\left(\frac{\pi n}{a} x\right) \cdot \bar{t}_2(x) \cdot dx\ ,$$

$$\bar{t}_{3,n} = \frac{2}{b} \cdot \int_0^b \sin\left(\frac{\pi n}{b} y\right) \cdot \bar{t}_3(y) \cdot dy\ , \quad \bar{t}_{4,n} = \frac{2}{b} \cdot \int_0^b \sin\left(\frac{\pi n}{b} y\right) \cdot \bar{t}_4(y) \cdot dy\ ,$$

(1.4)

де

$$\bar{t}_1(x) = t_1(x) - T_*(x,0) = T_1 \sin\left(\frac{k_1 \pi}{a} x\right) - T_*(x,0)\ ,$$

$$\bar{t}_2(x) = t_2(x) - T_*(x,b) = T_2 \sin\left(\frac{k_2 \pi}{a} x\right) - T_*(x,b)\ , \qquad (1.5)$$

$$\bar{t}_3(y) = t_3(y) - T_*(0,y) = T_3 \sin\left(\frac{k_3 \pi}{b} y\right) - T_*(0,y)\ ,$$

$$\bar{t}_4(y) = t_4(y) - T_*(a,y) = T_4 \sin\left(\frac{k_4 \pi}{b} y\right) - T_*(a,y)\ .$$

Функція $T_*(x,y) = A + B\cdot x + C\cdot y + D\cdot xy$ є загальним розв'язком даної крайової задачі, де

$$\begin{aligned} A &= t_1(0) = 0, \\ B &= \frac{1}{a}\cdot\left(t_1(a) - t_1(0)\right) = \frac{1}{a}\cdot T_1\cdot\sin(k_1\pi), \\ C &= \frac{1}{b}\cdot\left(t_3(b) - t_3(0)\right) = \frac{1}{b}\cdot T_3\cdot\sin(k_3\pi), \\ D &= \frac{\left(t_2(a) - t_2(0)\right) - \left(t_1(a) - t_1(0)\right)}{ab} = \frac{T_2\cdot\sin(k_2\pi) - T_1\cdot\sin(k_1\pi)}{ab}. \end{aligned} \tag{1.6}$$

Таким чином, функція $T_*(x,y)$ матиме наступний вигляд:

$$T_*(x,y) = \frac{T_1\sin(k_1\pi)}{a}x + \frac{T_3\sin(k_3\pi)}{b}y + \frac{T_2\sin(k_2\pi) - T_1\sin(k_1\pi)}{ab}xy. \tag{1.7}$$

Враховуючи те, що $\sin(k\pi) = 0$, якщо k – ціле число, то матимемо $T_*(x,y) = 0$.

Тоді
$$\begin{aligned} \bar{t}_1(x) &= T_1\cdot\sin\left(\frac{k_1\pi}{a}x\right), \quad \bar{t}_2(x) = T_2\cdot\sin\left(\frac{k_2\pi}{a}x\right), \\ \bar{t}_3(x) &= T_3\cdot\sin\left(\frac{k_3\pi}{b}y\right), \quad \bar{t}_4(x) = T_4\cdot\sin\left(\frac{k_4\pi}{b}y\right). \end{aligned} \tag{1.8}$$

Тепер ми можемо обчислити інтеграли (1.4) та визначити точні значення коефіцієнтів Фур'є.

$$\begin{aligned} \bar{t}_{1,n} &= \frac{2}{a}T_1\cdot\int_0^a \sin\left(\frac{\pi n}{a}x\right)\sin\left(\frac{k_1\pi}{a}x\right)dx = \\ &= \frac{T_1}{a}\cdot\int_0^a \cos\left(\frac{\pi(n-k_1)}{a}x\right)dx - \frac{T_1}{a}\cdot\int_0^a \cos\left(\frac{\pi(n+k_1)}{a}x\right)dx = \\ &= \frac{T_1}{\pi}\left[\frac{\sin\frac{x\pi}{a}(n-k_1)}{(n-k_1)} - \frac{\sin\frac{x\pi}{a}(n+k_1)}{(n+k_1)}\right]_0^a = \\ &= \frac{T_1}{\pi}\left(\frac{\sin\pi(n-k_1)}{n-k_1} - \frac{\sin\pi(n+k_1)}{n+k_1}\right) = 0. \end{aligned} \tag{1.9}$$

Останній вираз дорівнює нулю, оскільки $\sin(m\pi) = 0$, якщо m – будь-яке ціле число.

Але, коли має місце $n = k_1$ інтеграл (9) обчислюється інакше, маємо:

$$\begin{aligned} \bar{t}_{1,n} &= \frac{2}{a} T_1 \cdot \int_0^a \sin^2\left(\frac{n\pi}{a} x\right) dx = \frac{T_1}{a} \cdot \int_0^a \left[1 - \cos\left(\frac{2n\pi}{a} x\right)\right] dx = \\ &= \frac{T_1}{a} \cdot x\Big|_0^a - \frac{T_1}{2n\pi} \cdot \sin\left(\frac{2n\pi}{a} x\right)\Big|_0^a = T_1 . \end{aligned} \tag{1.10}$$

Таким чином, отримуємо перший коефіцієнт:

$$\bar{t}_{1,n} = \begin{cases} 0, & коли\ n \neq k_1 , \\ T_1, & коли\ n = k_1 . \end{cases} \tag{1.11}$$

Обчислимо другий коефіцієнт:

$$\bar{t}_{2,n} = \begin{cases} \frac{2}{a} T_2 \cdot \int_0^a \sin\left(\frac{\pi n}{a} x\right) \sin\left(\frac{k_2\pi}{a} x\right) dx = 0 , & коли\ \ n \neq k_2 , \\ \frac{2}{a} T_2 \cdot \int_0^a \sin^2\left(\frac{n\pi}{a} x\right) dx = T_2 , & коли\ \ n = k_2 . \end{cases} \tag{1.12}$$

Обчислимо третій коефіцієнт:

$$\bar{t}_{3,n} = \begin{cases} \frac{2}{b} T_3 \cdot \int_0^b \sin\left(\frac{\pi n}{a} x\right) \sin\left(\frac{k_3\pi}{a} x\right) dx = 0 , & коли\ \ n \neq k_3 , \\ \frac{2}{b} T_3 \cdot \int_0^b \sin^2\left(\frac{n\pi}{a} x\right) dx = T_3 , & коли\ \ n = k_3 . \end{cases} \tag{1.13}$$

Аналогічним способом обчислюємо останній коефіцієнт:

$$\bar{t}_{4,n} = \begin{cases} \frac{2}{b} T_4 \cdot \int_0^b \sin\left(\frac{\pi n}{a} x\right) \sin\left(\frac{k_4\pi}{a} x\right) dx = 0 , & коли\ \ n \neq k_4 , \\ \frac{2}{b} T_4 \cdot \int_0^b \sin^2\left(\frac{n\pi}{a} x\right) dx = T_4 , & коли\ \ n = k_4 . \end{cases} \tag{1.14}$$

Отже, враховуючи зображення для коефіцієнтів Фур'є (1.11)–(1.14), замість нескінченної суми функцій (1.3) ми будемо мати наступну кінцеву формулу для аналітичного розв'язку для даної модельної крайової задачі:

$$T(x,y) = T_1 \cdot \frac{\operatorname{sh}\left(\frac{\pi k_1}{a}(b-y)\right)}{\operatorname{sh}\left(\frac{\pi k_1}{a}b\right)} \cdot \sin\left(\frac{\pi k_1}{a}x\right) + T_2 \cdot \frac{\operatorname{sh}\left(\frac{\pi k_2}{a}y\right)}{\operatorname{sh}\left(\frac{\pi k_2}{a}b\right)} \cdot \sin\left(\frac{\pi k_2}{a}x\right) + \tag{1.15}$$
$$+T_3 \cdot \frac{\operatorname{sh}\left(\frac{\pi k_3}{b}(a-x)\right)}{\operatorname{sh}\left(\frac{\pi k_3}{b}a\right)} \cdot \sin\left(\frac{\pi k_3}{b}y\right) + T_4 \cdot \frac{\operatorname{sh}\left(\frac{\pi k_4}{b}x\right)}{\operatorname{sh}\left(\frac{\pi k_4}{b}a\right)} \cdot \sin\left(\frac{\pi k_4}{b}y\right),$$

або

$$T(x,y) = T_1 \cdot \frac{\operatorname{sh}\left(\alpha_1(b-y)\right)}{\operatorname{sh}\left(\alpha_1 b\right)} \cdot \sin\left(\alpha_1 x\right) + T_2 \cdot \frac{\operatorname{sh}\left(\alpha_2 y\right)}{\operatorname{sh}\left(\alpha_2 b\right)} \cdot \sin\left(\alpha_2 x\right) + \tag{1.16}$$
$$+T_3 \cdot \frac{\operatorname{sh}\left(\beta_3(a-x)\right)}{\operatorname{sh}\left(\beta_3 a\right)} \cdot \sin\left(\beta_3 y\right) + T_4 \cdot \frac{\operatorname{sh}\left(\beta_4 x\right)}{\operatorname{sh}\left(\beta_4 a\right)} \cdot \sin\left(\beta_4 y\right),$$

де $$\alpha_1 = \frac{\pi k_1}{a},\ \alpha_2 = \frac{\pi k_2}{a} \quad \text{та} \quad \beta_3 = \frac{\pi k_3}{b},\ \beta_4 = \frac{\pi k_4}{b}. \tag{1.17}$$

Для проведення якісних комп'ютерних обчислень нам треба заздалегідь певним чином перетворити члени з (1.16)–(1.17), які містять вирази (дроби) із гіперболічними синусами.

Скористуємося для цього відомим зображенням для цієї функції:

$$\operatorname{sh}(z) = \frac{\exp(z) - \exp(-z)}{2} = \frac{\exp(z)}{2} \cdot \left(1 - \exp(-2z)\right). \tag{1.18}$$

Тоді

$$\frac{\mathrm{sh}\left(\alpha_1(b-y)\right)}{\mathrm{sh}\left(\alpha_1 b\right)} = \frac{\exp(\alpha_1(b-y))}{\exp(\alpha_1 b)} \cdot \frac{1-\exp(-2\alpha_1(b-y))}{1-\exp(-2\alpha_1 b)} = \\ = \exp(-\alpha_1 y) \cdot \frac{1-\exp(-2\alpha_1(b-y))}{1-\exp(-2\alpha_1 b)} = \theta_1(y)\,, \tag{1.19}$$

$$\frac{\mathrm{sh}\left(\alpha_2 y\right)}{\mathrm{sh}\left(\alpha_2 b\right)} = \frac{\exp(\alpha_2 y)}{\exp(\alpha_2 b)} \cdot \frac{1-\exp(-2\alpha_2 y)}{1-\exp(-2\alpha_2 b)} = \\ = \exp(-\alpha_2(b-y)) \cdot \frac{1-\exp(-2\alpha_2 y)}{1-\exp(-2\alpha_2 b)} = \theta_2(y)\,, \tag{1.20}$$

$$\frac{\mathrm{sh}\left(\beta_3(a-x)\right)}{\mathrm{sh}\left(\beta_3 a\right)} = \frac{\exp(\beta_3(a-x))}{\exp(\beta_3 a)} \cdot \frac{1-\exp(-2\beta_3(a-x))}{1-\exp(-2\beta_3 a)} = \\ = \exp(-\beta_3 x) \cdot \frac{1-\exp(-2\beta_3(a-x))}{1-\exp(-2\beta_3 a)} = \theta_3(x)\,, \tag{1.21}$$

$$\frac{\mathrm{sh}\left(\beta_4 x\right)}{\mathrm{sh}\left(\beta_4 a\right)} = \frac{\exp(\beta_4 x)}{\exp(\beta_4 a)} \cdot \frac{1-\exp(-2\beta_4 x)}{1-\exp(-2\beta_4 a)} = \\ = \exp(-\beta_4(a-x)) \cdot \frac{1-\exp(-2\beta_4 x)}{1-\exp(-2\beta_4 a)} = \theta_4(x)\,. \tag{1.22}$$

Тепер формулу (1.16) із урахуванням зображень (1.19)–(1.22) можна записати в компактному виді, цілком підготовленому для реалізації комп’ютерних обчислень:

$$T(x,y) = T_1 \cdot \theta_1(y) \cdot \sin\left(\alpha_1 x\right) + T_2 \cdot \theta_2(y) \cdot \sin\left(\alpha_2 x\right) + \\ + T_3 \cdot \theta_3(x) \cdot \sin\left(\beta_3 y\right) + T_4 \cdot \theta_4(x) \cdot \sin\left(\beta_4 y\right). \tag{1.23}$$

Далі за текстом, на Рисунку 1.2, буде наведене зображення стаціонарного поля температур, яке отримане на основі аналітичного розв’язку (1.23) для даної модельної задачі.

Задля обчислення стаціонарного поля температур було створено відповідний проект консольного додатку `Map_Rect_Plate_01` та відповідний клас `Rect_Plate` в бібліотеці класів `NExp_01`. Результати

обчислень за конкретним варіантом даних оброблялися за допомогою програмної оболонки Surfer.

Температура в кожній точці прямокутної пластинки відображається певним кольором із заданого градієнта кольорів. Лінії чорного кольору позначають ізотерми для даного температурного поля.

Так помаранчеві області поля відповідають додатним температурам, а блакитні області – від'ємним температурам в точках прямокутної пластинки.

Ізолінії білого (або яскраво жовтого) кольору відповідають «нульовим» ізотермам, тобто це ті точки нашої прямокутної пластинки в яких температура $T(x,y) = \tilde{T}(x,y) - T_o$ дорівнює нулю.

Дане поле температур прямокутної пластинки було обчислене для наступних значень параметрів задачі:

$$T_1 = 30.0, \quad T_2 = 20.0, \quad T_3 = -15.0, \quad T_4 = 15.0;$$
$$k_1 = 5, \; k_2 = 3, \; k_3 = 1, \; k_4 = 2; \quad a = 2.0, \quad b = 1.0\,. \tag{1.24}$$

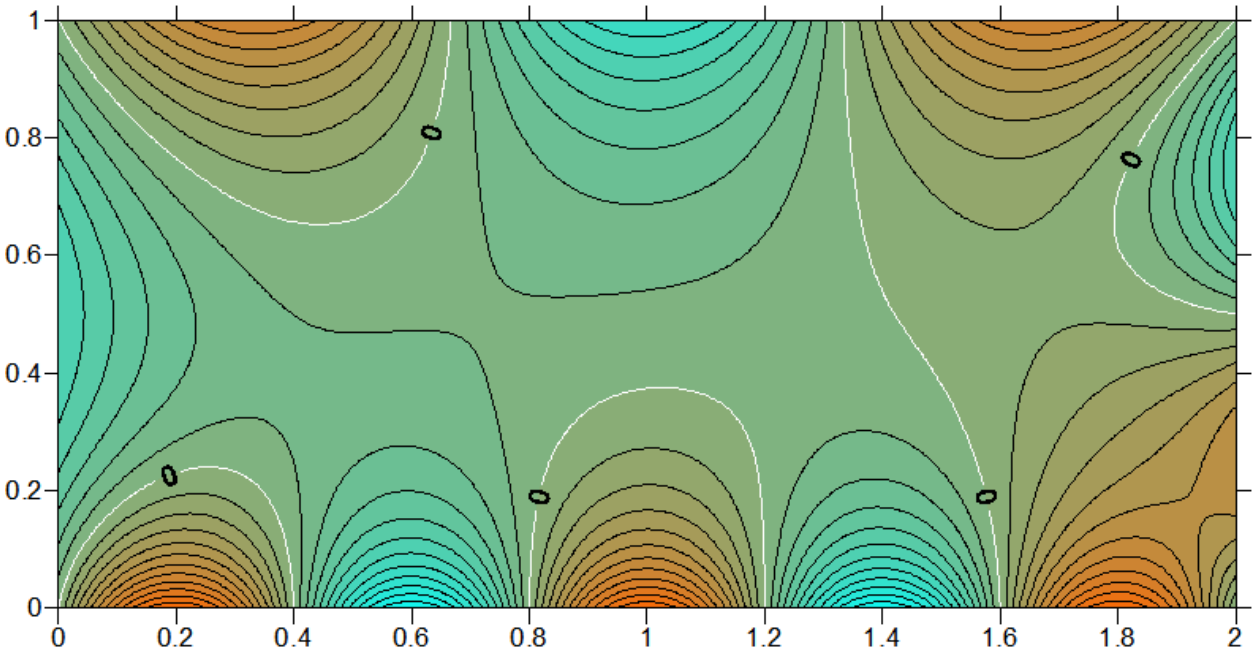

Рисунок 1.2 – Температурне поле для вхідних даних (1.24) та аналітичного розв'язку (1.16) даної крайової задачі

Програмний код даного проекту наведений у Додатку В.

1.2 Чисельне розв’язування задачі

Дану крайову задачу було обрано, оскільки її наближене розв’язування потребує використання алгоритмів, які проводять довготривалі за реальним часом обчислення, що є підставою до застосування паралельного програмування. Якщо ж програма виконується швидко, то вона взагалі не потребує розпаралелювання, бо у такому випадку поділ програми на окремі паралельні частини може займати більше часу, ніж сумарний час обчислень.

Диференціальні рівняння у часткових похідних (в нашому випадку – оператор Лапласа) також мають широке застосування в задачах з моделювання таких фізичних проблем як напружений стан різних пружних тіл, дифузія між речовинами, прогноз погоди, обтікання крила потоком повітря, турбулентність при русі в’язкої рідини тощо.

У цій роботі ми маємо справу із двовимірним рівнянням Лапласа.

Його наближений розв’язок шукатимемо із використанням метода скінченних різниць за схемою Якобі.

Тому прямокутна область пластинки «покривається» різницевою сіткою за наступною схемою:

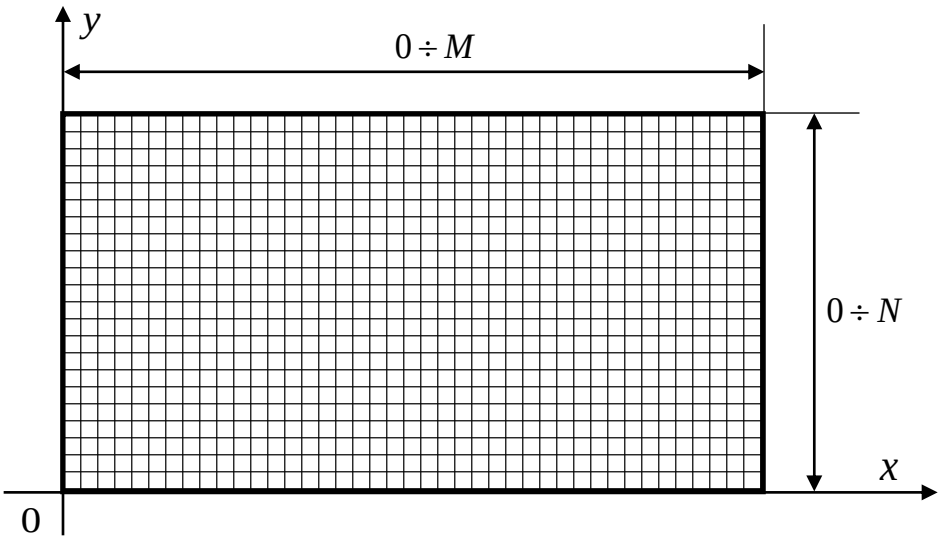

Рисунок 1.3 – Площа пластинки, яка покрита різницевою сіткою

Загальна кількість вузлів сітки складає $(M+1)\cdot(N+1)$, де цілі параметри M та N відповідно дорівнюють $M = a \cdot m$ та $N = b \cdot m$, де m – кількість інтервалів сітки на один дециметр довжини ребра пластинки.

Всі вузли різницевої сітки поділяються на внутрішні, та граничні.

Граничні вузли розташовані безпосередньо на ребрах прямокутної пластинки. Всі інші вузли пластинки – внутрішні. Нас буде цікавити температура саме у внутрішніх вузлах різницевої сітки.

Температура у граничних вузлах задається безпосередньо через граничні умови (1.1) і буде такою на кожній ітерації за схемою Якобі.

На схемі, яку наведено нижче, умовно позначені вузли сітки. Хрестиком – граничні вузли, темні жирні крапки – внутрішні вузли.

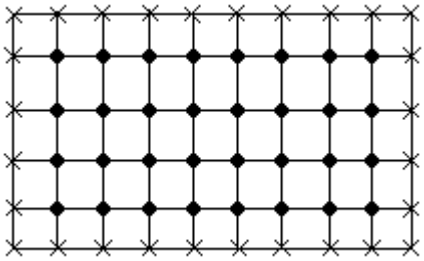

Рисунок 1.4 – Різницева сітка для прямокутної області обчислень

Нас, перш за все, цікавитимуть значення функції $T(x, y)$ у внутрішніх вузлах різницевої сітки.

Початкове значення температур по всіх внутрішніх вузлах сітки задається таким, що відповідає температурі «зовнішнього» середовища, тобто задаємо для цих вузлів $T(x, y) = 0$.

Якщо ідентифікувати двома індексними значеннями i та j координати вузла (x_i, y_j) різницевої сітки, де $i = \overline{0, M}$ та $j = \overline{0, N}$, то температура в цьому вузлі позначатиметься наступним чином: $T(x_i, y_j) = T_{ij}$.

Значення температури на даному, k – му кроці ітераційних обчислень, для вузла (x_i, y_j) різницевої сітки будемо позначати як T_{ij}^k.

На наступному кроці ітерацій температура пластинки для цього ж вузла різницевої сітки буде позначатися як T_{ij}^{k+1}.

Отже початковий стан температурного поля прямокутної пластинки, з якого починаються ітераційні обчислення за схемою Якобі, можна позначити наступним чином: T_{ij}^{0}.

Кожне поле температур, яке відповідає певній ітерації, можна трактувати як деякий шар багатошарової структури даних, де початковий шар відповідає розподілу температур T_{ij}^{0}, $i = \overline{0, M}$ та $j = \overline{0, N}$.

Наступні ітерації супроводжуються додаванням відповідних шарів (полів температур) до цієї структури даних.

Алгоритм обчислення нового значення температури T_{ij}^{k+1} для внутрішнього вузла (x_i, y_j) різницевої сітки здійснюється за наступним правилом (схема Якобі):

$$T_{ij}^{k+1} = \frac{1}{4}\left(T_{i-1j}^{k} + T_{ij-1}^{k} + T_{i+1j}^{k} + T_{ij+1}^{k}\right). \qquad (1.25)$$

Цю схему можна проілюструвати наступним чином:

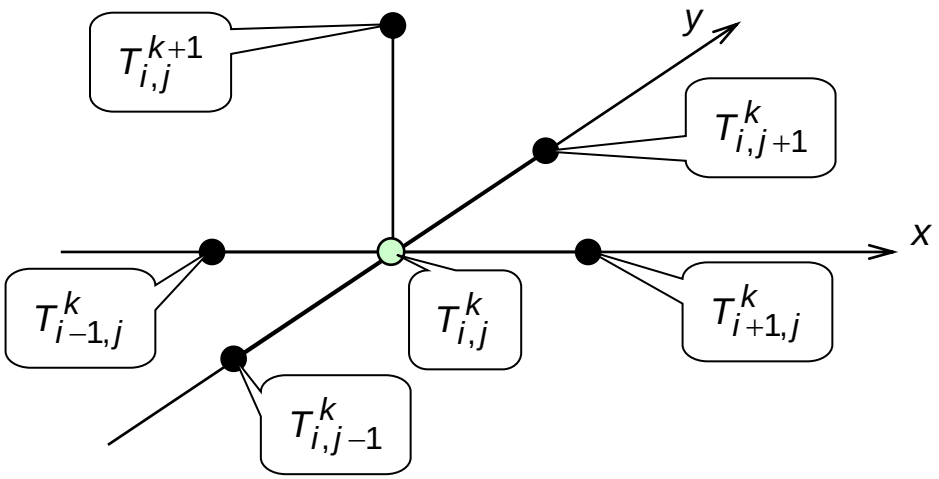

Рисунок 1.5 – Схема Якобі

Процес ітерацій за схемою (1.25) триває до тих пір, поки не буде виконаною умова припинення обчислень. У відповідності до цієї умови різниця між значеннями температур у заданих вузлах на поточній та попередній ітераціях повинна бути не більшою за задану точність обчислень.

Для програмної реалізації цієї умови серед внутрішніх вузлів різницевої сітки обираються декілька, для яких після кожної ітерації перевіряється умова:

$$\left| T_{ij}^{k+1} - T_{ij}^{k} \right| < \varepsilon, \qquad (1.26)$$

де ε – задана абсолютна похибка ітераційних обчислень наближеного розв’язка задачі (1.1).

Кількість контрольних внутрішніх вузлів, в яких перевіряється умова (1.26) припинення ітераційних обчислень, залежить від конкретного чисельного експерименту, та обирається, як правило, емпіричним шляхом.

Задля реалізації вказаного чисельного експерименту було створено консольний проект `CUDA_Task_01`, в якому обчислювалася таблиця результатів, яку потім можна було обробляти за допомогою програмної оболонки Surfer.

Результати обчислень порівнювалися із такими, що були отримані на основі аналітичного розв’язку (1.16) задачі із вхідними даними (1.24).

Відповідна мапа ізотерм має наступний вид:

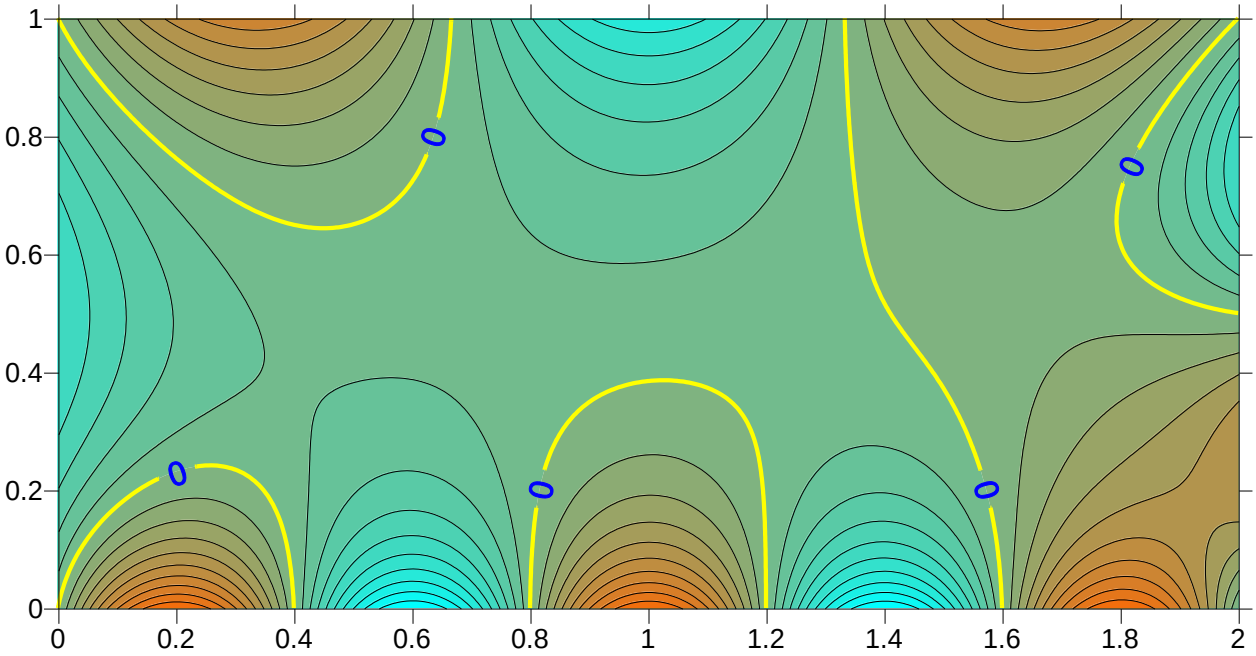

Рисунок 1.6 – Температурне поле для вхідних даних (1.24) та чисельного розв’язку даної крайової задачі, отриманого за схемою Якобі

Програмний код даного проекту наведений у Додатку Б.

На завершення даного розділу можна зазначити наступне.

Для розв’язування рівняння Лапласа існує ще кілька ітераційних методів, окрім метода Якобі: метод Гаусса-Зейделя, багатосітковий метод.

В нашому випадку ми використовуємо саме метод ітерацій Якобі, оскільки для нього нескладно реалізувати паралельні обчислення і він сам по собі є довго тривалим (оскільки здійснює багато ітерацій задля досягнення заданої точності обчислень, тобто має повільну швидкість збігання).

Метод Гаусса-Зайделя оновлює шукані значення ітераційним шляхом, де кожне оновлення одразу залежить від попередніх значень.

Таким чином обмежується паралелізм, оскільки виникає потреба у постійній синхронізації та комунікації потоків між собою, але зазвичай метод Гаусса-Зейделя збігається швидше, ніж ітераційний метод Якобі.

Зробити паралельний алгоритм обчислень для багатосіткового методу є ще більш складною задачею, ніж розпаралелювання методу Гаусса-Зейделя.

У багатосітковому методі вихідна система рівнянь розв’язується на великих сітках, після чого точність розв’язку коригується на дрібнішій сітці.

Оскільки різні сітки взаємопов’язані у цьому методі, розпаралелювання може бути складною задачею, яка потребує постійної синхронізації.

Тому можна вважати найпростішим саме метод Якобі.

В цьому методі (ми це бачимо в формулі (1.25) та на схемі-рисунку 1.5) нове значення температури в кожній внутрішній точці різницевої сітки дорівнює середньому значенню температур з попередніх чотирьох сусідніх точок (ліворуч, праворуч, зверху та знизу).

Цей нескладний процес суто арифметичних обчислень триває, доки не виповняться умови припинення (1.26).

2 ПРАКТИЧНА ЧАСТИНА

2.1 Програмні засоби. Технологія CUDA

CUDA - це паралельна обчислювальна платформа та модель програмування, розроблена NVIDIA. Використання CUDA надає розробникам можливість підвищити швидкість обчислень за рахунок використання потужності ГП. ГП складаються з тисяч ядер, що значно впливає на час виконання програм, які проводять обчислення, тому що з'являється можливість розбити завдання на декілька підзавдань та виконувати ці невеликі підзавдання паралельно. Це робить їх дуже придатними для інтенсивних обчислювальних програм.

Програмувати із використанням технології CUDA можливо за допомогою таких стандартних мов програмування, як C, C++ і Fortran. Щоб виконувати код паралельно, CUDA надає можливість використовувати специфічні для ГП конструкції, такі як функції ядра.

У CUDA існує ієрархія пам'яті, яка включає різні типи пам'яті на ГП, у яку входять глобальна пам'ять (global memory), спільна пам'ять (shared memory) і постійна пам'ять (global memory). Якщо правильно використовувати ці типи пам'яті, можна досягнути високої продуктивності паралельних програм.

CUDA працює тільки із ГП NVIDIA, однак існує можливість користуватися CUDA на різних операційних системах, таких як Windows, Linux і macOS.

CUDA має широке використання у різноманітних сферах. Паралельні обчислення є важливою частиною наукових обчислень, аналізу даних, машинного навчання. CUDA допомагає працівникам у наукових колах та різних галузях промисловості отримувати ефективніші та швидші результати інтенсивних та складних обчислень та є гідною альтернативою кластерним

системам та суперкомп’ютерам, які часто є недоступними для використання та дорожчими.

Системні вимоги, щоб використовувати CUDA:

1) ГП із підтримкою CUDA (у даному випадку - NVIDIA GeForce RTX 3070 Laptop GPU);

2) Підтримувана версія Microsoft Windows (у даному випадку - Windows 10);

3) Підтримувана версія Microsoft Visual Studio (у даному випадку - Visual Studio 2022);

4) Набір інструментів NVIDIA CUDA (у даному випадку - версія 12.0 CUDA Toolkit). [3]

Розглянемо основні особливості програмування з використанням технології CUDA.

Хост призначений для виконання послідовних частин програми, підготовки та копіювання даних на пристрій, запуску ядра та копіювання даних з пристрою. Паралельні частини алгоритму виконуються у потоках, які оформлені у вигляді ядер.

Розглянемо специфікатори для функцій та змінних, які використовуються у CUDA:

1) специфікатор `__global__` застосовується для функцій, які задають ядро, виконується на пристрої та може бути викликаний із хоста. Функції __global__ можуть повертати лише void;

2) специфікатор `__host__` застосовується для завдання функцій, що виконуються на хості та може бути викликаний із хоста;

3) специфікатор `__device__` застосовується для завдання функцій, що виконуються на пристрої та може бути викликаний із пристрою.

Специфікатори `__host__` і `__device__` можуть використовуватися разом.

У CUDA існує кілька спеціальних змінних, які існують усередині кожного обчислювального ядра, які використовуються для того, щоб відрізняти потоки:

1) `dim3 gridDim` — містить інформацію про конфігурацію гриду при запуску ядра;

2) `uint3 blockIdx` — координати блоку всередині гриду;

3) `dim3 blockDim` — розмірність блоку при запуску ядра;

4) `uint3 threadIdx` — координати потоку всередині блоку.

Оголошення функції для ядра з параметром `params`:

```
__global__ void Kernel_name(params);
```

Запуск ядра проводиться наступним чином:

`Kernel_name<<<grid, block, mem, stream>>>( params )`, де

`dim3 grid` — розмір гриду для запуску;

`dim3 block` — розмір блоку при запуску;

`size_t mem` — кількість пам'яті, що розділяється на блок, яка виділяється для даного запуску під динамічне використання всередині ядра;

`cudaStream_t stream` — опис потоку, в якому запускається це ядро. [1]

2.2 Програмні засоби. C++

C++ — це мова програмування високого рівня, яка була розроблена як розширення мови програмування C у 1985 році.

До особливостей C++ можна віднести такі характеристики:

1) C++ є об'єктно-орієнтованою мовою програмування: у ній присутні класи, об’єкти, успадкування, поліморфізм та інкапсуляція, що сприяє створенню коду, який є модульним і який можливо використовувати багато разів. Таким чином, керування та підтримка проектів стають простішими;

2) C++ є швидкою та ефективною мовою програмування: C++ дозволяє керувати пам’яттю низького рівня та надає такі функції, як вказівники та посилання, що призводить до більш ефективного виконання коду;

3) C++ є широко використовуваною та підтримуваною мовою програмування: існує велика кількість документації, C++ має широку підтримку від спільноти, є можливість користуватися постійно розширюваними та оновлюваними бібліотеками та фреймворками. [4]

C++ також є мовою, що підтримується технологією CUDA. Однак, C++ не є єдиною мовою програмування, що можна використовувати для програмування з використанням технології CUDA. Наведемо переваги, які виділяють C++ серед мов програмування, які підтримуються CUDA:

1) Оскільки C++ є більш сучасною мовою програмування, ніж C або Fortran, C++ пропонує більше функцій, які покращують організацію коду, його керування та підтримку;

2) Багато розробників обирають C++ в якості мови програмування для застосування технології CUDA. Щоб почати програмувати на C++ із використанням технології CUDA, можна скористатися можливістю ознайомитися з великою кількістю безкоштовних навчальних посібників та ресурсів та розробників, які розміщені в Інтернеті. Товариство розробників на C++ є багаточисленним та активним, що значить, що для розробників-початківців значно підвищуються шанси отримати підтримку стосовно розробки із використанням CUDA. Форуми, документація від розробників NVIDIA, приклади коду на GitHub можуть стати корисними під час розробки на C++ із використанням CUDA.

Хоча існує можливість програмувати із використанням CUDA на таких мовах, як C і Fortran, і у деяких випадках не існує великої різниці між тим або іншим варіантом, вибір мови C++ для програмування із використанням технології CUDA може надати розробникові ширший функціонал, більшу кількість документації та сильнішу підтримку спільноти розробників.

2.3 Програмні засоби. Surfer

Surfer — програма, розроблена компанією Golden Software. Вона в основному використовується для 3D картографування поверхні та візуалізації наукових і геопросторових даних. Surfer дозволяє користувачам візуалізувати дані у різних форматах і створювати детальні карти.

Surfer надає своїм користувачам такі можливості:

1) Візуалізація даних: Surfer пропонує широкий вибір інструментів, щоб якісно і точно відображати дані таким чином, як необхідно користувачеві;

2) Аналіз даних: Surfer пропонує можливість аналізу даних, що є дуже корисною функцією для працівників таких галузей, як сільське господарство, археологія, будівництво, освіта, екологія, геофізика;

3) Surfer підтримує велику кількість форматів даних. Існує можливість імпортувати та експортувати дані у різних форматах, такі як TXT, XLS, GIF, CSV, DBF, GML, KMZ, KML, LAS та багато інших, без необхідності конвертації. [5]

У нашому випадку Surfer використовується для побудування поля температур, яке є вирішенням задачі стаціонарної теплопровідності для плоскої прямокутної пластинки. Розраховані дані записуються до текстового файлу і потім імпортуються до програми Surfer, після чого обробляються програмою, і в результаті формується контурна карта, яка відповідає полю температур.

2.4 Програмна реалізація

Для реалізації послідовної та паралельної версії програми створимо два робочих простори, один з яких назвемо `Task_01`, а інший – `Cuda_Task_01`. Перший робочий простір міститиме проекти, призначені для реалізації послідовної програми, другий – для паралельної. Відповідно,

проекти у першому робочому просторі буде реалізовано мовою програмування C#, а проекти у другому – мовою C++ із використанням інформаційної технології CUDA паралельних обчислень.

Робочий простір `Task_01` складається з трьох проектів:

1) проект `NExp_01` є бібліотекою класів, призначених для обчислення аналітичного розв'язку задачі;

2) проект `Map_Rect_Plate_01` містить консольний додаток, в якому обчислюється аналітичний розв'язок задачі та формується таблиця результатів, яку потім можна було обробляти за допомогою програмної оболонки Surfer;

3) проект `Rect_Plate_01_Jacoby` містить консольний додаток, в якому обчислюється чисельний розв'язок задачі та формується таблиця результатів, яку потім можна було обробляти за допомогою програмної оболонки Surfer.

Робочий простір `Cuda_Task_01` складається з аналогічних проектів, за винятком проекту `Map_Rect_Plate_01`.

Тепер розглянемо окремо кожний з проектів. Почнемо з бібліотеки класів `NExp_01`, яка у паралельному варіанті програми складається з заголовкового файлу `rect_plate.cuh` та вихідного файлу `Rect_Plate.cu`, а у послідовному варіанті – з одного файлу `Rect_Plate.cs`.

Заголовковий файл `rect_plate.cuh` (див. Додаток А.1) містить декларації змінних та прототипів функцій класу `Rect_Plate`. Змінні `T1, T2, T3, T4` відповідають деяким заданим значенням температури `T`, змінні `k1, k2, k3, k4` відповідають параметрам, які мають деякі задані цілочислові значення (див. формулу (1.24)), а змінні `al_1, al_2, bt_3, bt_4` відповідають значенням у формулі (1.17). Усі перелічені змінні оголошуються за допомогою модифікатора доступу `private`, що значить, що до них не можна отримати доступ поза класом.

Заголовковий файл rect_plate.cuh містить прототип конструктора Rect_Plate(double _a, double _b, double _T1, double _T2, double _T3, double _T4, int _k1, int _k2, int _k3, int _k4), який має специфікатори __host__ і __device__, що значить, що об'єкт класу Rect_Plate може бути викликаний як на хості, так і на пристрою. Також у файлі присутній прототип функції Txy(double x, double y), який має аналогічні специфікатори. Перелічені функції мають модифікатор доступу public, що значить, що до них можна отримати доступ поза класом.

Вихідний файл Rect_Plate.cu (див. Додаток А.2) містить реалізацію функцій, прототипи яких містить заголовковий файл rect_plate.cuh. У конструкторі Rect_Plate(double _a, double _b, double _T1, double _T2, double _T3, double _T4, int _k1, int _k2, int _k3, int _k4) змінним T1, T2, T3, T4 присвоюється відповідне значення, передане за допомогою параметрів _T1, _T2, _T3 та _T4, а змінним al_1, al_2, bt_3, bt_4 присвоюється значення згідно з формулою (1.17). Функція Txy(double x, double y) призначена для визначення температури у точці з координатами (x, y). Із урахуванням формул (1.19)–(1.22) розраховується значення температури згідно з формулою (1.23), яке і повертається з даної функції.

Тепер розглянемо проект Map_Rect_Plate_01, який складається з консольного додатку Main_01.cs (див. Додаток В). Даний файл містить клас Main_01, який складається з функції Main(string[] args). У цій функції обчислюється аналітичний розв'язок задачі та формується таблиця результатів, яку потім можна було обробляти за допомогою програмної оболонки Surfer. У циклі for() обчислюється значення температури у кожній точці пластинки із використанням функції Txy(double x, double y), після чого обчислене значення записується до файлу

Map_01.txt за допомогою класу StreamWriter, який надає можливість для синхронного запису тексту до файлу.

Далі розглянемо проект Rect_Plate_01_Jacoby, який складається з консольного додатку Rect_Plate_01_Jacoby.cu у випадку паралельної програми або Rect_Plate_01_Jacoby.cs у випадку послідовної програми. Даний файл містить функцію main(int argc, char** argv), у якій проводяться послідовні обчислення, підготовка та копіювання даних на пристрій та з пристрою, запуск ядер, запис результатів до текстового файлу та очищення пам'яті. Також у файлі є функція ядра Jacobi(int stream,int num_streams, int num_blocks, int num_threads, double T1,double T2, double T3, double T4,s**T,int ic,int jc, int* iters, double* err1, double* err2, double* err), у якій відбуваються паралельні обчислення.

Розглянемо функцію main(int argc, char** argv) детальніше. Згідно з формулою (1.24), змінним T1, T2, T3, T4, a, b, k1, k2, k3 та k4 присвоюються відповідні значення. Далі визначаються кількість вузлів сітки на один дециметр довжини ребра пластинки та координати контрольної точки. Змінні al_1, al_2, bt_3, bt_4 ініціалізуються згідно з формулою (1.17). Далі створюється об'єкт класу Rect_Plate, який у майбутньому буде використаний для порівняння результатів чисельного розв'язку із результатом аналітичного розв'язку у контрольній точці. Далі створюється різницева сітка та відбувається ініціалізація "початкового" поля температур значеннями { 0.0, 0.0 }, після чого згідно з формулою (1.1) визначаються граничні умови на ребрах.

Далі перейдемо до етапу, який присутній у паралельному варіанті програми і відсутній у послідовному варіанті. Щоб запустити ядро і отримати результат на хості після виконання ядра, для зберігання змінних, які передаються до ядра в якості параметрів, необхідно виділити пам'ять на

ГП. Викликаючи `cudaMalloc`, вказуємо розмір блоку пам’яті, який хочемо виділити на ГП. Функція повертає вказівник на виділений блок пам’яті.

Після цього копіюємо змінну з ЦП до ГП із використанням функції `cudaMemcpy`. Потрібно вказати джерело (пам’ять ЦП) і призначення (пам’ять ГП), розмір даних, які потрібно скопіювати, і напрямок копіювання (від хоста до пристрою).

Після цього відбувається запуск ядра: коли змінна знаходиться на ГП, можна запустити функцію ядра, яка працюватиме з даними. У коді ядра тепер можна отримати доступ до цієї змінної.

Коли робота ядер завершена, копіюємо змінну з ГП до ЦП із використанням функції `cudaMemcpy`, але цього разу вказуємо напрямок копіювання від пристрою до хоста.

Таким чином ми працюємо з такими змінними:

1) `T`, яка відповідальна за масив, який містить різницеву сітку;

2) `iters`, яка позначає кількість ітерацій під час паралельних обчислень;

3) `err, err1, err2`, необхідні для визначення похибки.

Перед запуском ядра необхідно визначити розмір ядра, тобто кількість потоків та блоків (див. Підрозділ 2.1). Для того, щоб паралельна програма працювала найефективніше, важливо тестувати запуск ядер з різним розміром, оскільки інколи запуск більшої кількості потоків може виявитися дорожчим, ніж запуск меншої кількості потоків. Також треба враховувати, що запуск кількох ядер одночасно може зробити програму повільнішою, тому що на запуск додаткових ядер йдуть ресурси ГП. Детальніше про оптимальні параметри запуску ядер йдеться у розділі 3.

Щоб виміряти час, витрачений на виконання програми, скористаємося подіями CUDA: оголошуємо дві змінні `cudaEvent_t`, `start` і `stop`, щоб представляти події початку та кінця для відліку часу. Перед запуском ядер записуємо подію початку відліку часу, а після завершення роботи ядер

записуємо подію кінця відліку часу та визначаємо час, який пройшов між двома подіями, за допомогою функції `cudaEventElapsedTime`.

Запустивши ядра, чекаємо на завершення всіх ядер за допомогою функції `cudaDeviceSynchronize()`. Ця функція гарантує, що виконання функції `main()` не продовжиться, поки всі ядра не завершать роботу. Після копіювання даних з пристрою до хоста виводимо результати на консоль, після чого записуємо дані до текстового файлу `Map_Jx1.txt` та звільняємо пам'ять.

Перейдемо до функції ядра. Щоб завершити ітерації, необхідно, щоб різниця між значеннями температур у заданих вузлах на поточній та попередній ітераціях була не більшою за задану точність обчислень. Відповідно, нам необхідно обрати потік, у якому ми будемо обчислювати цю різницю та порівнювати із точністю обчислень. Обираємо потік, який має значення змінних `stream`, `blockIdx.x` та `threadIdx.x` рівними 0, тобто найперший потік, і в ньому ініціалізуємо значення `err`, `err1` та `err2`. Далі в усіх потоках визначаємо кількість елементів масиву T, яку має обробляти кожний потік, кожний блок та кожне ядро, та за допомогою цих значень визначаємо, на якому проміжку масиву має працювати кожний потік. Після цього у циклі `do while` за формулою (1.25) проводимо розрахунки. У першому потоці кожні 300 ітерацій визначаємо різницю між значеннями температур у заданому вузлі на поточній та попередній ітераціях та виводимо результат на консоль. Коли ця різниця менша за `1.0E-4`, потоки завершують роботу, і робота функції `main()` продовжується.

Робота послідовної програми простіша за роботу паралельної програми: для проведення розрахунків проводяться ті ж самі дії, однак без запуску ядер, усі розрахунки проводяться у функції `main()` та, відповідно, без застосування паралельних обчислень.

2.5 Побудування поля температур із використанням програми Surfer

Коли виконання паралельної програми завершується успішно, отримуємо файл даних Map_Jx1.txt, який виглядає наступним чином:

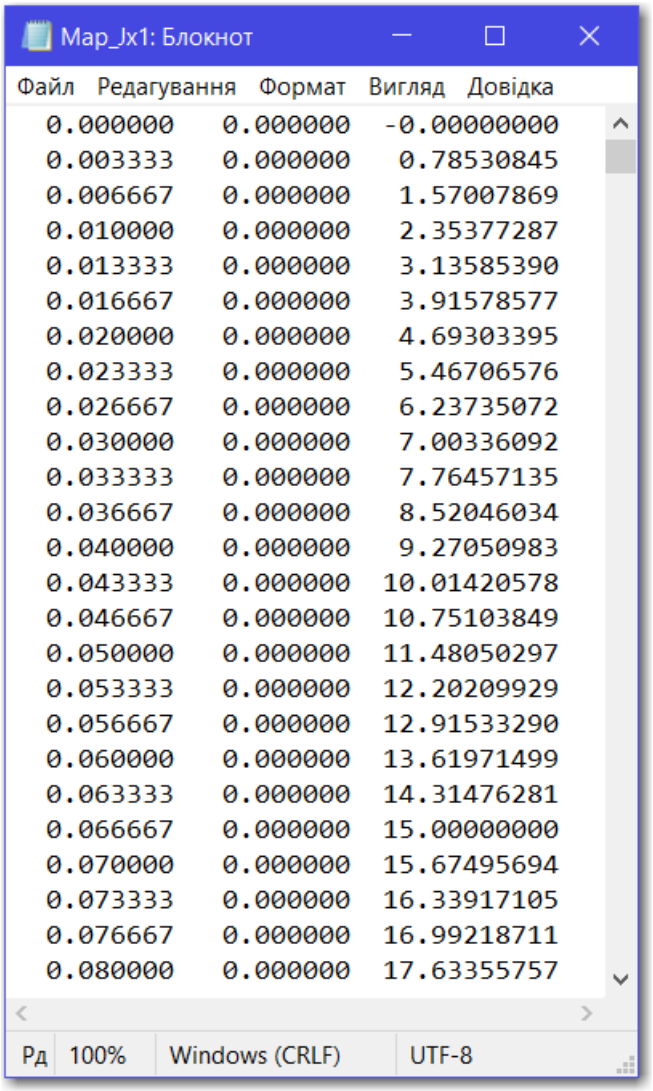

Рисунок 2.1 - Файл даних Map_Jx1.txt для паралельної програми

Також маємо аналогічні файли для чисельного та аналітичного розв’язків, отримані в результаті виконання послідовної програми :

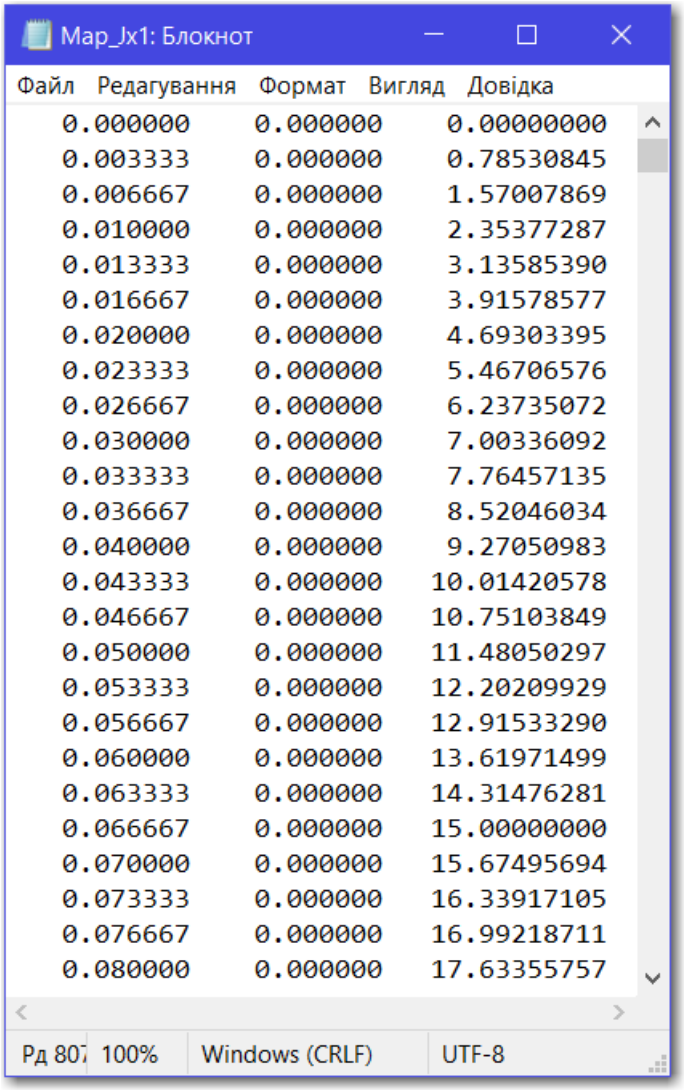

Рисунок 2.2 - Файл даних Map_Jx1.txt для послідовної програми

Коли ми маємо готові файли, які містять необхідні дані, можемо зайнятися побудуванням поля температур із використанням програми Surfer. Запустивши програму Surfer, на формі програми перейдемо до пункту меню Grid і в цьому пункті оберемо підпункт Data, після чого оберемо текстовий файл з даними, які ми маємо обробляти програмою Surfer.

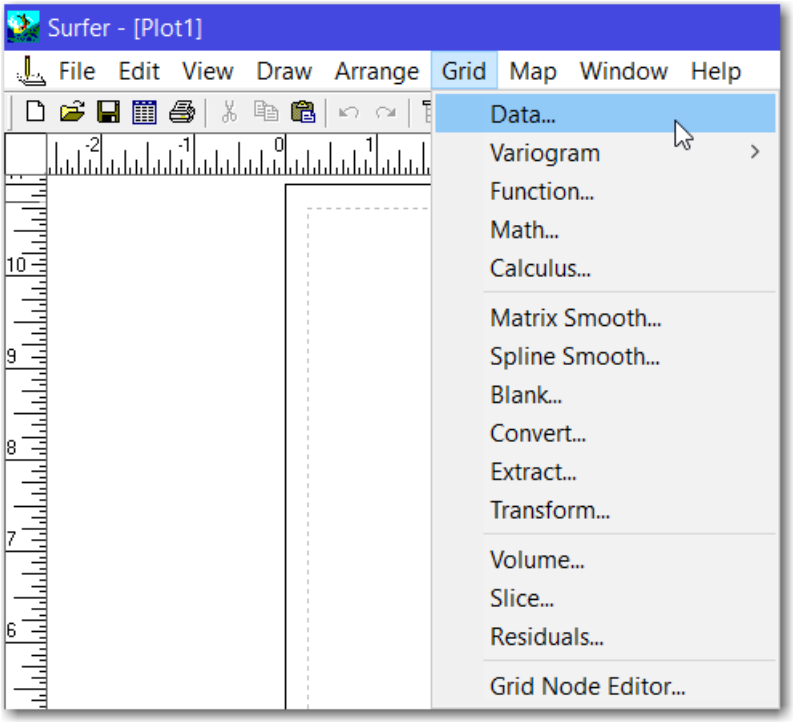

Рисунок 2.3 - Пункт меню Grid

На формі Open обираємо текстовий файл Map_Jx1.txt.

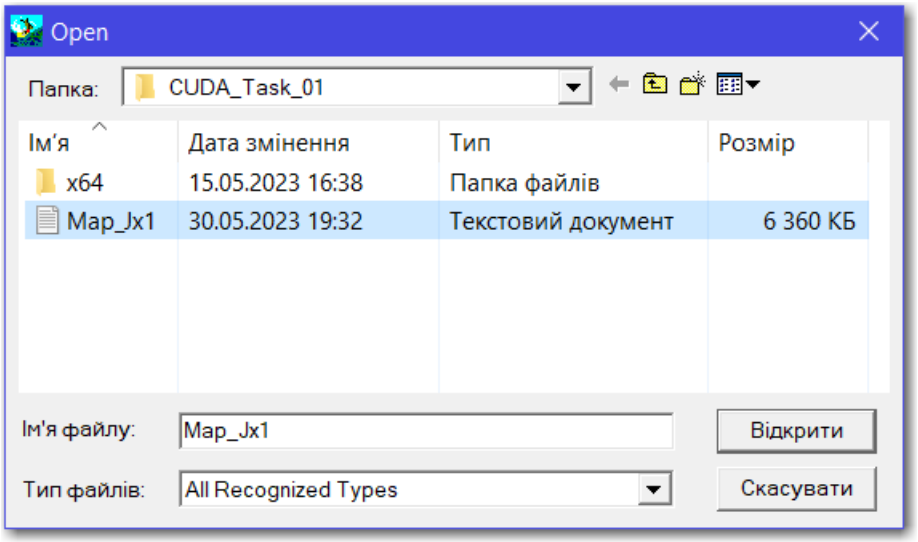

Рисунок 2.4 - Форма Open

Після цього відкривається службова форма, на якій ми маємо задати параметри інтерполяційної сітки, яка буде сформована програмою Surfer на основі даних, розміщених в файлі Map_Jx1.txt.

На формі бачимо границі змінних x і y, які ми використовували при розрахунку цієї таблиці даних, але крок таблиці відрізняється від того кроку,

за яким будувалася таблиця, тому необхідно змінити крок на той, який використовувався під час будування таблиці.

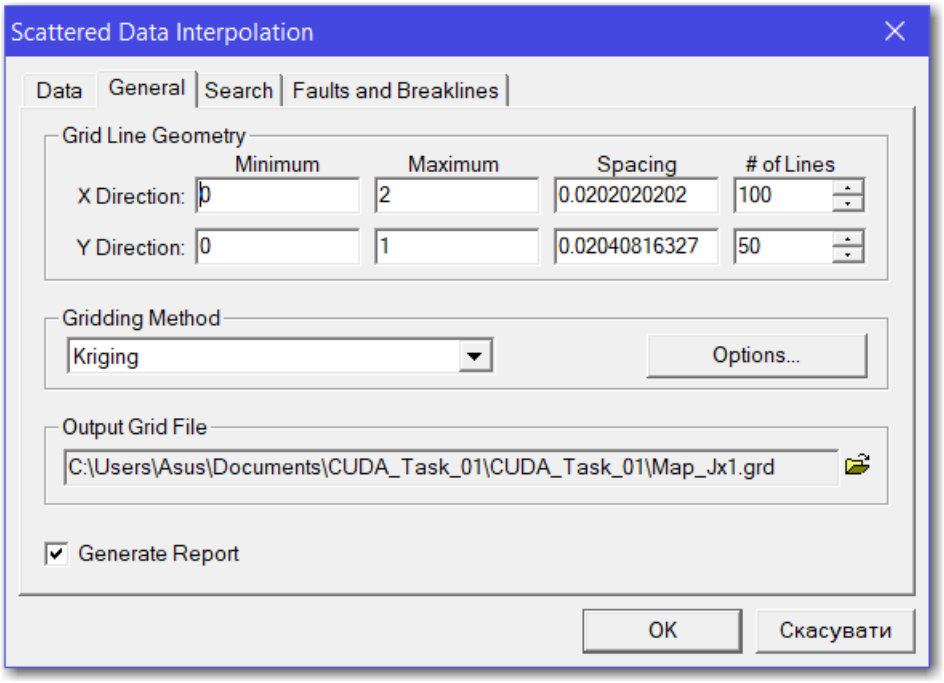

Рисунок 2.5 - Початкові параметри інтерполяційної сітки

Змінимо дані (кількість ліній сітки) та метод створення інтерполяційної сітки:

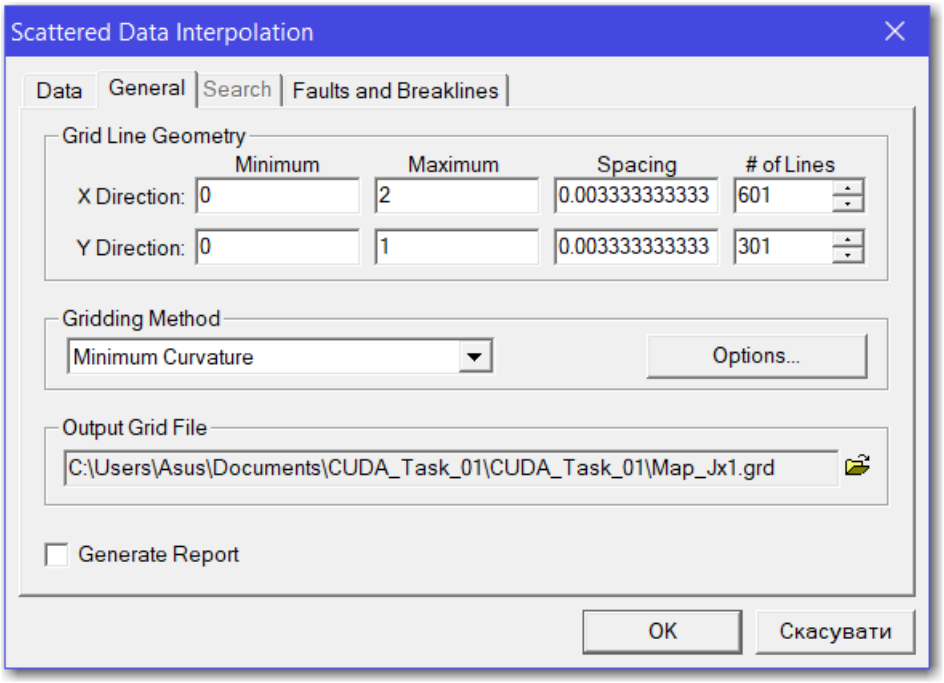

Рисунок 2.6 - Змінені параметри інтерполяційної сітки

Тепер крок просторової сітки став таким же за значенням, як при розрахунку таблиці функції. Перейдемо до етапу створення мапи рівнів (ізоліній). Оберемо пункт меню Map, у ньому Contour Map та New Contour Map:

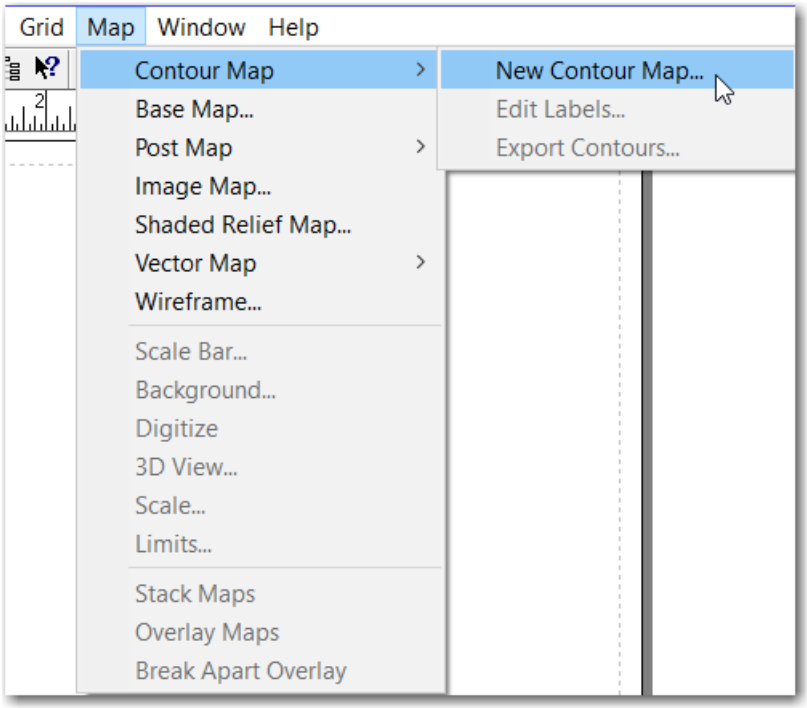

Рисунок 2.7 - Пункт меню Contour Map

Оберемо на формі Open Grid файл Map_Jx1.grd, який був створений програмою Surfer на попередньому етапі:

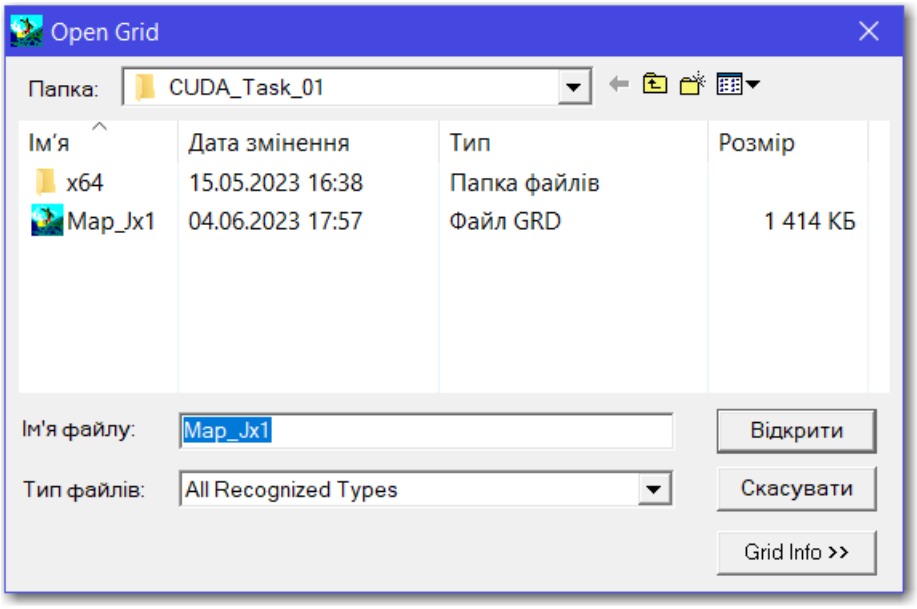

Рисунок 2.8 - Форма Open

На формі Contour Map Properties оберемо пункт Fill Contours:

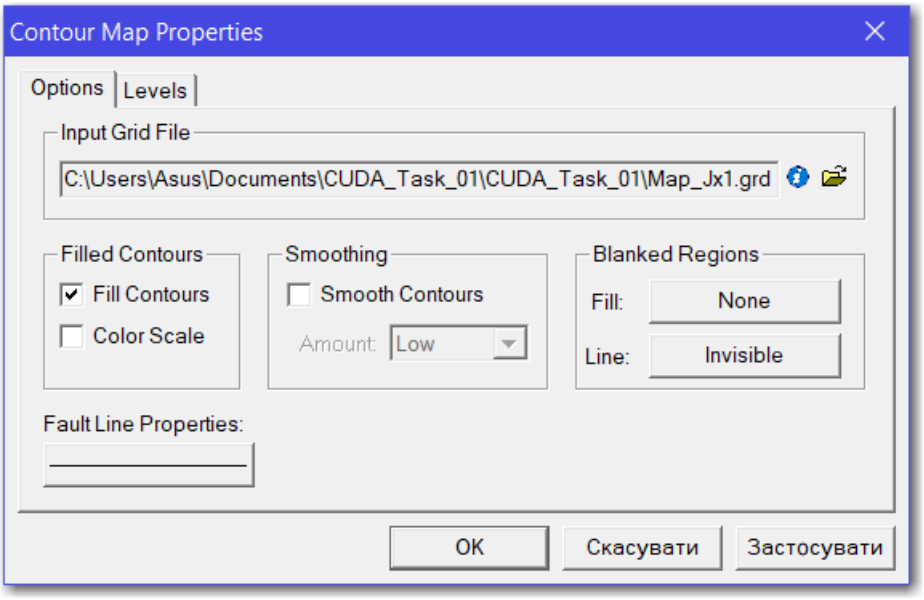

Рисунок 2.9 - Форма Contour Map Properties

та перейдемо до закладки Levels:

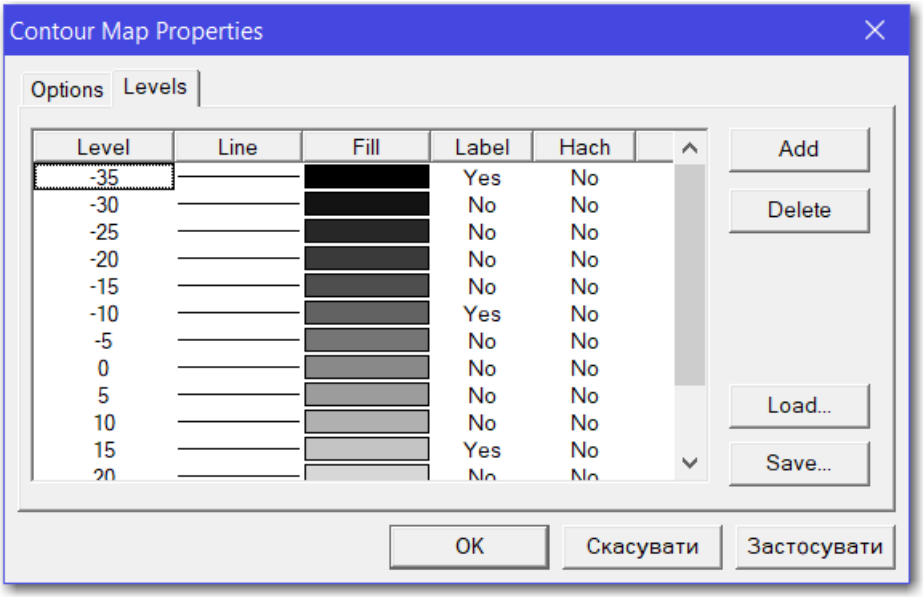

Рисунок 2.10 - Закладка Levels

На цій закладці ми будемо формувати кольорову палітру майбутнього зображення ізоліній нашої функції. Для наочності оберемо кольорову гамму (кольоровий градієнт), в якій буде представлено поле ізоліній.

Для найменших значень функції ми оберемо блакитний колір, а для найбільших значень – помаранчевий колір.

Після того, як ми натиснемо на кнопку Fill, з’явиться діалогове вікно, в якому маємо обрати позицію Foreground Color.

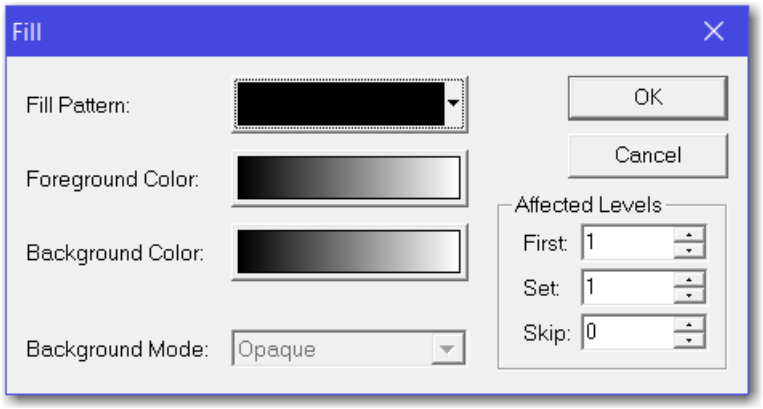

Рисунок 2.11 - Діалогове вікно Fill

На формі Color Spectrum сформуємо обраний нами кольоровий спектр для мінімальних значень (блакитного кольору):

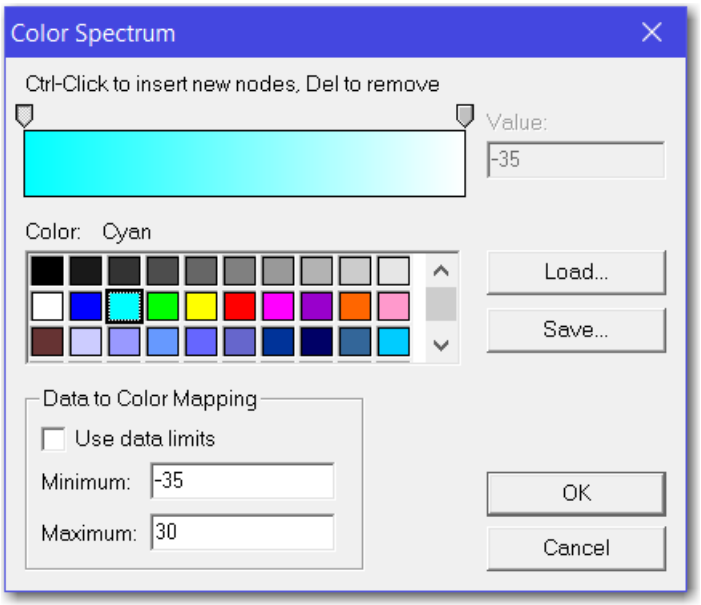

Рисунок 2.12 - Вибір кольорового спектру для мінімальних значень

А потім – для максимальних значень:

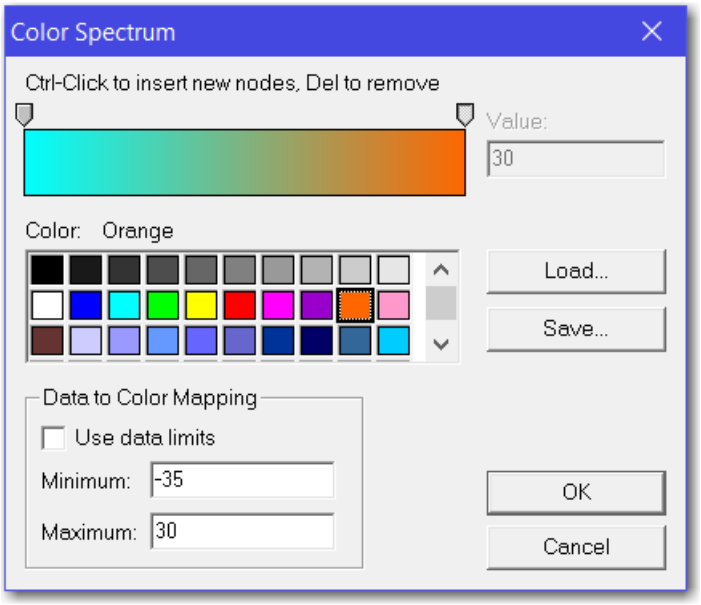

Рисунок 2.13 - Вибір кольорового спектру для максимальних значень

Нас цікавлять ті області, в яких функція, яка досліджується, може досягати нульових значень, тому на формі Line Properties для значення 0 замінимо колір в позиції Color на червоний :

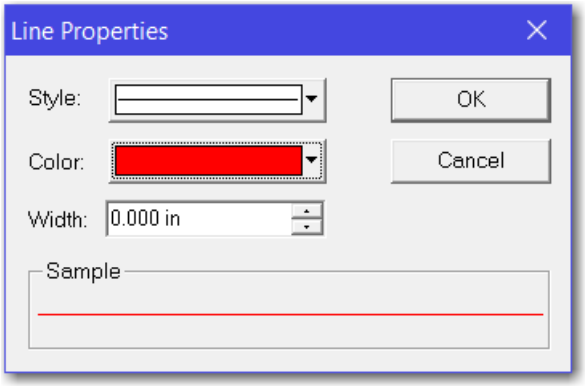

Рисунок 2.14 - Вибір кольору для значення 0

Змінимо інтервал між ізолініями функції. На формі властивостей мапи клацнемо на кнопці Levels та перейдемо на форму Contour Levels, де задамо новий інтервал між ізолініями нашої функції – 2.5.

Таким чином, маємо наступний вид мапи з ізолініями:

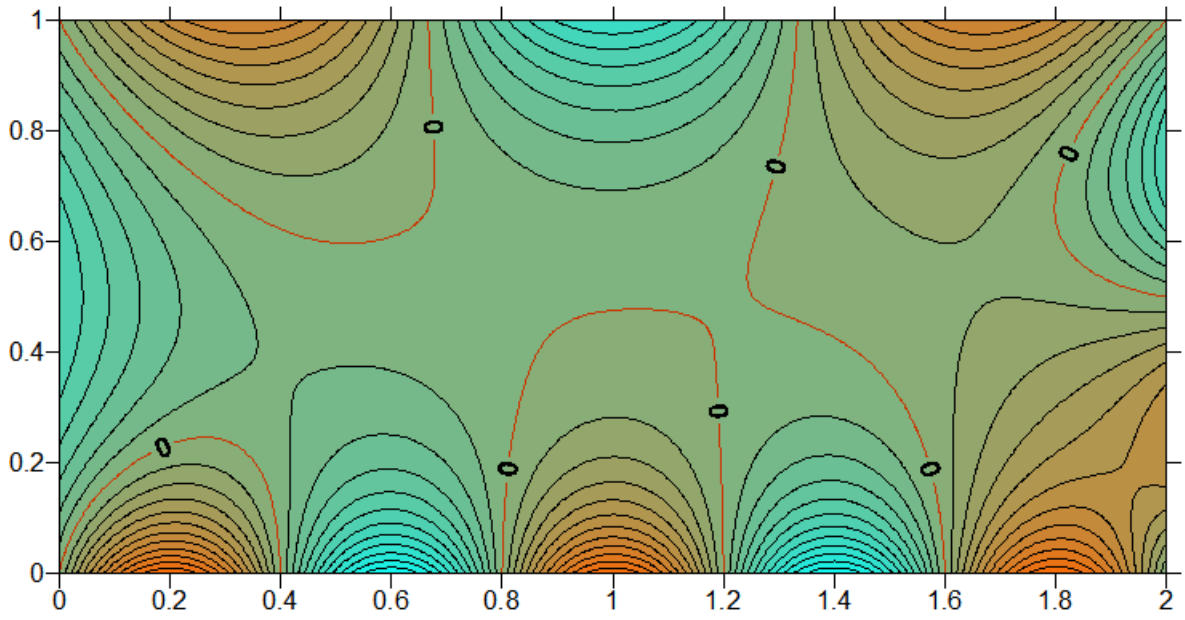

Рисунок 2.15 - Поле температур із використанням чисельного розв’язку

Аналогічним чином будуємо поле температур із використанням послідовної програми, отримані результати для числового та аналітичного розв’язків наведені у підрозділах 1.1 та 1.2.

3 ТЕСТУВАННЯ

Порівняємо виконання послідовної та паралельної програм. Запустивши послідовну програму, маємо такий результат:

```
   20400    T[   570,    30] =  14,44064072    0,00083046
   20700    T[   570,    30] =  14,44142591    0,00078519
   21000    T[   570,    30] =  14,44216822    0,00074231
   21300    T[   570,    30] =  14,44286992    0,00070170
   21600    T[   570,    30] =  14,44353314    0,00066322
   21900    T[   570,    30] =  14,44415989    0,00062674
   22200    T[   570,    30] =  14,44475205    0,00059217
   22500    T[   570,    30] =  14,44531143    0,00055938
   22800    T[   570,    30] =  14,44583972    0,00052829
   23100    T[   570,    30] =  14,44633851    0,00049879
   23400    T[   570,    30] =  14,44680932    0,00047081
   23700    T[   570,    30] =  14,44725359    0,00044427
   24000    T[   570,    30] =  14,44767266    0,00041908
   24300    T[   570,    30] =  14,44806784    0,00039517
   24600    T[   570,    30] =  14,44844032    0,00037248
   24900    T[   570,    30] =  14,44879127    0,00035095
   25200    T[   570,    30] =  14,44912178    0,00033051
   25500    T[   570,    30] =  14,44943290    0,00031111
   25800    T[   570,    30] =  14,44972559    0,00029270
   26100    T[   570,    30] =  14,45000081    0,00027521
   26400    T[   570,    30] =  14,45025943    0,00025862
   26700    T[   570,    30] =  14,45050229    0,00024287
   27000    T[   570,    30] =  14,45073020    0,00022791
   27300    T[   570,    30] =  14,45094392    0,00021372
   27600    T[   570,    30] =  14,45114416    0,00020024
   27900    T[   570,    30] =  14,45133161    0,00018745
   28200    T[   570,    30] =  14,45150692    0,00017531
   28500    T[   570,    30] =  14,45167071    0,00016379
   28800    T[   570,    30] =  14,45182356    0,00015285
   29100    T[   570,    30] =  14,45196604    0,00014248
   29400    T[   570,    30] =  14,45209867    0,00013263
   29700    T[   570,    30] =  14,45222196    0,00012329
   30000    T[   570,    30] =  14,45233638    0,00011443
   30300    T[   570,    30] =  14,45244240    0,00010602
   30600    T[   570,    30] =  14,45254045    0,00009805
iters = 30600  error = 9,8E-005  time = 149 sec
T[570,30] =  14,45254045
  1,900000    0,100000    14,44977858
```

Рисунок 3.1 - Результат виконання послідовної програми

Щоб досягти необхідної точності, нам знадобилося 30600 ітерацій і 149 секунд.

Тепер подивимося на результат роботи паралельної програми. Оскільки програма побудована таким чином, щоб рівномірно розподілити роботу між усіма потоками, обрана кількість потоків має бути дільником числа 300.

Запустимо одне ядро графічного процесора, яке містить 6 блоків, кожний з яких містить 5 потоків:

```
   21900    T[   570,    30] =  14.44105124    0.00075752
   22200    T[   570,    30] =  14.44176933    0.00071809
   22500    T[   570,    30] =  14.44244996    0.00068064
   22800    T[   570,    30] =  14.44309501    0.00064505
   23100    T[   570,    30] =  14.44370624    0.00061123
   23400    T[   570,    30] =  14.44428533    0.00057909
   23700    T[   570,    30] =  14.44483386    0.00054853
   24000    T[   570,    30] =  14.44535333    0.00051947
   24300    T[   570,    30] =  14.44584689    0.00049355
   24600    T[   570,    30] =  14.44631236    0.00046547
   24900    T[   570,    30] =  14.44675284    0.00044048
   25200    T[   570,    30] =  14.44716955    0.00041671
   25500    T[   570,    30] =  14.44756365    0.00039410
   25800    T[   570,    30] =  14.44793624    0.00037259
   26100    T[   570,    30] =  14.44828836    0.00035213
   26400    T[   570,    30] =  14.44862102    0.00033266
   26700    T[   570,    30] =  14.44893625    0.00031523
   27000    T[   570,    30] =  14.44923270    0.00029645
   27300    T[   570,    30] =  14.44951239    0.00027969
   27600    T[   570,    30] =  14.44977612    0.00026374
   27900    T[   570,    30] =  14.45002469    0.00024856
   28200    T[   570,    30] =  14.45025882    0.00023413
   28500    T[   570,    30] =  14.45047998    0.00022116
   28800    T[   570,    30] =  14.45068727    0.00020729
   29100    T[   570,    30] =  14.45088214    0.00019487
   29400    T[   570,    30] =  14.45106519    0.00018305
   29700    T[   570,    30] =  14.45123700    0.00017181
   30000    T[   570,    30] =  14.45139813    0.00016113
   30300    T[   570,    30] =  14.45154910    0.00015097
   30600    T[   570,    30] =  14.45169040    0.00014130
   30900    T[   570,    30] =  14.45182297    0.00013258
   31200    T[   570,    30] =  14.45194633    0.00012336
   31500    T[   570,    30] =  14.45206139    0.00011506
   31800    T[   570,    30] =  14.45216857    0.00010718
   32100    T[   570,    30] =  14.45226825    0.00009968
iters = 32100  error =  1.0E-04  time: 135 s
T[570,30] =  14.45226859
  1.900000    0.100000   14.44977858
```

Рисунок 3.2 - Результат виконання одного ядра GPU

Щоб досягти необхідної точності, нам знадобилося 32100 ітерацій і 135 секунд.

Далі протестуємо виконання двох ядер GPU: цього разу також кожне з ядер матиме 6 блоків по 5 потоків.

```
   24300    T[   570,    30] =  14.44385782    0.00057614
   24600    T[   570,    30] =  14.44440831    0.00055049
   24900    T[   570,    30] =  14.44493231    0.00052400
   25200    T[   570,    30] =  14.44542924    0.00049694
   25500    T[   570,    30] =  14.44590401    0.00047477
   25800    T[   570,    30] =  14.44635584    0.00045183
   26100    T[   570,    30] =  14.44678421    0.00042838
   26400    T[   570,    30] =  14.44719335    0.00040913
   26700    T[   570,    30] =  14.44758264    0.00038929
   27000    T[   570,    30] =  14.44795162    0.00036898
   27300    T[   570,    30] =  14.44830393    0.00035231
   27600    T[   570,    30] =  14.44863899    0.00033507
   27900    T[   570,    30] =  14.44895762    0.00031863
   28200    T[   570,    30] =  14.44925951    0.00030189
   28500    T[   570,    30] =  14.44954760    0.00028809
   28800    T[   570,    30] =  14.44982142    0.00027383
   29100    T[   570,    30] =  14.45008072    0.00025930
   29400    T[   570,    30] =  14.45032802    0.00024730
   29700    T[   570,    30] =  14.45056298    0.00023495
   30000    T[   570,    30] =  14.45078532    0.00022234
   30300    T[   570,    30] =  14.45099726    0.00021195
   30600    T[   570,    30] =  14.45119848    0.00020122
   30900    T[   570,    30] =  14.45138945    0.00019097
   31200    T[   570,    30] =  14.45157001    0.00018055
   31500    T[   570,    30] =  14.45174193    0.00017192
   31800    T[   570,    30] =  14.45190498    0.00016305
   32100    T[   570,    30] =  14.45205955    0.00015457
   32400    T[   570,    30] =  14.45220551    0.00014596
   32700    T[   570,    30] =  14.45234434    0.00013883
   33000    T[   570,    30] =  14.45247581    0.00013147
   33300    T[   570,    30] =  14.45260027    0.00012447
   33600    T[   570,    30] =  14.45271763    0.00011735
   33900    T[   570,    30] =  14.45282906    0.00011143
   34200    T[   570,    30] =  14.45293440    0.00010535
   34500    T[   570,    30] =  14.45303359    0.00009919
iters = 34500  error =  9.9E-05  time: 67 s
T[570,30] =  14.45303396
  1.900000    0.100000   14.44977858
```

Рисунок 3.3 - Результат виконання двох ядер GPU

Щоб досягти необхідної точності, нам знадобилося 34500 ітерацій і 67 секунд.

Далі запустимо чотири ядра GPU: як і минулого разу, кожне з ядер матиме 6 блоків по 5 потоків.

```
   28500    T[   570,    30] =  14.44574824    0.00044315
   28800    T[   570,    30] =  14.44617254    0.00042431
   29100    T[   570,    30] =  14.44658021    0.00040766
   29400    T[   570,    30] =  14.44696884    0.00038864
   29700    T[   570,    30] =  14.44734083    0.00037199
   30000    T[   570,    30] =  14.44769681    0.00035598
   30300    T[   570,    30] =  14.44803764    0.00034083
   30600    T[   570,    30] =  14.44836375    0.00032611
   30900    T[   570,    30] =  14.44867585    0.00031210
   31200    T[   570,    30] =  14.44897453    0.00029868
   31500    T[   570,    30] =  14.44926042    0.00028589
   31800    T[   570,    30] =  14.44953404    0.00027362
   32100    T[   570,    30] =  14.44979580    0.00026176
   32400    T[   570,    30] =  14.45004631    0.00025051
   32700    T[   570,    30] =  14.45028609    0.00023978
   33000    T[   570,    30] =  14.45051539    0.00022930
   33300    T[   570,    30] =  14.45073469    0.00021930
   33600    T[   570,    30] =  14.45094535    0.00021066
   33900    T[   570,    30] =  14.45114601    0.00020066
   34200    T[   570,    30] =  14.45133798    0.00019197
   34500    T[   570,    30] =  14.45152152    0.00018354
   34800    T[   570,    30] =  14.45169700    0.00017549
   35100    T[   570,    30] =  14.45186484    0.00016784
   35400    T[   570,    30] =  14.45202537    0.00016053
   35700    T[   570,    30] =  14.45217883    0.00015346
   36000    T[   570,    30] =  14.45232546    0.00014664
   36300    T[   570,    30] =  14.45246565    0.00014019
   36600    T[   570,    30] =  14.45259964    0.00013399
   36900    T[   570,    30] =  14.45272773    0.00012809
   37200    T[   570,    30] =  14.45285006    0.00012233
   37500    T[   570,    30] =  14.45296692    0.00011686
   37800    T[   570,    30] =  14.45307900    0.00011208
   38100    T[   570,    30] =  14.45318562    0.00010662
   38400    T[   570,    30] =  14.45328747    0.00010185
   38700    T[   570,    30] =  14.45338466    0.00009718
iters = 38700  error =  9.7E-05  time: 33 s
T[570,30] =  14.45338466
  1.900000    0.100000   14.44977858
```

Рисунок 3.4 - Результат виконання чотирьох ядер GPU

Щоб досягти необхідної точності, нам знадобилося 38700 ітерацій і 33 секунди.

Далі запустимо шість ядер GPU: кожне з ядер матиме 10 блоків по 5 потоків.

```
   61800    T[   570,    30] =  14.47841772    0.00025928
   62100    T[   570,    30] =  14.47868933    0.00027161
   62400    T[   570,    30] =  14.47893996    0.00025064
   62700    T[   570,    30] =  14.47918849    0.00024852
   63000    T[   570,    30] =  14.47942727    0.00023879
   63300    T[   570,    30] =  14.47967250    0.00024523
   63600    T[   570,    30] =  14.47990988    0.00023738
   63900    T[   570,    30] =  14.48012812    0.00021824
   64200    T[   570,    30] =  14.48035741    0.00022929
   64500    T[   570,    30] =  14.48057219    0.00021478
   64800    T[   570,    30] =  14.48076873    0.00019654
   65100    T[   570,    30] =  14.48097729    0.00020856
   65400    T[   570,    30] =  14.48117278    0.00019549
   65700    T[   570,    30] =  14.48137225    0.00019947
   66000    T[   570,    30] =  14.48155201    0.00017977
   66300    T[   570,    30] =  14.48174440    0.00019239
   66600    T[   570,    30] =  14.48192222    0.00017782
   66900    T[   570,    30] =  14.48209837    0.00017615
   67200    T[   570,    30] =  14.48227683    0.00017846
   67500    T[   570,    30] =  14.48243715    0.00016031
   67800    T[   570,    30] =  14.48260538    0.00016824
   68100    T[   570,    30] =  14.48276365    0.00015827
   68400    T[   570,    30] =  14.48291874    0.00015509
   68700    T[   570,    30] =  14.48307492    0.00015618
   69000    T[   570,    30] =  14.48321581    0.00014089
   69300    T[   570,    30] =  14.48335860    0.00014279
   69600    T[   570,    30] =  14.48349592    0.00013733
   69900    T[   570,    30] =  14.48364028    0.00014436
   70200    T[   570,    30] =  14.48377598    0.00013571
   70500    T[   570,    30] =  14.48391593    0.00013994
   70800    T[   570,    30] =  14.48404844    0.00013251
   71100    T[   570,    30] =  14.48418288    0.00013445
   71400    T[   570,    30] =  14.48431161    0.00012872
   71700    T[   570,    30] =  14.48444204    0.00013043
   72000    T[   570,    30] =  14.48453724    0.00009520
iters = 72000  error =  9.5E-05  time: 14 s
T[570,30] =  14.48453770
  1.900000    0.100000   14.44977858
```

Рисунок 3.5 - Результат виконання шістьох ядер GPU

Щоб досягти необхідної точності, нам знадобилося 81900 ітерацій і 16 секунд.

Подальше збільшення кількості ядер GPU не є продуктивним для даної задачі, оскільки у даному випадку ємність кожного ядра не є достатньою, щоб виправдати їхній запуск.

ВИСНОВКИ

У цій роботі була розглянута крайова задача теплопровідності та отриманий у кінцевому вигляді її аналітичний розв’язок.

Цей аналітичний розв’язок був використаний в якості тесту для аналізу коректності результатів, що були отримані при чисельному розв’язуванні цієї ж задачі наближеним ітераційним методом Якобі.

Для реалізації чисельного розрахунку використовувалися інформаційна технологія CUDA паралельних обчислень (CUDA Version 12.0) та мови програмування високого рівня C++ та C#.

Сама можливість такого підходу перевірялася на іншій модельній задачі, яка пов’язана із наближеним обчисленням невласних інтегралів першого роду, що містять функції Бесселя.

Шукане поле температур, яке є кінцевим розв’язком задачі стаціонарної теплопровідності для плоскої прямокутної пластинки, було побудовано у вигляді кольорової мапи ізотерм за допомогою програмної оболонки Surfer.

Кожен чисельний алгоритм налагоджений та підтверджений шляхом проведення верифікаційних обчислень на зрозумілих аналітичних задачах та їх розв’язках. Такі алгоритми реалізуються у вигляді бібліотечних компонентів.

Загальна кількість проектів, розроблених під час виконання кваліфікаційної роботи – 5.

Розрахунки проводилися на мобільному комп’ютері ASUSTeK ROG Strix G713QR_G713QR, оснащеному ЦП AMD Ryzen 9 5900HX та ГП NVIDIA Geforce RTX 3070 Laptop GPU.

За результатами даної роботи зроблено доповідь та опубліковано тези на двадцятій всеукраїнській конференції студентів і молодих науковців “Інформатика, інформаційні системи та технології”, яка проводилася факультетом математики, фізики та інформаційних технологій Одеського

національного університету імені І. І. Мечникова та фізико-математичним факультетом державного закладу “Південноукраїнський національний педагогічний університет імені К. Д. Ушинського” 28 квітня 2023 року. [7]

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Казённов А.М. Основы технологии CUDA // Компьютерные исследования и моделирование. – М.: 2010, т. 2, № 3, с. 295-308
2. Technology | GeForce (nvidia.com) [Електронний ресурс]. – Режим доступу: https://www.nvidia.com/en-gb/geforce/technologies/cuda/technology/
3. CUDA Installation Guide for Microsoft Windows (nvidia.com) [Електронний ресурс]. – Режим доступу: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html
4. What is C++? Easy Intro to the C++ Programming Language (hackr.io) [Електронний ресурс]. – Режим доступу: https://hackr.io/blog/what-is-cpp
5. Surfer® | 2D & 3D mapping, modeling & analysis software for scientists and engineers (goldensoftware.com) [Електронний ресурс]. – Режим доступу: https://www.goldensoftware.com/products/surfer
6. Тихонов А.Н., Самарский А.А. Уравнения математической физики. Москва. Издательство «Наука», 1977
7. Нуждіна М.І. Використання технології CUDA для паралельних обчислень в задачах стаціонарної теплопровідності / Нуждіна М.І., Царенко О.П. // Тези доповідей двадцятої всеукраїнської конференції студентів і молодих науковців “Інформатика, інформаційні системи та технології”. Одеса, ОНУ імені І.І.Мечникова, ПНПУ імені К.Д.Ушинського, 28 квітня 2023 р. – Одеса, 2023, С. 114-115

ДОДАТОК А.1

Лістинг програмної реалізації класу проекту NExp_01 (заголовкові файли)

rect_plate.cuh:

```
#pragma once
#include <cuda.h>
#include <curand_kernel.h>
#include "device_launch_parameters.h"

namespace RP
{
    class Rect_Plate
    {
    private:
        double T1, T2, T3, T4, a, b;
        int k1, k2, k3, k4;
        double al_1, al_2, bt_3, bt_4;
    public:
        __host__ __device__ Rect_Plate(double _a, double _b,
            double _T1, double _T2, double _T3, double _T4,
            int _k1, int _k2, int _k3, int _k4);
        __host__ __device__ double Txy(double x, double y);
    };
}
```

ДОДАТОК А.2

Лістинг програмної реалізації проекту NExp_01 (вихідні файли)

Rect_Plate.cu:

```
#define _USE_MATH_DEFINES
#include <math.h>
#include "rect_plate.cuh"

using namespace RP;

Rect_Plate::Rect_Plate(double _a, double _b, double _T1, double
_T2,
    double _T3, double _T4, int _k1, int _k2, int _k3, int _k4)
{
    T1 = _T1; T2 = _T2; T3 = _T3; T4 = _T4; a = _a;
    k1 = _k1; k2 = _k2; k3 = _k3; k4 = _k4; b = _b;
    // формули (18)
    al_1 = M_PI * k1 / a; al_2 = M_PI * k2 / a;
    bt_3 = M_PI * k3 / b; bt_4 = M_PI * k4 / b;
}

double Rect_Plate::Txy(double x, double y)
{
    // формули (20)-(23)
    double t1 = exp(-al_1 * y) * (1.0 - exp(-2 * al_1 * (b -
y))) /
        (1.0 - exp(-2 * al_1 * b));
    double t2 = exp(-al_2 * (b - y)) * (1.0 - exp(-2 * al_2 *
y)) /
        (1.0 - exp(-2 * al_2 * b));
    double t3 = exp(-bt_3 * x) * (1.0 - exp(-2 * bt_3 * (a -
x))) /
        (1.0 - exp(-2 * bt_3 * a));
    double t4 = exp(-bt_4 * (a - x)) * (1.0 - exp(-2 * bt_4 *
x)) /
        (1.0 - exp(-2 * bt_4 * a));
    // формула (24)
    t1 *= T1 * sin(al_1 * x); t2 *= T2 * sin(al_2 * x);
    t3 *= T3 * sin(bt_3 * y); t4 *= T4 * sin(bt_4 * y);
    return t1 + t2 + t3 + t4;
}
```

ДОДАТОК Б

Лістинг програмної реалізації проекту CUDA_Task_01

Rect_Plate_01_Jacoby.cu:

```
#define _USE_MATH_DEFINES
#include <math.h>
#include "../NExp_01/rect_plate.cuh"
#include "../NExp_01/Rect_plate.cu"
#include <stdio.h>
#include <string>
#include <cfloat>
#include <iostream>
#include <locale.h>
#include <cuda.h>
#include <curand_kernel.h>
#include "device_launch_parameters.h"
#include <chrono>

using namespace RP;

struct s
{
    double t0;
    double t1;
};

__global__ void Jacobi(int stream,int num_streams, int num_blocks, int num_threads, double T1,double T2, double T3, double T4,s**T,int ic,int jc, int* iters, double* err1, double* err2, double* err)
{
    double eps = 1.0E-4;
    if (stream == 0 && blockIdx.x == 0 && threadIdx.x==0)
    {
        *err1 = 0.0, * err = DBL_MAX,
            * err2 = (abs(T1) + abs(T2) + abs(T3) + abs(T4)) / 4;
    }
    int elements_per_stream = 600 / num_streams;
    int elements_per_block = elements_per_stream / num_blocks;
    int elements_per_thread= elements_per_block / num_threads;
    int M1 = elements_per_stream * stream + elements_per_block * blockIdx.x + elements_per_thread*threadIdx.x;
    int M2 = elements_per_stream * stream + elements_per_block * blockIdx.x + elements_per_thread * (threadIdx.x + 1);
    if (stream == 0 && blockIdx.x == 0)
        M1 += 1;
    do
    {
```

```
        for (int i = M1; i < M2; i++)
            for (int j = 1; j < 300; j++)
                T[i][j].t1 = (T[i - 1][j].t0 + T[i + 1][j].t0 +
T[i][j - 1].t0 + T[i][j + 1].t0) / 4;
        for (int i = M1; i < M2; i++)
            for (int j = 1; j < 300; j++)
                T[i][j].t0 = (T[i - 1][j].t1 + T[i + 1][j].t1 +
T[i][j - 1].t1 + T[i][j + 1].t1) / 4;
        if (stream==0 && blockIdx.x == 0 && threadIdx.x==0)
        {
            *iters += 1;
            if (*iters % 300==0)
            {
                *err1 = *err2;
                *err2 = T[ic][jc].t0;
                *err = *err2 - *err1;
                printf("   %8d           T[%6d,%6d]   =   %12.8lf
%12.8lf\n", *iters, ic, jc, *err2, *err);
            }
        }
    } while (abs(*err) > eps);
}

int main(int argc, char** argv)
{
    setlocale(LC_NUMERIC, "C");
    double T1 = 30.0, T2 = 20.0, T3 = -15.0, T4 = 15.0;
    double a = 2, b = 1;
    int k1 = 5, k2 = 3, k3 = 1, k4 = 2;
    int m = 300;  //К-ть розбиття на 1 дм довжини пластинки
    double xc = 1.9, yc = 0.1; // Координати контрольної точки
    int ic = (int)(xc * m), jc = (int)(yc * m);
    double al_1 = M_PI * k1 / a; double al_2 = M_PI * k2 / a;
    double bt_3 = M_PI * k3 / b; double bt_4 = M_PI * k4 / b;
    Rect_Plate plate = Rect_Plate(a, b, T1, T2, T3, T4, k1, k2,
k3, k4);
    // Різніцева сітка
    int M = (int)(a * m), N = (int)(b * m);
    s** T = new s * [M + 1];
    for (int i = 0; i < M + 1; i++)
    {
        T[i] = new s[N + 1];
    }
    int i, j;
    // Ініціалізація "початкового" поля температур значеннями
    for (i = 1; i < M; i++)
        for (j = 1; j < N; j++) T[i][j] = { 0.0, 0.0 };
    double h = 1.0 / m, x = 0.0, y = 0.0, t;
    // Граничні умови на ребрах (1) і (2)
    for (i = 0; i <= M; i++)
    {
        x = i * h;
        t = T1 * sin(al_1 * x); T[i][0] = { t, t };
```

```
        t = T2 * sin(al_2 * x); T[i][N] = { t, t };
    }
    // Граничні умови на ребрах (3) і (4)
    for (j = 0; j <= N; j++)
    {
        y = j * h;
        t = T3 * sin(bt_3 * y); T[0][j] = { t, t };
        t = T4 * sin(bt_4 * y); T[M][j] = { t, t };
    }
    s** d_T;
    cudaMalloc((void**) & d_T, (M+1) * sizeof(s*));
    s* temp;
    for (i = 0; i < M+1; i++)
    {
        cudaMalloc((void**)&temp, (N+1) * sizeof(s));
        cudaMemcpy(temp,     T[i],     (N+1)     *     sizeof(s),
cudaMemcpyHostToDevice);
        cudaMemcpy(&d_T[i],          &temp,          sizeof(s*),
cudaMemcpyHostToDevice);
    }
    int iters;
    double err, err1, err2;
    int *d_iters;
    double *d_err1, *d_err2, *d_err;
    cudaMalloc(&d_iters, sizeof(int));
    cudaMalloc(&d_err, sizeof(double));
    cudaMalloc(&d_err1, sizeof(double));
    cudaMalloc(&d_err2, sizeof(double));
    const int num_blocks = 15;
    dim3 gridDim(num_blocks, 1, 1);
    const int num_threads = 5;
    dim3 blockDim(num_threads, 1, 1);
    const int num_streams = 8;
    cudaStream_t streams[num_streams];
    cudaEvent_t start, stop;
    float elapsedTime;
    cudaEventCreate(&start);
    cudaEventCreate(&stop);
    cudaEventRecord(start, 0);
    auto start1 = std::chrono::steady_clock::now();
    for (int k = 0; k < num_streams; k++)
    {
        cudaStreamCreate(&streams[k]);
        Jacobi  <<  <  gridDim,  blockDim,  0,  streams[k]  >>  >
(k,num_streams,num_blocks,num_threads, T1, T2, T3, T4, d_T, ic,
jc,d_iters,d_err1,d_err2,d_err);
    }
    cudaDeviceSynchronize();
    auto end = std::chrono::steady_clock::now();
    cudaEventRecord(stop, 0);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elapsedTime, start, stop);
    for (int i = 0; i < M + 1; i++) {
```

```
        cudaMemcpy(&temp,           &d_T[i],           sizeof(s*),
cudaMemcpyDeviceToHost);
        cudaMemcpy(T[i],    temp,    (N    +    1)    *    sizeof(s),
cudaMemcpyDeviceToHost);
    }
    cudaMemcpy(&err,            d_err,            sizeof(double),
cudaMemcpyDeviceToHost);
    cudaMemcpy(&err1,           d_err1,           sizeof(double),
cudaMemcpyDeviceToHost);
    cudaMemcpy(&err2,           d_err2,           sizeof(double),
cudaMemcpyDeviceToHost);
    cudaMemcpy(&iters,           d_iters,           sizeof(int),
cudaMemcpyDeviceToHost);
    printf(" iters = %d  error = %8.1E  CPU time = %lld ms GPU
time: %d ms\n", iters, err,

std::chrono::duration_cast<std::chrono::milliseconds>(end       -
start1).count(), int(elapsedTime));
    printf(" T[%d,%d] = %12.8lf\n", ic, jc, T[ic][jc].t0);
    printf(" %10.6lf  %10.6lf  %12.8lf\n", xc, yc, plate.Txy(xc,
yc));
     /*Виведення результатів чисельного експерименту у файл для
подальшої обробки Surfer*/
    FILE* surfer;
    surfer = fopen("Map_Jx1.txt", "w");
    for (j = 0; j <= N; j++)
    {
        y = h * j;
        for (i = 0; i <= M; i++)
        {
            x = h * i; t = T[i][j].t0;
            char txt[100];
            snprintf(txt,    sizeof(txt),    "%10.6lf    %10.6lf
%12.8lf", x, y, t);
            fprintf(surfer, "%s\n", txt);
        }
    }
    fclose((FILE*)surfer);
    cudaFree(d_T);
    for (i = 0; i < M+1; i++)
    {
        delete[] T[i];
    }
    delete[] T;
    return 0;
}
```

ДОДАТОК В

Лістинг програмної реалізації проекту Map_Rect_Plate_01

Main_01.cs:

```
using System;
using RP = NExp_01.Rect_Plate;
using System.IO;

namespace Map_Rect_Plate_01
{
    class Main_01
    {
        static void Main(string[] args)
        {
            double T1 = 30.0, T2 = 20.0, T3 = -15.0,
                T4 = 15.0, a = 2, b = 1;
            int k1 = 5, k2 = 3, k3 = 1, k4 = 2;
            // Координати контрольної точки
            double xc = 1.9, yc = 0.1;
            RP plate = new RP(a, b, T1, T2, T3, T4, k1, k2, k3,
k4);
            double t = plate.Txy(xc, yc);
            Console.WriteLine($" t = {t,18:F8}");
            int m = 300, i, j, N, M;
            M = (int)(a * m); N = (int)(b * m);
            StreamWriter writer = new
StreamWriter("Map_01.txt");
            double h = 1.0 / m, x = 0.0, y = 0.0;
            string txt = "";
            for (j = 0; j <= N; j++)
            {
                y = h * j;
                for (i = 0; i <= M; i++)
                {
                    x = h * i; t = plate.Txy(x, y);
                    txt = $" {x,10:F6}  {y,10:F6} " +
                        $" {t,12:F8}".Replace(',', '.');
                    writer.WriteLine(txt);
                }
            }
            writer.Close();
        }
    }
}
```