МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені В. Н. КАРАЗІНА

МОДЕЛЮВАННЯ ФІЗИЧНИХ ПРОЦЕСІВ ІЗ ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ CUDA

Монографія

Харків – 2017

УДК 519.6, 004.272
М 74

Рецензенти:
В. О. Міщенко – д.т.н., доцент, професор кафедри моделювання систем і технологій ХНУ імені В. Н. Каразіна;
О. Г. Толстолузька – д.т.н., с.н.с., професор кафедри теоретичної та прикладної системотехніки ХНУ імені В. Н. Каразіна;
І. Г. Марченко – д.ф.-м.н., професор, провідний науковий співробітник ННЦ ХФТІ.

Затверджено до друку рішенням Вченої ради
Харківського національного університету імені В. Н. Каразіна
(протокол № 6 від 24.04.2017 р.)

Моделювання фізичних процесів із використанням технології CUDA : монографія / І. В. Гущин, В. М. Куклін, О. В. Мішин, О. В. Приймак. – Х. : ХНУ імені В. Н. Каразіна, 2017. – 116 с.
М 74
ISBN 978-966-285-420-6

Представлені результати моделювання явищ виникнення хвиль аномально великої амплітуди внаслідок розвитку модуляційної нестійкості. Вивчено характер зверхвипромінення електронних згустків, що рухаються в плазмі. Розроблено модель динаміки конвективного шару і структурно-фазових переходів, що виникають у ньому. Розглянуто особливості застосування технології CUDA для моделювання складних фізичних процесів. Обговорюються методи опису і підходи для представлення даних.

Видання призначене для фахівців у сфері моделювання фізичних явищ і студентів старших курсів природничо-наукових і комп'ютерних факультетів університетів.

УДК 519.6, 004.272

ISBN 978-966-285-420-6

ЗМІСТ

ВІД АВТОРІВ

Необхідність вирішення складних задач моделювання фізичних процесів призвела до того, що почалися пошуки методів, які б дозволили на звичайних комп'ютерах або кластерах, зібраних в комп'ютерну мережу, вирішувати нагальні завдання, які ставить перед дослідниками наука. Найбільш складні моделі явищ і процесів на сьогодні розроблені у фізиці. Мабуть, і в інших галузях наукового пізнання скоро з'являться задачі подібної складності, тому наведені нижче методи і підходи, безсумнівно, допоможуть дослідникам у їх настільки необхідній діяльності.

Моделювання є проміжною ланкою між аналітичними описами і обчислювальним експериментом, зберігаючи складність останнього і ясність першого. З одного боку, моделювання дозволяє просунутися у розумінні механізмів в значно більшому ступені, ніж під час обчислювального експерименту, при цьому жертвуючи точністю і широтою висвітлення динаміки процесу. З іншого боку, моделювання, як і аналітичний опис, дає можливість переглянути вплив різних параметрів і умов на розвиток процесу. Але моделювання має значно більше можливостей знайти розв'язки, що вимагають громіздких обчислень, у відносно короткий час і більш точно, не потребуючи великої кількості малообгрунтованих наближень і не орієнтуючись в такій мірі, як у аналітичних описах, на не завжди адекватні оцінки.

Паралельні обчислення дозволяють значно прискорити вирішення завдань моделювання, економлячи час дослідників і машинний час. Паралельні обчислення передбачають поділ комп'ютерних програм на підпрограми, які можуть виконуватися паралельно, тобто одночасно. Спочатку для обчислень була призначена основна частина комп'ютера – центральний процесор. В останні роки для обчислень все частіше використовують відеокарти (графічні процесори), основне призначення яких – підготовка і виведення зображення на екран комп'ютера.

Паралельні обчислення на процесорах і відеокартах можливі завдяки тому, що ці пристрої мають як мінімум кілька обчислювальних одиниць – ядер. Кожне ядро може виконувати обчислення незалежно від інших ядер. Тому можна забезпечити одночасне виконання підпрограм незалежно одна від іншої на різних ядрах. Останніми роками спостерігається зростання кількості ядер в процесорах і відеокартах. При цьому архітектури процесора і відеокарти відрізняються: процесор включає в себе одне або кілька ядер високої тактової частоти, а відеокарта – велику кількість ядер малої тактової частоти. Це робить відеокарту більш придатною для паралельних обчислень, тому що одночасно можна виконувати більшу кількість підпрограм.

Для зручності проведення паралельних обчислень стали створюватися відповідні технології. Такі технології спрощують розробку паралельних алгоритмів і забезпечують їх виконання на відеокартах. Однією з основних

технологій паралельних обчислень на відеокартах стала CUDA, створена компанією Nvidia – одним із головних розробників графічних процесорів. CUDA може використовуватися тільки в графічних процесорах Nvidia, які діляться на кілька сімейств, призначених для використання в певних пристроях: настільних комп'ютерах, ноутбуках, кластерах, суперкомп'ютерах, мобільних та інших пристроях. При цьому використання технології CUDA не залежить від сімейства графічних процесорів Nvidia.

У розділах 1–3 описані три фізичні задачі і результати їх моделювання. Ці задачі є задачами Коші. Для задач показано використання CUDA, розроблені паралельні алгоритми, вивчені можливості підвищення швидкості їх роботи за збереження точності обчислень, показано прискорення обчислень на CUDA порівняно з обчисленнями на центральних процесорах.

В 4-му розділі монографії наведені основи технології CUDA, її історія та перспективи розвитку, її переваги та недоліки, відмінності від обчислень на центральних процесорах і від інших технологій, які використовують графічні процесори.

У 5-му розділі показано, як створювати програми на CUDA двома способами – використовуючи CUDA-версії відомих математичних бібліотек функцій і програмуючи власні функції. Основна увага приділена створенню власних функцій як найпоширенішому способу використання CUDA. Коротко розглянуті інші можливості технології CUDA, використання яких не є обов'язковим під час написання програм, такі як динамічний паралелізм, уніфікована віртуальна пам'ять, використання декількох GPU для обчислень, атомарні операції, багатовимірні сітки і блоки. Розглянуті питання точності та оптимізації обчислень, що актуально під час моделювання наукових задач.

У 6-му розділі представлені паралельні алгоритми для обчислення систем диференційних рівнянь у вигляді задачі Коші, яка поширена під час опису фізичних процесів. Наведені алгоритми і програмний код на CUDA для двох відомих методів розв'язання задачі Коші – явних методів Ейлера і Рунге–Кутти.

У додатках А, Б, В розглянуто відпрацювання способів налаштування CUDA на комп'ютері користувача. У додатку А показано, як налаштувати операційну систему, щоб вона не переривала тривалі обчислення на графічному процесорі. У додатку Б показано налаштування CUDA для середовища розробки Visual Studio, що дає можливість написання CUDA програм мовами програмування C/C++. У додатку В представлено опис технології JCUDA і її налаштування для середовища розробки Eclipse. Використання JCUDA подібно до використання CUDA і дає можливість створення CUDA програм мовою програмування Java. Подано порівняння синтаксису команд CUDA і JCUDA.

Розділ 1 підготували О. В. Приймак і В. М. Куклін. Розділ 2 – О. В. Мішин і В. М. Куклін. Розділ 3 – І. В. Гущин і В. М. Куклін. Розділи 4–6 і додатки А–В підготував О. В. Приймак.

РОЗДІЛ 1

ВЕРИФІКАЦІЯ S-ТЕОРІЇ МОДУЛЯЦІЙНИХ НЕСТІЙКОСТЕЙ ХВИЛЬОВОГО ПОЛЯ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ CUDA

1.1. ПРОБЛЕМИ ОПИСУ НЕСТІЙКОСТІ ІНТЕНСИВНИХ ХВИЛЬОВИХ ПОЛІВ У ПРЕДМЕТНІЙ ОБЛАСТІ

Періодичні хвилі кінцевої амплітуди в середовищах із локальною кубічною нелінійністю є нестійкими зі збудженням двох бічних спектрів вимушених збурень, відповідно, з більшою і меншою довжиною хвилі [1–6]. Розвиток такої нестійкості призводить до амплітудної модуляції початкової хвилі [2]. Дослідження модуляційної нестійкості корисно для вивчення еволюції хвильового руху і транспортування хвильових пакетів. У консервативних середовищах із кубічною нелінійністю за відсутності загасання коливань формуються нелінійні утворення – автохвилі [7–10], зокрема солітони. Однак у відкритих системах із зовнішніми джерелами і поглинанням енергії виникають нові види нелінійних утворень (див., наприклад, [11–13]). Одними з перших вивчення відкритих нелінійних систем почали автори робіт [14–18], які сформулювали підходи до опису нелінійної стадії модуляційної нестійкості спінових хвиль на базі так званої S-теорії. Пізніше була запропонована модифікована модель S-теорії [19–22], яка дала змогу детально вивчити еволюцію хвильових пакетів інтенсивного хвильового руху різної природи.

Нижче проводиться верифікація застосування S-теорії для опису модуляційних нестійкостей шляхом порівняння результатів розрахунків на її основі з результатами розрахунків загальної теорії, яка враховує різні види взаємодії мод спектра [23; 24]. Мета роботи – за допомогою технології CUDA показати схожість цих двох підходів, порівняти характеристики процесу нестійкості, максимальні амплітуди модуляції (обвідної) і окремих хвиль, а також час їх появи.

1.2. МАТЕМАТИЧНІ МОДЕЛІ ОПИСУ МОДУЛЯЦІЙНИХ НЕСТІЙКОСТЕЙ ХВИЛЬОВОГО ПОЛЯ

За наявності джерела і стоку (розподіленого виводу, поглинання і дисипації) енергії хвилі рівняння Лайтхілла (різновид нелінійного рівняння Шредінгера) набуває вигляду

$$\frac{\partial A}{\partial t} = -\delta A - i\hat{f}A - i\hat{h}A \mid A \mid^2 + g, \tag{1.1}$$

де δ – декремент поглинання та g – зовнішнє джерело хвильової енергії, $\hat{f}$ та $\hat{h}$ – просторові оператори. Амплітуда коливань $A(t,x)$, що повільно змінюється з часом, може бути представлена у вигляді

$$\begin{aligned} A &= u_0(t)\exp\{i\phi_{k_0}(t) - ik_0x\} + \sum_{n\neq 0} u_n(t)\exp\{i\phi_{k_n}(t) - ik_nx\} = \\ &= \{u_0(t) + \sum_{n\neq 0} u_n(t)\exp[i(\phi_{k_n} - \phi_{k_0}) - i(k_n - k_0)x]\}\exp\{i\phi_{k_0}(t) - ik_0x\}. \end{aligned} \tag{1.2}$$

Тобто нестійкість розуміють як збудження спектра $\sum_{n\neq 0} u_n(t)\cdot\exp\{i\varphi_{k_n}(t)\}\cdot\exp\{i\omega_0 t - ik_nx\}$, де $u_n(t)\cdot\exp\{i\varphi_{k_n}(t)\}$ – повільно змінна комплексна амплітуда n моди спектра. Сумарне поле являє собою модульовану хвилю на частоті ω_0.

Дійсно, виділяючи «швидкий» фазовий множник $\exp\{i\omega_0 t - ik_0x\}$, що відповідає основній хвилі, отримаємо в цьому випадку поле коливань, як добуток

$$A = \exp\{i\omega_0 t - ik_0x\}\cdot\{u_0\exp[i\varphi_{k_0}] + \sum_{n\neq 0} u_n\exp[i\varphi_{k_n} - i(k_n - k_0)x]\}, \tag{1.3}$$

де $\exp\{i\omega_0 t - ik_0x\}$ – швидко змінна фаза.

Обвідну цього хвильового процесу можна визначити наступним чином. Позначимо швидку фазу $\omega_0 t - k_0x + \phi_{k_0} = \varphi$, а повільну $(\varphi_{k_n} - \varphi_{k_0}) - (k_n - k_0)x = \psi_n$. Тоді, наприклад,

$$\mathrm{Re}\, A = \overline{A}\cdot\{Cos\varphi\cdot Sin\overline{\varphi} - Sin\varphi\cdot Cos\overline{\varphi}\} = -\overline{A}\cdot Sin(\varphi - \overline{\varphi}),$$

де $Sin\overline{\varphi} = (u_0 + \sum_{n\neq 0} u_n Cos\psi_n)/\overline{A}$, $Cos\overline{\varphi} = \sum_{n\neq 0} u_n Sin\psi_n/\overline{A}$, причому амплітуда обвідної

$$\overline{A} = \sqrt{(u_0 + \sum_{n\neq 0} u_n Cos\psi_n)^2 + (\sum_{n\neq 0} u_n Sin\psi_n)^2}.$$

Або, повертаючись до колишніх позначок,

$$\begin{aligned} \overline{A} = \{&(u_0 + \sum_{n\neq 0} u_n Cos[(\varphi_{k_n} - \varphi_{k_0}) - (k_n - k_0)x])^2 + \\ &+(\sum_{n\neq 0} u_n Sin[(\varphi_{k_n} - \varphi_{k_0}) - (k_n - k_0)x])^2\}^{1/2}. \end{aligned} \tag{1.4}$$

На лінійній за амплітудами збурень стадії модуляційної нестійкості збуджується спектр коливань, хвильові числа яких розташовуються симетрично щодо хвильового числа k_0 основної моди кінцевої амплітуди $k_n \equiv k_0 + K_n > k_0$ та $k_{-n} \equiv k_0 - K_n < k_0$, де $n > 0$. Кожна пара симетрично розташованих відносно основної хвилі мод k_n, k_{-n} безпосередньо взаємодіє з полем основної хвилі, причому виконується співвідношення $k_n + k_{-n} = 2k_0$. Це зумовлено видом нелінійності.

З урахуванням особливості модуляційної нестійкості була побудована так звана S-теорія, яка враховувала взаємодію тільки «спарених» мод спектра, хвильові вектора яких симетрично розташовувалися щодо хвильового вектора основної хвилі кінцевої амплітуди ($2k_0 = k_S + k_{-S} = k_n + k_{-n}$).

Більш загальний опис дозволяє у виразі для нелінійного доданку виду $\{A|A|^2\}$ в рівнянні (1.1) утримувати всі складові, не обмежуючись симетричними по відношенню до накачування модами спектра (тобто $2k_0 = k_S + k_{-S} = k_n + k_{-n}$), які використовуються для формування S-теорії.

Опис обвідної хвильового поля в середовищах зі слабкою дисперсією. Подібні хвилі великої амплітуди збуджуються у хвилеводах, як вакуумних, так і заповнених діелектриком або плазмою. Рівняння Лайтхілла (1) для комплексної амплітуди хвильового поля можна отримати [19], вважаючи $\hat{f} = \partial^2 / \partial x^2$ та $\hat{h} = 1$. Рівняння для амплітуди і фази малих мод $u_n(t) \cdot \exp\{i\varphi_n(t)\}$ в цьому випадку можна записати у вигляді

$$\frac{\partial u_n}{\partial \tau} = -\delta u_n - \beta_{1L} \mathrm{A}_{n,1} - \beta_{2L} \mathrm{A}_{n,2}, \tag{1.5}$$

$$u_n \frac{\partial \varphi_n}{\partial \tau} = K_n^2 \cdot u_n - \beta_{1L} \mathrm{B}_{n,1} - \beta_{2L} \mathrm{B}_{n,2}, \tag{1.6}$$

а для амплітуди і фази основної хвилі

$$\frac{\partial u_0}{\partial \tau} = -\delta u_0 - \beta_{1L} \mathrm{A}_{0,1} - \beta_{2L} \mathrm{A}_{0,2} + G_0, \tag{1.7}$$

$$u_0 \frac{\partial \varphi_0}{\partial \tau} = -\beta_{1L} \mathrm{B}_{0,1} - \beta_{2L} \mathrm{B}_{0,2}, \tag{1.8}$$

де $G_{k_n} \equiv G_n = 0, k_n \neq k_0, n \neq 0$; $G_{k_n} \equiv G_n = \delta, k_n = k_0, n = 0$, $\varphi_n = \varphi_{k_n}$.

Причому для $A_{i,j}$ и $B_{i,j}$ справедливі вирази:

$$A_{0,1} = +u_0 \sum_{m\neq 0} u_m u_{-m} Sin(2\varphi_0 - \varphi_m - \varphi_{-m})], \tag{1.9}$$

$$B_{0,1} = [u_0^3 + 2u_0 \sum_{m\neq 0} u_m^2 + u_0 \sum_{m\neq 0} u_m u_{-m} Cos(\varphi_m + \varphi_{-m} - 2\varphi_0)], \tag{1.10}$$

$$A_{0,2} = -\sum_m \sum_p u_{-m-p} u_{-p} u_m \cdot Sin(\varphi_{-m-p} - \varphi_{-p} + \varphi_m - \varphi_0) - A_{0,1}, \tag{1.11}$$

$$B_{0,2} = +\sum_m \sum_p u_{-m-p} u_{-p} u_m \cdot Cos(\varphi_{-m-p} - \varphi_{-p} + \varphi_m - \varphi_0) - B_{0,1}, \tag{1.12}$$

$$\begin{aligned} A_{n,1} = -[u_{-n} u_0^2 Sin(2\varphi_0 - \varphi_n - \varphi_{-n}) + \\ + u_{-n} \sum_{m\neq 0,n} u_m u_{-m} Sin(\varphi_m + \varphi_{-m} - \varphi_n - \varphi_{-n})], \end{aligned} \tag{1.13}$$

$$\begin{aligned} B_{n,1} = [u_n u_n^2 + 2u_n u_0^2 + 2u_n \sum_{m\neq 0,n} u_m^2 + u_{-n} u_0^2 Cos(2\varphi_0 - \varphi_n - \varphi_{-n}) + \\ + u_{-n} \sum_{m\neq 0,n} u_m u_{-m} Cos(\varphi_m + \varphi_{-m} - \varphi_n - \varphi_{-n})], \end{aligned} \tag{1.14}$$

$$A_{n,2}\,|_{n\neq 0} = -\sum_m \sum_p u_{n-m-p} u_{-p} u_m \cdot Sin(\varphi_{n-m-p} - \varphi_{-p} + \varphi_m - \varphi_n) - A_{n,1}, \tag{1.15}$$

$$B_{n,2}\,|_{n\neq 0} = +\sum_m \sum_p u_{n-m-p} u_{-p} u_m \cdot Cos(\varphi_{n-m-p} - \varphi_{-p} + \varphi_m - \varphi_n) - B_{n,1}, \tag{1.16}$$

де $-N < n < N = 200$. Для нормування амплітуди основної хвилі на одиницю на початковій стадії процесу тут і нижче був обраний рівень зовнішнього накачування $G = \delta$. За $\beta_{1L} = 1$ та $\beta_{2L} = 0$ приходимо до випадку S-теорії (а) [19], за $\beta_{1L} = 1$ та $\beta_{2L} = 1$ система рівнянь є наслідком повного рівняння Лайтхілла, тобто без спрощень S-теорії із цим типом дисперсійного доданка (б). Просторово-часова структура хвильового поля та його обвідної може бути представлена виразами (1.3) і (1.4).

Опис хвильового поля в середовищах із сильною дисперсією. Розглянемо для визначеності випадок модуляційної нестійкості гравітаційних поверхневих хвиль на глибокій воді у формі [21–22], що представляють особливий інтерес для судноплавства в районах із високим

рівнем океанського хвилювання. Для частоти хвиль великої амплітуди [25] справедливий наступний вираз:

$$\omega = kW = \sqrt{gk} \cdot \{1 + \frac{A^2k^2}{2} + ...\}, \tag{1.17}$$

де A – відхилення поверхні, W – швидкість хвилі, g – прискорення вільного падіння. Рівняння Лайтхілла–Невілла (1.1) для комплексної амплітуди окремої моди спектра хвильового поля при цьому представляється у вигляді [21–22]

$$\frac{\partial A_K}{\partial t} = -\delta A_K - i[\sqrt{g(k_0+K)} - \sqrt{gk_0}]A_K - i\sqrt{g(k_0+K)}\frac{(k_0+K)^2}{2}\{|A|^2 A\}_K . \tag{1.18}$$

Рівняння для амплітуди і фази малих мод можна записати у вигляді

$$\frac{\partial u_n}{\partial \tau} = -\delta u_n - \beta_1 A_{n,1} - \beta_2 A_{n,2}, \tag{1.19}$$

$$u_n \frac{\partial \varphi_n}{\partial \tau} = -2\{\frac{\sqrt{1+n\cdot\Delta}-1}{\alpha}\}u_n - \beta_1 B_{n,1} - \beta_2 B_{n,2}. \tag{1.20}$$

Рівняння для амплітуди і фази моди основної хвилі:

$$\frac{\partial u_0}{\partial \tau} = -\delta u_0 - \beta_1 A_{0,1} - \beta_2 A_{0,2} + G_0, \tag{1.21}$$

$$u_0 \frac{\partial \varphi_0}{\partial \tau} = -\beta_1 B_{0,1} - \beta_2 B_{0,2}, \tag{1.22}$$

де

$$A_{0,1} = -[u_0 \sum_{m\neq 0,n} u_m u_{-m} Sin(\varphi_m + \varphi_{-m} - 2\varphi_0)], \tag{1.23}$$

$$B_{0,1} = [u_0 u_0{}^2 + 2u_0 \sum_{m\neq 0} u_m{}^2 + u_0 \sum_{m\neq 0} u_m u_{-m} Cos(\varphi_m + \varphi_{-m} - 2\varphi_0)], \tag{1.24}$$

$$A_{0,2} = -\sum_m \sum_p u_{-m-p} u_{-p} u_m \cdot Sin(\varphi_{-m-p} - \varphi_{-p} + \varphi_m - \varphi_0) - A_{0,1}, \tag{1.25}$$

$$B_{0,2} = +\sum_m \sum_p u_{-m-p} u_{-p} u_m \cdot Cos(\varphi_{-m-p} - \varphi_{-p} + \varphi_m - \varphi_0) - B_{0,1}, \qquad (1.26)$$

$$A_{n,1} = -(1+n\cdot\Delta)^{5/2}[u_{-n} {u_0}^2 Sin(2\varphi_0 - \varphi_n - \varphi_{-n}) + \\ + u_{-n} \sum_{m\neq 0,n} u_m u_{-m} Sin(\varphi_m + \varphi_{-m} - \varphi_n - \varphi_{-n})], \qquad (1.27)$$

$$B_{n,1} = (1+n\cdot\Delta)^{5/2}[u_n {u_n}^2 + 2u_n {u_0}^2 + \\ + 2u_n \sum_{m\neq 0,n} {u_m}^2 + u_{-n} {u_0}^2 Cos(2\varphi_0 - \varphi_n - \varphi_{-n}) + \\ + u_{-n} \sum_{m\neq 0,n} u_m u_{-m} Cos(\varphi_m + \varphi_{-m} - \varphi_n - \varphi_{-n})], \qquad (1.28)$$

$$A_{n,2}\,|_{n\neq 0} = -(1+n\cdot\Delta)^{5/2} \sum_m \sum_p u_{n-m-p} u_{-p} u_m \\ Sin(\varphi_{n-m-p} - \varphi_{-p} + \varphi_m - \varphi_n) - A_{n,1}, \qquad (1.29)$$

$$B_{n,2}\,|_{n\neq 0} = +(1+n\cdot\Delta)^{5/2} \sum_m \sum_p u_{n-m-p} u_{-p} u_m \\ Cos(\varphi_{n-m-p} - \varphi_{-p} + \varphi_m - \varphi_n) - B_{n,1}. \qquad (1.30)$$

За $\beta_{1L} = 1$ та $\beta_{2L} = 0$ приходимо до випадку S-теорії [21–22], за $\beta_{1L} = 1$ та $\beta_{2L} = 1$ система рівнянь є наслідком повного рівняння Лайтхілла виду (1.18), тобто без спрощень S-теорії. Просторово-часова структура хвильового поля в системі спокою основної хвилі може бути представлена виразом

$$E(\xi,\tau) = u_0 Cos(\xi + \varphi_0) + \\ + \sum_{\substack{n\neq 0 \\ n>0}} [u_n \cdot Cos\{-2\frac{\sqrt{(1+n\cdot\Delta)} - (1+n\cdot\Delta)}{\alpha}\tau + (1+n\cdot\Delta)\cdot\xi + \varphi_n\} + \\ + u_{-n} \cdot Cos\{2\frac{\sqrt{(1-n\cdot\Delta)} - (1-n\cdot\Delta)}{\alpha}\tau + (1-n\cdot\Delta)\cdot\xi + \varphi_{-n}\}], \qquad (1.31)$$

де $k_0 x = \xi$ і область простору розгляду $-\pi/\Delta < \xi < \pi/\Delta$, $\alpha = {k_0}^2 \mid A_0 \mid^2$, $\tau = t\cdot\omega_0 \frac{{k_0}^2 \mid A_0 \mid^2}{2} = t\cdot\sqrt{gk_0}\,\frac{{k_0}^2 \mid A_0 \mid^2}{2}$, $k_n / k_0 = 1 + n\cdot\Delta$, $-N < n < N = 200$.

1.3. ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ CUDA ДЛЯ ОПИСУ ПРОЦЕСІВ МОДУЛЯЦІЙНОЇ НЕСТІЙКОСТІ

Для виконання мети роботи проводиться низка обчислювальних експериментів і аналіз результатів. Розроблено програмне забезпечення, що реалізує представлені вище математичні моделі. Для середовища зі слабкою дисперсією визначається поведінка основної хвилі і її енергії, спектрів нестійкості і їх енергії, амплітуди обвідної хвильового поля. Для середовища із сильною дисперсією визначається поведінка основної хвилі, спектрів нестійкості, розподіл розмахів (відстаней між гребенем і западиною) хвиль і визначення величини і частоти появи аномальних розмахів [23; 24]. Обчислювальний експеримент являє собою розв'язання задачі Коші методом Ейлера. Обчислювальні експерименти проводяться на графічному процесорі GeForce GT630 (GV-N630D3-2GL) за допомогою технології JCUDA.

Наведемо алгоритм використання CUDA в обчислювальному експерименті, деталізувавши блок паралельних обчислень на GPU для одного кроку за часом (рис. 1.1). Для середовищ із сильною і зі слабкою дисперсією алгоритм однаковий, тому що в цих моделях присутні одні й ті ж рівняння. Розрахунок інших параметрів моделей, наприклад, просторово-часової структури хвильового поля $E(\xi,\tau)$, не проводитиметься на GPU, тому що їх розрахунок відбувається не кожен крок часу, а тільки у разі отримання результатів обчислень із пам'яті GPU (це відбувається 200 разів за одну симуляцію).

Для алгоритму (рис. 1.1) обрані наступні параметри розпаралелювання. Для рівнянь $A_{n,1}, A_{n,2}, B_{n,1}, B_{n,2}, \partial u_n, \partial\varphi_n, u_n, \varphi_n$, де $-N < n < N = 200$ створюється сітка з одного блоку і 401 потоку, тому що число рівнянь $N + N + 1 = 200 + 200 + 1 = 401$ (400 і одне рівняння для $n = 0$, хоча значення $A_{n,1}, A_{n,2}, B_{n,1}, B_{n,2}, \partial u_n, \partial\varphi_n, u_n, \varphi_n$ для $n = 0$ беруться такими, що дорівнюють нулю). Для рівнянь $A_{0,1}, A_{0,2}, B_{0,1}, B_{0,2}, \partial u_0, \partial\varphi_0, u_0, \varphi_0$ (ці рівняння описують випадок $n = 0$ і обчислюються окремо від інших значень n) створюється сітка з одного блоку і одного потоку, тобто рівняння обчислюються паралельно.

Всі обчислення виконуються в подвійній точності. Однак їх можна прискорити, майже не вплинувши на точність результатів, шляхом обчислення тригонометричних функцій синуса і косинуса в урізаній точності (замінивши функції $\sin(x), \cos(x)$ на $\_\_\sin f(x), \_\_\cos f(x)$).

Далі порівнюється час обчислення рівнянь для одного кроку за часом для трьох випадків $-N < n < N = 10$, $-N < n < N = 50$, $-N < n < N = 200$ для середовища з сильною дисперсією. З ростом числа

мод зростає і перевага GPU над CPU (рис. 1.2). Слід зазначити, що оскільки мова програмування Java повільніше за мову Сі, то під час обчислення на Сі перевага GPU над CPU буде в кілька разів меншою.

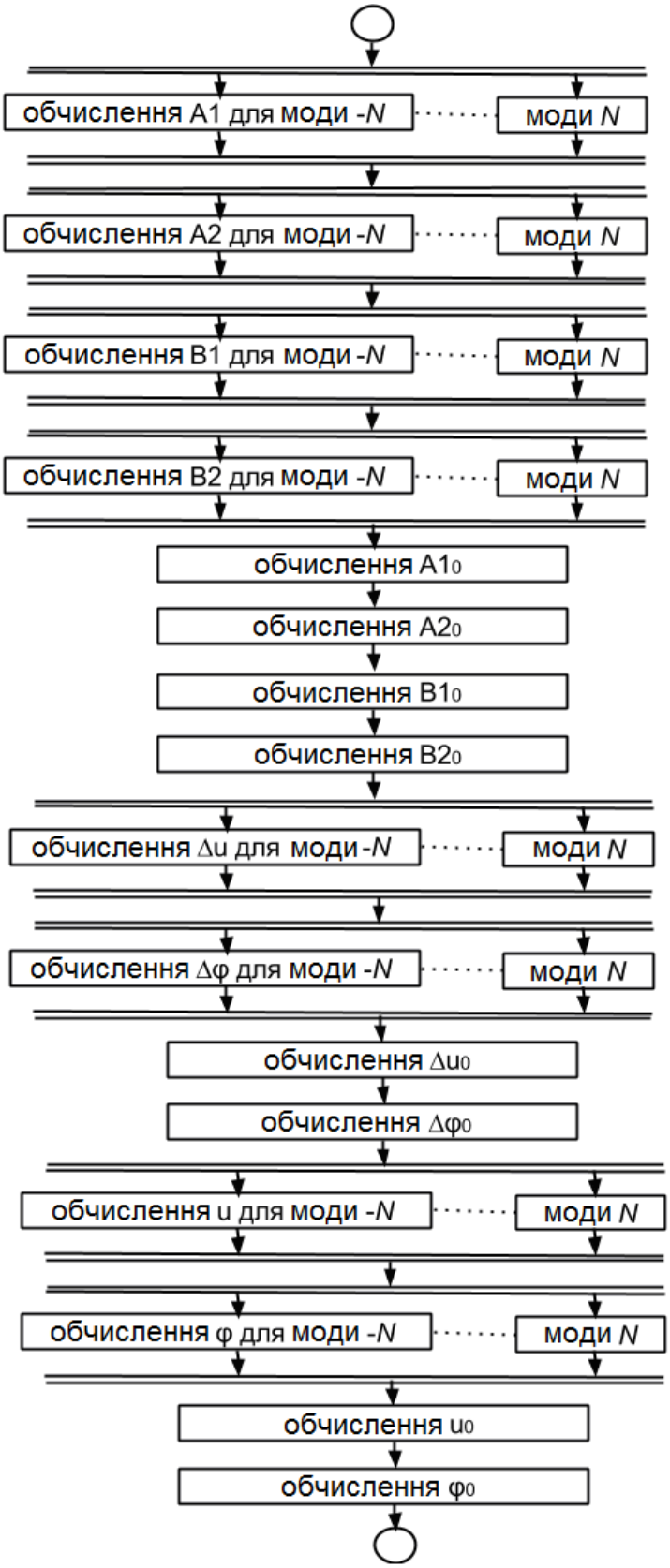

Рис. 1.1. Деталізація алгоритму обчислень на GPU для одного кроку за часом

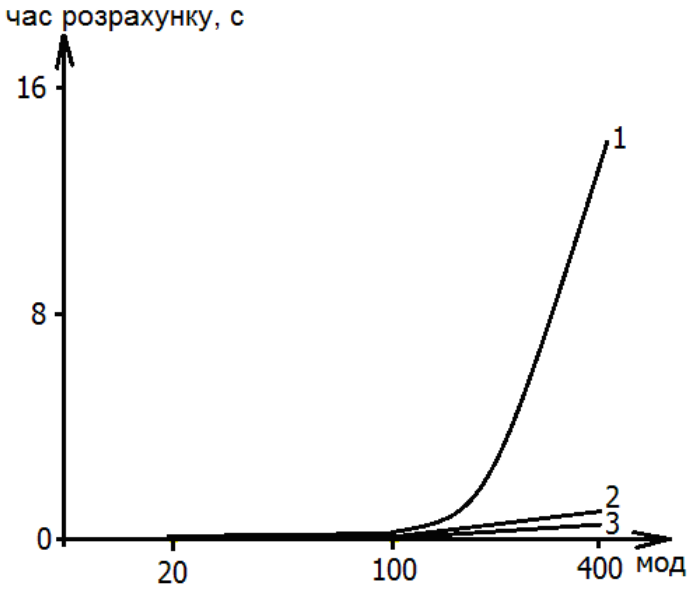

Рис. 1.2. Час обчислення рівнянь на GPU для одного кроку за часом (1 – на Java, 2 – на CUDA, 3 – на CUDA з тригонометричними функціями урізаної точності)

Можна вдосконалити алгоритм (рис. 1.1) шляхом об'єднання сіток потоків в одну сітку. Окремі рівняння, які розраховуються не паралельно, наприклад $\partial u_0, \partial \varphi_0$, можна об'єднати в одну сітку і обчислювати паралельно. Але якщо так зробити з рівняннями $A_{0,1}, A_{0,2}, B_{0,1}, B_{0,2}, \partial u_0, \partial \varphi_0, u_0, \varphi_0$, то швидкість обчислень не збільшиться ні на жодну мілісекунду, тому що рівнянь мало і вони складаються з незначної кількості обчислювальних операцій. Якщо зробити об'єднання сіток потоків під час обчислення рівнянь $\partial u_n, \partial \varphi_n, u_n, \varphi_n$, то швидкість обчислень цих рівнянь також не збільшиться через їх простоту.

Має сенс зробити об'єднання сіток потоків під час обчислення рівнянь $A_{n,1}, A_{n,2}, B_{n,1}, B_{n,2}$. Оскільки результати обчислення $A_{n,1}, B_{n,1}$ використовуються для обчислення $A_{n,2}, B_{n,2}$, то спочатку можна одночасно обчислити $A_{n,1}, B_{n,1}$, а потім одночасно $A_{n,2}, B_{n,2}$ (рис. 1.3).

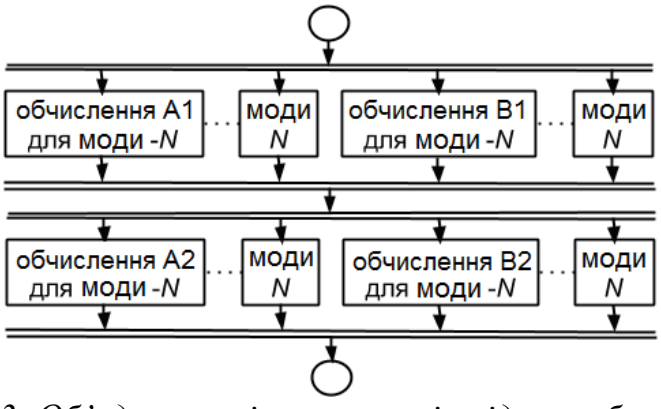

Рис. 1.3. Об'єднання сіток потоків під час обчислення рівнянь $A_{n,1}, A_{n,2}, B_{n,1}, B_{n,2}$

У випадку об'єднання сіток кожне рівняння $A_{n,1}, A_{n,2}, B_{n,1}, B_{n,2}$ залишається в окремому блоці, як це було раніше. Однак тепер два блоки потоків $A_{n,1}, B_{n,1}$ уміщуються в одну сітку, те ж відбувається і з $A_{n,2}, B_{n,2}$. Для обчислення, наприклад, $A_{n,1}, B_{n,1}$, спочатку на GPU запускається функція «function1», яка запускає обчислення $A_{n,1}, B_{n,1}$ (функцію «function 2»). Всі потоки блоку з індексом 0 будуть обчислювати рівняння $A_{n,1}$, потоки блоку з індексом 1 будуть обчислювати рівняння $B_{n,1}$.

```
__device__ void function2(int n, int block, double *A1, double *B1,
< інші параметри >){
        if(block == 0) A1[n] = <результат обчислення>;
        else if(block == 1) B1[n] = <результат обчислення>;
}
__global__ void function1(double *A1, double *B1, <інші
параметри>){
        function2(threadIdx.x, blockIdx.x, A1, B1, <інші параметри>);
}
```

В результаті під час об'єднання сіток потоків час обчислення рівнянь для одного кроку за часом для середовища з сильною дисперсією для випадку $-N < n < N = 50$ знизиться з 62 до 45 мілісекунд, для випадку $-N < n < N = 200$ знизиться з 1003 до 730 мілісекунд.

1.4. РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

Обвідна хвильового поля в середовищах зі слабкою дисперсією. Практичний інтерес пов'язаний із транспортуванням хвиль великої амплітуди за малих рівнів поглинання далеко від порога $\delta = \delta_{thr} = 1$. Характерні часи модуляції амплітуди основної хвилі за рівня поглинання $\delta = 0.1$ під час обліку всіх видів взаємодії мод стають менш регулярними (рис. 1.4).

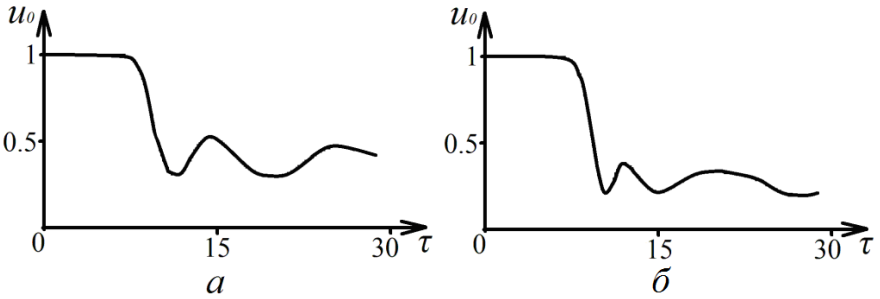

Рис. 1.4. Поведінка амплітуди основної хвилі з часом для випадків застосування S-теорії (а) і розгляду без наближень (б) за $\delta = 0.1,\ N = 200$

Залежності енергії спектра мод $\sum_{m\neq 0} u_m^{\,2}$ і енергії основної хвилі $u_0^{\,2}$ від часу в процесі розвитку нестійкості представлені на рис. 1.5. Явно помітний осциляторний характер обміну енергією між основною хвилею і спектром нестійких мод.

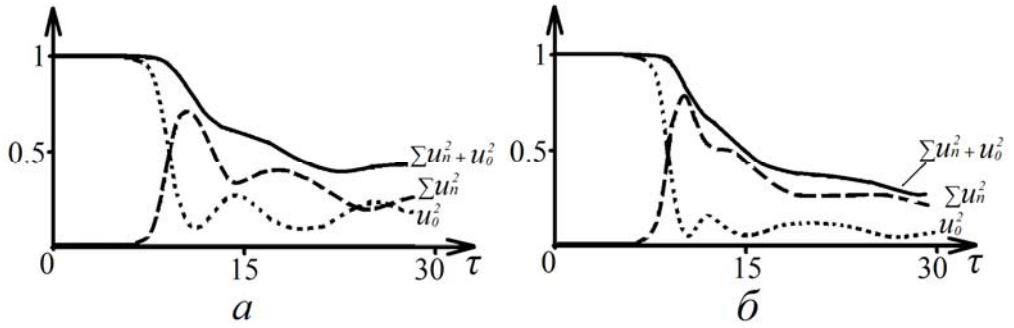

Рис. 1.5. Залежності енергії спектра мод $\sum_{m\neq 0} u_m^{\,2}$*, енергії основної хвилі* $u_0^{\,2}$ *і їх суми від часу для випадків застосування S-теорії (а) і розгляду без наближень (б) за* $\delta = 0.1,\ N = 200$

Максимуми амплітуди обвідної в обох випадках досягаються практично в один час і приблизно рівні один одному (рис. 1.6).

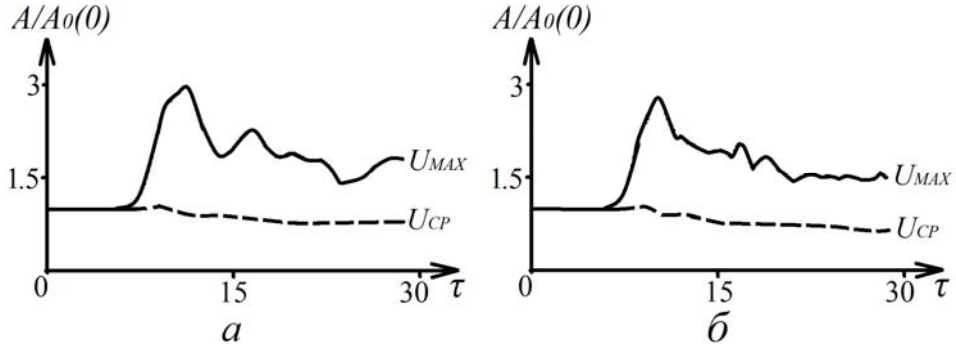

Рис. 1.6. Максимальна (суцільна крива) і середня (пунктир) амплітуди обвідної хвильового поля для випадків застосування S-теорії (а) і розгляду без наближень (б) за $\delta = 0.1,\ N = 200$

Змінюється і поведінка спектра нестійкості. Характерний для модуляційної нестійкості двогорбий спектр у разі опису в рамках S-теорії (а) звужується, а під час розгляду без наближень (б) спостерігається протилежна тенденція до розширення. Часи розгляду спектра обрані на лінійній за амплітудами збурень стадії процесу в момент досягнення максимальної амплітуди обвідної хвильового поля і в стадії розвинутої нестійкості (рис. 1.7).

В умовах слабкого поглинання енергія спектра модуляційної нестійкості досягає значень, що можна порівняти з початковою енергією хвилі кінцевої амплітуди. З рис. 1.6 видно, що на початковій стадії

нелінійного режиму процесу можлива поява хвиль і сплесків обвідної з вельми великою амплітудою. Надалі відбувається зниження амплітуди основної хвилі (рис. 1.5, 1.6) і її вплив на інтерференцію мод спектра послаблюється, а амплітуда мод зменшується.

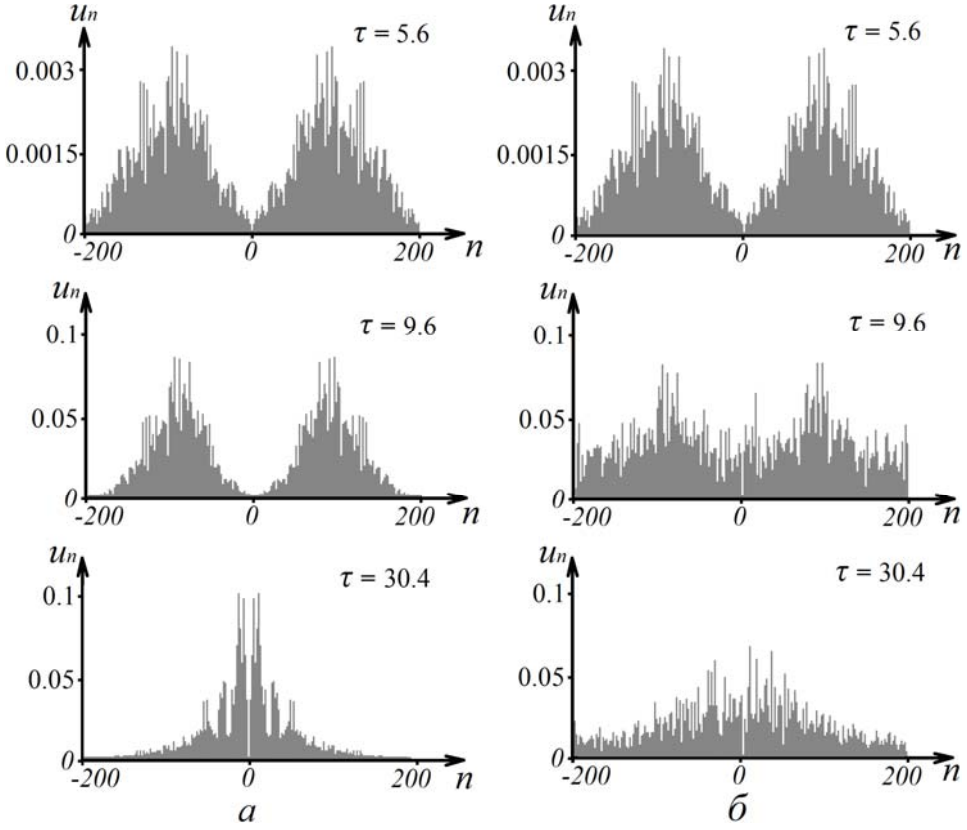

Рис. 1.7. Спектри нестійкості для трьох моментів часу в разі опису в рамках S-теорії (а) і розгляду без наближень (б) за $\delta = 0.1,\ N = 200$

Характер модуляції основної хвилі в просторі (фрагмент поблизу області з максимальною амплітудою обвідної) для тих же моментів часу для двох випадків опису нестійкості представлений на рис. 1.8.

З результатів чисельного моделювання випливає, що на початковій стадії процесу інтенсивність поля в області максимуму обвідної хвиль приблизно на порядок перевершує середній рівень інтенсивності.

Хвильове поле в середовищах із сильною дисперсією. Океанські хвилі. В реальних умовах поглинання енергії гравітаційних хвиль великої амплітуди на поверхні океану вельми мало. Тому раціональним буде проводити порівняння двох моделей опису для реалістичного випадку дуже малого поглинання $\delta = 0.01$ і значної амплітуди хвилювання $\Delta = \dfrac{0.566}{N}; \alpha = 0.01$. Для аналізу розмахів хвиль (тобто відстані між верхньою точкою гребеня хвилі і нижньою точкою западини) виділимо з них третину найбільших. Критерій, за яким виділяють аномально великі хвилі, зазвичай

$$U_{AG} > 2U_{SWH}, \tag{1.32}$$

де U_{AG} – аномальна хвиля, U_{SWH} – середнє значення розмаху третини найбільших розмахів.

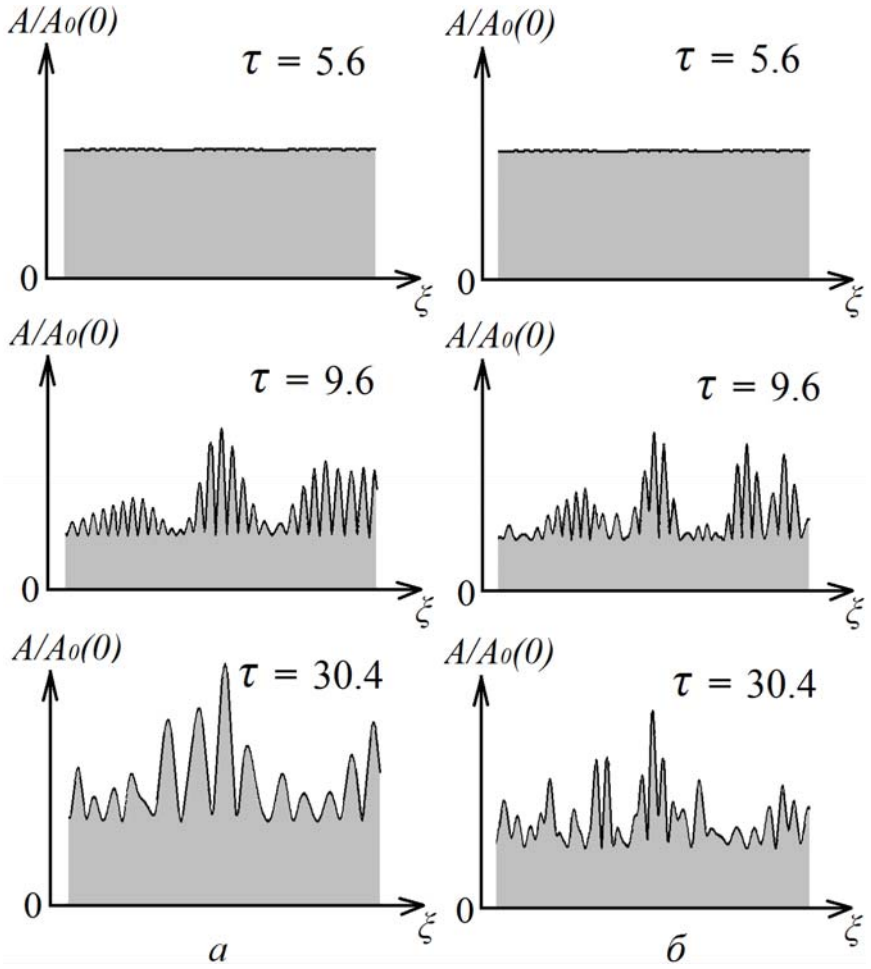

Рис. 1.8. Модуляція основної хвилі для трьох моментів часу в разі опису в рамках S-теорії (а) і розгляду без наближень (б) за $\delta = 0.1,\ N = 200$

На рис. 1.9–1.13 наведемо результати розрахунків, що демонструють розвиток спектру нестійкості для трьох моментів часу в разі опису в рамках S-теорії (а) і в загальному випадку розгляду без наближень (б).

Добре видно формування характерного двогорбого спектра (рис. 1.9) модуляційної нестійкості. Якщо в разі опису в рамках S-теорії така форма спектра зберігається, то в загальному випадку опису спектр з розвитком нестійкості згладжується.

З рис. 1.10 видно, що нерезонансні взаємодії, для яких не виконані співвідношення S-теорії, призводять до зриву осциляторного режиму поведінки амплітуди основної хвилі, характерного для режиму резонансної взаємодії, що описується S-теорією [21–22].

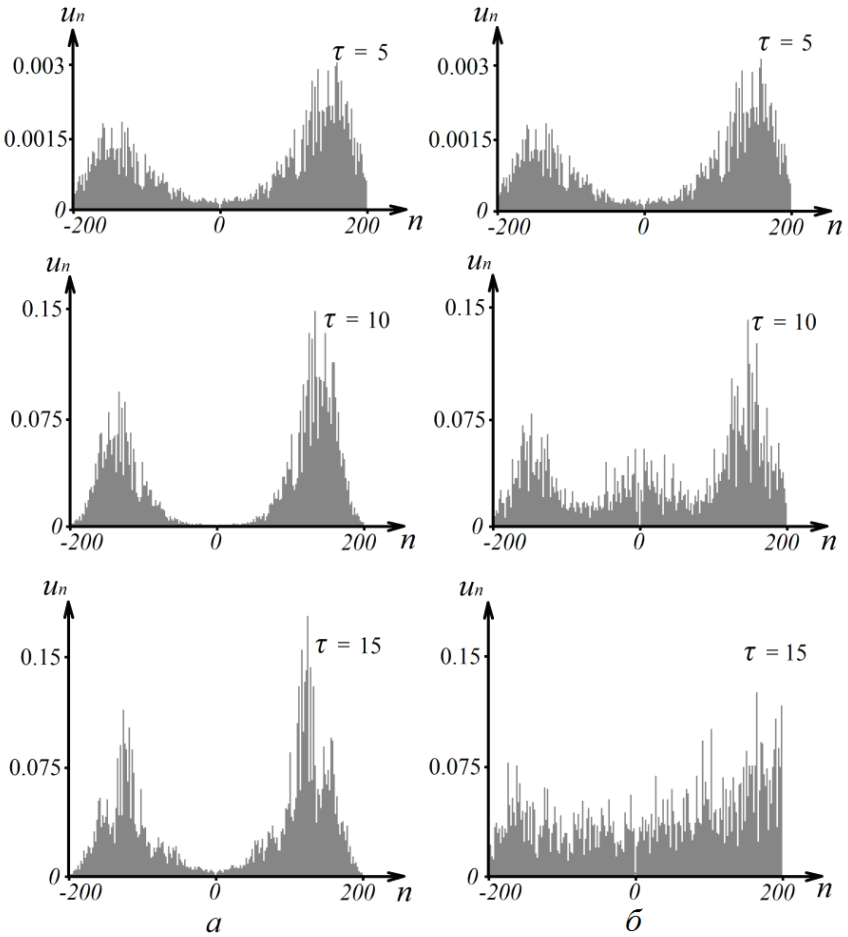

Рис. 1.9. Спектр нестійкості для трьох моментів часу в разі опису в рамках S-теорії (а) і в загальному випадку розгляду без наближень (б)

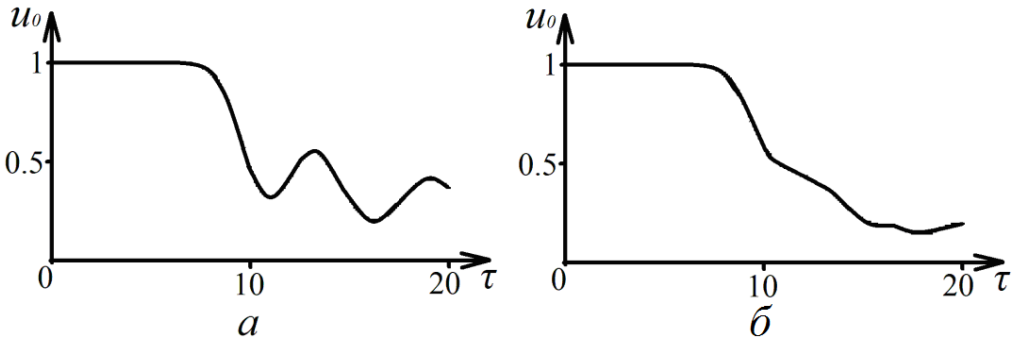

Рис. 1.10. Зміна амплітуди основної хвилі з часом під час опису в рамках S-теорії (а) і в загальному випадку розгляду без наближень (б)

Розподіли амплітуд розмахів H, тобто відстаней між верхньою точкою гребеня хвилі і нижньою точкою западини, в режимі розвиненої нестійкості за одну симуляцію представлені на рис. 1.12. Підрахунок хвиль відбувався через моменти часу, які приблизно дорівнюють часу життя аномально великої хвилі, на відміну від випадку [21–22], де хвилі з різними амплітудами підраховувалися в численних симуляціях. У двох цих випадках частоти появи аномальних хвиль у статистиці за ансамблями

і за часом в обох моделях опису океанського хвилювання практично не відрізняються (одна аномальна хвиля на 15–20 тисяч хвиль).

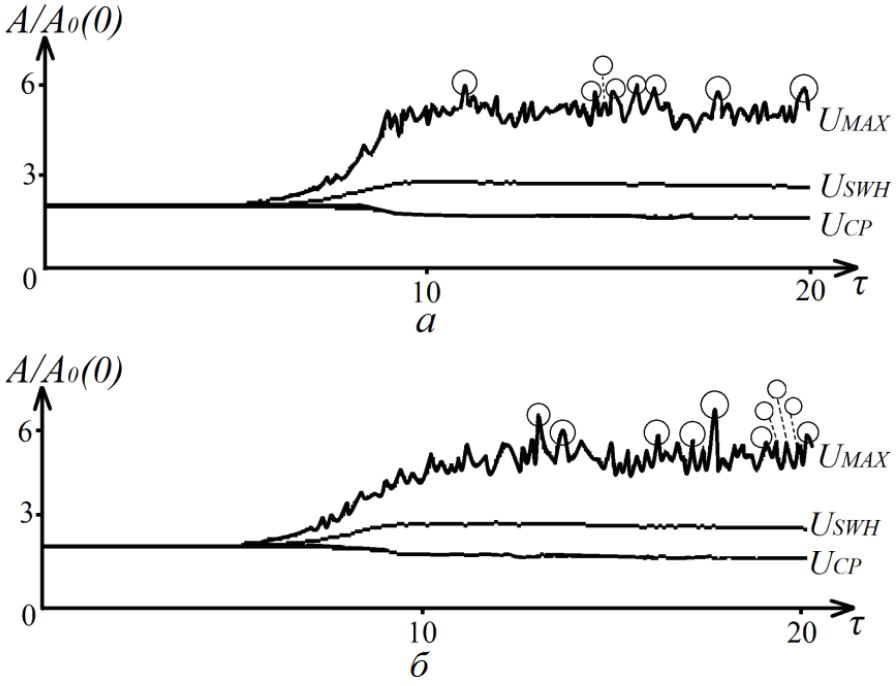

Рис. 1.11. Середня амплітуда U_{CP}, середня амплітуда третини найбільших мод U_{SWH} і найбільший розмах хвилі з ансамблю U_{Max} як функції часу. Колами відзначена поява хвиль аномальної амплітуди U_{AG} (32) в рамках S-теорії (а) і в загальному випадку розгляду без наближень (б)

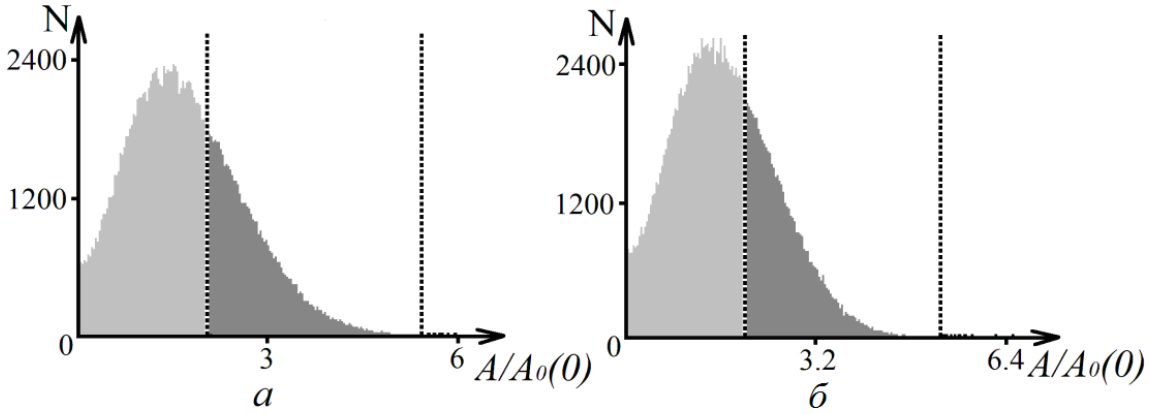

Рис. 1.12. Розподіл амплітуд розмахів за одну симуляцію у разі опису в рамках S-теорії (а) і в загальному випадку розгляду без наближень (б). Пунктирні лінії визначають межу між модами малої амплітуди і третиною найбільших мод і величиною, що в два рази перевищує середнє значення від третини найбільших мод; а – всього розмахів 173526, третина найбільших розмахів 57842, розмахів у 2 рази більших середнього від третини найбільших розмахів 8; б – всього розмахів 176386, третина найбільших розмахів 58795, розмахів у 2 рази більше середнього від третини найбільших розмахів 10

Характер розподілу розмахів подібний до [21–22], де їх кількість підраховувалася в різних симуляціях і величини усереднювалися за ансамблем симуляцій. Кількість і розподіл розмахів виявлених хвиль аномальної амплітуди представлені в табл. 1.1. Слід звернути увагу на наявність виражених «хвостів» розподілів в обох випадках.

Таблиця 1.1

Кількість і розподіл розмахів виявлених хвиль аномальної амплітуди

$U_{AG} / 2U_{SWH}$	Випадок а: $\beta_1 = 1, \beta_2 = 0$	Випадок б: $\beta_1 = 1, \beta_2 = 1$
від 2 до 2.1	4	7
від 2.1 до 2.2	2	1
від 2.2 до 2.3	2	-
від 2.3 до 2.4	-	1
від 2.4 до 2.5	-	1
всього	8	10

Аналіз спостережень і обчислювальних експериментів показує [26–36], що аномальні хвилі часто виникають у складі групи хвиль, що мають форму солітоноподібних утворень. І в цьому випадку такі хвилі з'являються саме в складі груп досить великих хвиль, причому в загальному випадку довжина модуляції менше, ніж за описом у рамках S-теорії (рис. 1.13).

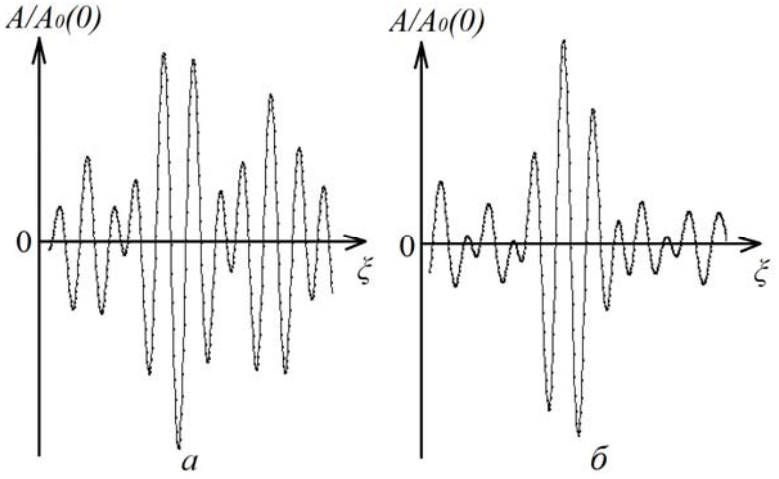

Рис. 1.13. Характерний вид аномальних хвиль у складі груп хвиль у разі опису в рамках S-теорії (а) і в загальному випадку розгляду без наближень (б)

Велика хвиля з амплітудою, що можна порівняти з $2U_{SWH}$, припадає на 10^4 хвиль, що узгоджується зі статистичними оцінками, однак хвилі з амплітудою $2.5U_{SWH}$ і більше з'являються значно частіше, ніж це можна було очікувати за випадкової інтерференції хвильового руху.

ВИСНОВКИ

Особливості застосування технології CUDA. Алгоритм використання CUDA в обчислювальному експерименті відповідає алгоритму використання CUDA під час розв'язання задачі Коші методом Ейлера. Для середовищ із сильною і слабкою дисперсією алгоритм однаковий через наявність одних і тих же рівнянь. Деякі параметри моделей не розраховуються на GPU, тому що їх розрахунок відбувається не кожен крок часу $\Delta\tau$.

Для рівнянь $A_{n,1}, A_{n,2}, B_{n,1}, B_{n,2}, \partial u_n, \partial\varphi_n, u_n, \varphi_n$ створена сітка з одного блоку з кількістю потоків, що дорівнює кількості рівнянь. Окремими рівняннями описані $A_{0,1}, A_{0,2}, B_{0,1}, B_{0,2}, \partial u_0, \partial\varphi_0, u_0, \varphi_0$, тому їх розрахунок відбувається не паралельно.

Під час порівняння часу обчислення рівнянь перевага GPU над CPU зростає із ростом числа мод. Обчислення на CUDA можна прискорити ще в 1.7 разів без зниження точності результатів шляхом обчислення тригонометричних функцій синуса і косинуса в урізаній точності.

Оптимізацію алгоритму шляхом об'єднання сіток потоків в одну сітку доцільно виконати для рівнянь $A_{n,1}, A_{n,2}, B_{n,1}, B_{n,2}$. Тоді швидкість обчислень на CUDA виросте ще в 1,4 рази.

Результати моделювання. Під час порівняння S-теорії і більш загальної теорії (тобто без спрощень S-теорії) модуляційних нестійкостей визначено, що багато характеристик процесу нестійкості виявляються близькі, принаймні на початковій стадії нелінійного режиму нестійкості. Подібними виявляються максимальні амплітуди модуляції (обвідної), окремих хвиль, часи їх появи. Таким чином, S-теорія дозволяє не тільки якісно, але й кількісно описувати початкову стадію нелінійного режиму процесу модуляційної нестійкості.

Результати моделювання виявляють формування цугів хвиль, короткий час життя таких груп хвиль і пояснюють природу розширення масштабу і зменшення амплітуди модуляції хвильового руху. Аномальні хвилі надвисокого розмаху й амплітуди виявлені на початковій стадії модуляційної нестійкості. На великих часах процесу амплітуда аномально великих хвиль виявляється меншою, ніж на початковій стадії розвиненої нестійкості. Визначено, що частота появи аномальних хвиль і їх амплітуди відповідають оцінкам [34; 37; 38].

РОЗДІЛ 2

МОДЕЛЮВАННЯ РУХУ ЗГУСТКУ ЕЛЕКТРОНІВ У ПЛАЗМІ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ CUDA

2.1. АКТУАЛЬНІСТЬ ЗАДАЧІ В ПРЕДМЕТНІЙ ОБЛАСТІ

Актуальні задачі застосування високоенергетичних і потужнострумових коротких електронних пучків – згустків, що рухаються в плазмі для прискорення іонів [39; 40], поставили перед дослідниками дві проблеми: як домогтися максимальної амплітуди кільватерного поля за електронним згустком і яким чином забезпечити стійкість його найбільш ефективної конфігурації, яку, як вважали, потрібно спеціальним чином приготувати. Раніше проблема стійкості розглядалася з позиції врахування власних полів пучка, зовнішньої магнітної конфігурації та індукційної реакції плазмового і конструкційного оточення [41; 42]. Детальніший огляд літератури з питань стійкості пучків у плазмовому середовищі можна знайти в монографіях [43–45].

Запропоновані Я. Б. Файнбергом методи підвищення стійкості пучка як цілого за рахунок збуджених ним полів у плазмі [46] викликали інтерес до поперечного (радіального) фокусування модульованих пучків (на частоті, що менша за плазмову) [47; 48], що було виявлено експериментально [49]. Крім помітного самофокусування [50; 51], інтенсивність випромінювання протяжних пучків не демонструє помітного зниження під час їх транспортування в плазмі [52; 53], на відміну від випадку коротких одиночних згустків. Уявлення про досягнення у сфері прискорюючих полів і можливості прискорення іонів у полі випромінювання таких пучків в різних умовах можна отримати з робіт [54; 55]. Амплітуди збуджених полів у протяжних обмежених пучках можна порівняти з амплітудами в необмежених системах, більшість механізмів взаємодії частинок пучка з плазмою подібні до вивчених раніше [56–59]. Однак облік обмеженості потребував детального вивчення стійкості пучка як цілого, його радіального фокусування (що досить добре обговорюється в монографії [60]) і з’ясування ступеня зарядної і струмової компенсації.

Дещо інші проблеми виникли під час вивчення досить компактних порівняно з довжиною випромінювання згустків заряджених частинок, які рухаються у плазмі. Основну увагу спочатку приділяли профілюванню щільності згустків, домагаючись найбільшого потенціалу поля позаду

нього [40]. Оскільки процес прискорення потребував більшого часу, стійкість і динаміка одиночного згустку у власних полях також виявилася важлива. Питання стійкості коротких пучків заряджених частинок, що рухаються в плазмі, розглядалися в роботах [60; 61], де було виявлено явище їх поперечного і поздовжнього самофокусування. Це явище еквівалентно до обернення знака макроскопічної діелектричної проникності в об'ємі такого згустку [61]. Особливістю такого згустку, що рухається в плазмі, є також практично повна компенсація його заряду [62], якщо його початкові розміри (поздовжні α_1 і поперечні $\alpha_\perp$ помітно перевищували величину V_0/ω_{pe}, де V_0, ω_{pe} – незбурена швидкість пучка і ленгмюрівська частота плазми).

Однак конкуруючим процесом по відношенню до самофокусування є розвиток нестійкості моноенергетичного пучка з подальшою його модуляцією, характерний розмір якої порядку довжини поля випромінювання. За розмірів електронного згустку $a_\parallel$, помітно менших за характерну відстань, на котрій розвивається пучкова нестійкість в безмежному плазмово-пучковому середовищі $(V_0/\omega_{pe})(\omega_{pb}/\omega_{pe})^{-2/3}$ (де $\omega_{b,pe}=(4\pi e^2 n_{p,bo}/m_{bo})^{1/2}$ – плазмові електронні частоти пучка і плазми), поле не встигає накопичитися в розмірах згустку. Ефективний декремент загасання коливань δ_D в згустку можна визначити як відношення потоку енергії коливань, які покидають згусток, до повної енергії коливань в його обсязі, при цьому $\delta_D = V_0/a$. Відношення ефективного декремента загасання до максимального інкремента пучково-плазмової нестійкості $\propto \omega_{pe}(\omega_{be}/\omega_{pe})^{2/3}$ порядку $\Theta=\delta_D/\gamma|_{\delta=0}=(V_0/\omega_{pe})(\omega_{pb}/\omega_{pe})^{-2/3}>>1$ [63]. Пучкова нестійкість, яка розвивається в цих умовах, є дисипативною з інкрементом [64; 65] з точністю до чисельного множника, який дорівнює максимальному інкременту (розрахованому для необмеженої системи «пучок – плазма»), помноженому на $\Theta^{-1/2}$. Енергія поля за час a/V_0 виноситься з об'єму згустку, а зростання амплітуди за цей час пропорційне $\exp\{(a/V_0)\cdot[\omega_{pe}(\omega_{be}/\omega_{pe})^{2/3}/\Theta^{1/2}]\approx 1-(1/\Theta^{1/2})\}$ і досить незначне [66]. Таким чином, зростання поля в об'ємі пучка обумовлено великою мірою групуванням часток (тобто синхронізацією випромінювачів, обумовленою нестійкістю) і підвищенням когерентності їх випромінювання. У разі розвитку нестійкості когерентність поля кільватерного сліду значно збільшується. У разі одновимірних згустків, розмір яких більший або в кілька разів перевищує довжину хвилі випромінювання, з однією і тією ж фіксованою кількістю частинок $(a\omega_{pe}/2\pi V_0)>1$, найбільша амплітуда випромінювання, що досягається в процесі нестійкості, слабо залежить від початкового поздовжнього розміру згустку, що дозволяло вважати її (амплітуду) максимально можливою [66]. Виявилося, що максимальна

досяжна напруженість електричного поля за таким одновимірним згустком в Θ разів менше, ніж у разі протяжного пучка тієї ж щільності [66]. Дійсно, можна показати [63], що в разі накопичення поля випромінювання в об'ємі пучка постійної щільності ефективність його бунчування зростає, зменшується характерний час процесу бунчування (в $\sqrt{\Theta}$ разів менше), істотно зростає напруженість електричного поля випромінювання (в Θ раз більше) порівняно з розглянутим вище випадком зверхвипромінення коротких пучків-згустків.

Розгляд поведінки тривимірного моноенергетичного згустку таких же розмірів показав [67–70], що в їх об'ємі також розвивається нестійкість дисипативного типу, подібна до розглянутої вище. Ступінь досягнутої когерентності випромінювання нижче, ніж в одновимірному випадку, а перемішування захоплених частинок у потенційній ямі випромінювання відбувається більш ефективно, що призводить до швидкого зменшення амплітуди випромінювання. Процеси в поздовжньому напрямку (по руху згустку) відбуваються швидше, тому поперечна модуляція щільності слабкіше виражена, ніж у випадку, обговорюваному в роботі [70], що слід пов'язувати з різним вибором початкової форми згустку.

Компактні згустки, початковий розмір яких багато менший за довжину хвилі випромінювання, в одновимірному і тривимірному випадку нестійкі і швидко розлітаються. Амплітуда випромінювання, яка в початковий момент максимально досяжна для згустку з певною кількістю частинок, монотонно убуває з часом. Взагалі кажучи, за час набагато більший зворотного інкремента нестійкості інтенсивність випромінювання зменшується до рівня сумарного некогерентного спонтанного випромінювання частинок згустку.

Нижче детально обговоримо характер випромінювання і стійкість рухомих у холодній плазмі одновимірних згустків заряджених частинок однієї енергії. Будемо вважати кількість часток у згустках різного розміру фіксованою. Покажемо, що в разі згустків, розмір яких перевищує довжину хвилі випромінювання, максимальна амплітуда кільватерного поля в два рази менше за максимально можливу для цієї кількості частинок. При цьому максимум обвідної випромінювання відстає від згустку, залишаючись у сфері його формування в лабораторній системі відліку.

2.2. РОЗРОБКА МАТЕМАТИЧНОЇ МОДЕЛІ ПРОЦЕСУ. КІЛЬВАТЕРНЕ ПОЛЕ ОКРЕМОЇ ЧАСТИНКИ

Важливо відзначити, що якщо розглядати нескінченну періодичну систему розташування окремих часток, як це часто роблять, а потім необмежено збільшувати величину періоду, то перехід до кільватерного поля окремої частки згустку виявиться важким. Бо в періодичній системі

поле присутнє як попереду, так і позаду окремо взятої частинки. А для одиночної частинки поле випромінювання попереду неї в напрямку її руху в плазмі відсутнє (див., наприклад, [62; 63]). Уявімо щільність заряду електрона, що рухається зі швидкістю $\nu > 0$, в наступному вигляді: $\rho = -e \cdot \delta(-\nu \cdot t + x - s) = -e \cdot \delta(\xi - s)$. Використовуємо рівняння Пуассона $\partial D / \partial x = 4\pi\rho$, яке в Фур'є-поданні запишемо у вигляді

$$-ik\varepsilon(\omega, k) \cdot E(\omega, k) = 4\pi\rho(\omega, k). \qquad (2.1)$$

Виконуючи зворотне перетворення в лівій частині рівняння, отримаємо

$$-i\int_{-\infty}^{\infty} \exp\{-ik\xi\} \cdot dk \cdot [k \cdot \varepsilon(k) \cdot E(k)] = k_0 \cdot \frac{\partial\varepsilon(k)}{\partial k}|_{k_0} \cdot \frac{\partial E}{\partial \xi} \cdot \exp\{-ik_0\xi\}. \qquad (2.2)$$

Тут ми скористалися тією обставиною, що хвильовий пакет розташовується в просторі хвильових векторів поблизу k_0, і тоді $d(k_0 + K) = dK$. Крім того, $E = \int_{-\infty}^{\infty} \exp\{-ik\xi\} \cdot dK \cdot E(K)$ – повільно змінна в просторі амплітуда напруженості електричного поля.

Використовуємо також той факт, що в рухомій системі відліку рівняння $\varepsilon(\omega, k) = \varepsilon(k\nu, k) = \varepsilon(k) = 1 - \omega_{pe}^2 / k\nu(k\nu + i\nu) = 0$ має корені $k_{1,2}\nu = k_0\nu = \pm\omega_{pe} - i\nu/2$.

Рівняння Пуассона при цьому набуває вигляду

$$(\partial E / \partial \xi) \cdot \exp\{-ik_0\xi\} = -4\pi e \cdot \{k_0 \cdot \partial\varepsilon(k)/\partial k|_{k_0}\}^{-1} \cdot \delta(\xi - s). \qquad (2.3)$$

Причому в цьому випадку виконується співвідношення $k_0 \cdot \partial\varepsilon(k)/\partial k|_{k_0} = \partial\omega\varepsilon(\omega)/\partial\omega|_{\omega = k_0\nu}$. Для подальшого перетворення рівняння (2.3) скористаємося поданням [71] $\delta(x) = d\theta(x)/dx$, де $\theta(x)$ – симетрична одинична функція, яка дорівнює нулю за $x > 0$ і дорівнює одиниці за $x > 0$.

З огляду на наявність дельта-функції, рівняння (2.3) можна представити у вигляді

$$\partial E / \partial \xi = \alpha \cdot \delta(\xi - s), \qquad (2.4)$$

де $\alpha = -4\pi e \cdot \{k_0 \cdot \partial\varepsilon(k)/\partial k|_{k_0}\}^{-1} \cdot \exp[ik_0 s]$. Розв'язок шукаємо у вигляді $E = C + \alpha \cdot \theta(\xi - s)$, де C – деяка невизначена константа. Оскільки напруженість поля має вигляд

$$E \cdot \exp\{ik_0 s\} = [C + \alpha \cdot \theta(\xi - s)] \cdot \exp\{-ik_0 \xi\} \ , \qquad (2.5)$$

то в області великих значень $\xi > 0$ вираз тяжіє до нескінченності, що неприпустимо. Тому слід вибрати $C = -\alpha$. Таким чином, остаточно напруженість поля кільватерного сліду за часткою, що рухається в позитивному напрямку [72]:

$$E = -4\pi e \cdot \{k_0 \cdot \partial\varepsilon(k)/\partial k\,|_{k_0}\}^{-1} \cdot \theta(s - \xi) \cdot \exp\{ik_0(s - \xi)\} \, . \qquad (2.6)$$

Рівняння (2.6) отримано для рівномірно рухомого заряду. Більш строгі обчислення [73] дозволяють отримати аналогічну формулу для поля заряду з довільним законом руху $t = t_L(x)$:

$$E = E_0 \theta[t - t_L(x)] \exp[i\omega_p (t - t_L(x))] \, . \qquad (2.7)$$

Тут $t_L(x)$ – так званий лагранжевий час, тобто час прильоту частки в точку x (в лабораторній системі відліку). Вираз (2.7) має прозорий фізичний зміст: частка, пролітаючи через точку x, збуджує поздовжнє поле з амплітудою $E_0 = -4\pi e \cdot \{k_0 \cdot \partial\varepsilon(k)/\partial k\,|_{k_0}\}^{-1}$. Надалі поле в цій точці осцилює з плазмовою частотою і ніяк не залежить від подальшої еволюції частинки. Важливо підкреслити, що фаза поля, що створюється зарядом у точці x, залежить тільки від різниці поточного часу і часу прольоту заряду через цю точку.

У разі рівномірного руху заряду вираження (2.6) і (2.7) еквівалентні, тому що $\omega_p(t - t_L(x)) = k_0(x(t) - x) = k_0(x(t) - x) = k_0(\xi(t) - \xi)$, де $x(t)$ та $\xi(t)$ – поточна координата частинки в лабораторній системі відліку і системі відліку, пов'язаній із часткою, відповідно. Звідси нескладно отримати оцінку застосовності вираження (2.6) для використання в самоузгодженій моделі збудження кільватерного поля згустком з урахуванням нерівномірності руху частинок згустку, зумовленої впливом самоузгодженого поля:

$$\frac{\phi_t - \phi_\chi}{2\pi} = \frac{\phi_\chi}{2\pi}\left[\frac{\nu_0}{\bar{\nu}(t,x)} - 1\right] = \frac{\phi_\chi}{2\pi}\left[1 - \frac{\bar{\nu}(t,x)}{\nu_0}\right] \approx \frac{\phi_t}{2\pi}\frac{\overline{\Delta\nu}}{\bar{\nu}} << 1 \, , \qquad (2.8)$$

де $\phi_t = \omega_p(t - t_L(x))$, $\phi_\chi = k_0(x(t) - x)$, $\bar{\nu}(t,x) = (x(t) - x)/(t - t_L(x))$ та $\overline{\Delta\nu}(x,t)$ – відповідно, середня швидкість і варіація швидкості частинки на ділянці $[x, x(t)]$.

2.3. ФОРМАЛІЗАЦІЯ МАТЕМАТИЧНОЇ МОДЕЛІ ПРОЦЕСУ

Неважко побачити, що кільватерне поле випромінювання окремої частинки являє собою результат її спонтанного випромінювання. Спонтанні поля окремих частинок згустку (за їх однорідного розподілу і за відсутності зовнішніх механізмів синхронізації) відрізняються фазою, яка принаймні в початковий момент випадкова, тобто спонтанне випромінювання однорідно розподілених і несфазованих n випромінювачів – некогерентне. Зміна енергії спонтанного випромінювання в одиницю часу пропорційна до кількості випромінювачів, тобто $\propto n$. На кожну частинку діє її власне поле і поле частинок, які знаходяться попереду неї. Поле спонтанного випромінювання в замкнутому просторі зростає лінійно з часом [63], у відкритій системі – в згустку зростання поля обмежене виносом енергії за межі згустку. Але, починаючи з деякого моменту, групування випромінювачів може виявитися таким, що ініціативу впливу на частинки здатне перехопити випромінювання, породжене згрупованими частинками. Якщо кількість таких згрупованих часток дорівнює s, то інтенсивність поля, породженого цією групою частинок, буде пропорційне s^2. За $s \propto \sqrt{n}$ поле згрупованих часток вже буде домінувати в процесі самомодуляції згустку і формування кільватерного сліду за згустком. Це зростання когерентного, що вже набуває рис індукованого, випромінювання відбувається експоненційно. Зростає ступінь когерентності і в загальному випромінюванні згустку, тобто фази багатьох окремих випромінювачів слабо відрізняються одна від одної. Зміна енергії поля в одиницю часу в цьому випадку пропорційна до квадрата кількості синхронізованих осциляторів. Причому подібна синхронізація відбувається під дією випромінюваної хвилі і керується нею.

Підсумовуючи поля всіх частинок згустку, отримаємо вираз для поля:

$$E(\xi) = -\frac{2}{N}\sum_{\alpha}^{N} f_\alpha \cos[2\pi g_\alpha(\xi - \xi_\alpha)]\Theta(\xi_\alpha - \xi), \qquad (2.9)$$

до якого слід додати рівняння руху для частинок:

$$\frac{d\xi}{d\tau} = \nu, \quad \frac{d\nu}{d\tau} = E(\xi), \qquad (2.10)$$

де $2\pi\xi = K_0(z - V_0 t)$, $\nu = K_0(V - V_0)/2\pi\gamma_L$, $\gamma_L^2 = e^2 K_0 M / m_e$, $g = (1 + \Delta\cdot\nu)^{-1}$, $\Delta = 2\pi\gamma_L / K_0 V_0$, $\tau = \gamma_L t$ та M – загальна кількість частинок у згустку в одиничному перетині, $E = eK_0 E / 2\pi m_e \gamma_L^2$, E – напруженість електричного поля, f_α – статистична вага великої частки, що моделює пучок, e, m_e, $K_0 = \omega_{pe}/V_0$ – заряд, маса електронів і хвильове число кільватерного поля.

Рівняння (2.9), (2.10) описують нелінійну динаміку короткого одновимірного електронного згустку, що поширюється крізь щільну плазму в системі його спокою [62]. Неважко показати, що ця система рівнянь практично повністю еквівалентна системі рівнянь для дисипативної пучкової нестійкості [64; 65] (див. також [66]), де зміною обвідної амплітуди поля з часом можна знехтувати, тобто $\partial E/E\partial t << \delta_D$, порівняно з декрементом поглинання, який у цьому випадку дорівнює $\delta_D = V_0/a$. Таким чином, ця модель, як показано в роботах [62; 63], коректно описує динаміку одновимірного короткого моноенергетичного пучка заряджених частинок, що поширюється в плазмі.

Енергія поля за час a/V_0 виноситься з об'єму згустку, а зростання амплітуди за цей час пропорційно $\exp\{(a/V_0)\}\cdot[\omega_{pe}(\omega_{be}/\omega_{pe})^{2/3}/\Theta^{1/2}] \approx 1-(1/\Theta^{3/2})$ і досить незначне. Таким чином, зростання поля в об'ємі пучка обумовлено більшою мірою групуванням частинок і підвищенням когерентності їх випромінювання [66].

2.4. УМОВИ ЗАСТОСОВНОСТІ ОПИСУ

Як було зазначено вище, рівняння для поля (2.6) і рівняння (2.9), що витікає з нього, мають певні обмеження. Застосовність цих рівнянь обмежена просторовою (або часовою) областю, в якій варіація швидкості дуже мала порівняно з початковою швидкістю згустку (в більшості випадків застосування її можна вважати близькою до швидкості світла). Уточнимо межі застосування моделі (2.9)–(2.10) в прийнятих нами безрозмірних одиницях. Будемо виходити з більш суворої моделі [73], що використовує лагранжевий час.

Рівняння для лагранжевого часу має в наведених раніше позначеннях вид

$$\frac{d\tau_{L\alpha}(\xi)}{d\xi} = \frac{\Delta}{1+\Delta\cdot\nu_{L\alpha}(\xi)}, \tag{2.11}$$

до якого слід додати рівняння руху:

$$\left(\frac{1+\Delta\cdot\nu_{L\alpha}}{\Delta}\right)\frac{d\nu_{L\alpha}(\xi)}{d\xi} = E(\xi,\tau) \tag{2.12}$$

і рівняння для поля:

$$E(\xi,\tau) = -\frac{2}{N}\sum_{\alpha}^{N} f_\alpha \cos[\widetilde{w}_p(\tau-\tau_{L\alpha}(\xi))]\Theta(\tau-\tau_{L\alpha}(\xi)). \tag{2.13}$$

Тут $\tau_{L\alpha}(\xi)$ – час, коли частинка з номером α проходить через точку ξ, $\nu_{L\alpha}(\xi)$ – її швидкість в цей момент. Інтегруючи рівняння (2.11) в інтервалі $[\xi,\xi_\alpha]$, отримаємо

$$\tau_{L\alpha}(\xi)=\tau-\Delta\cdot\int_{\xi}^{\xi_\alpha}\frac{d\xi}{1+\Delta\cdot\nu_{L\alpha}(\xi)}. \tag{2.14}$$

Щоб замкнути це рівняння, нам треба знати $\nu_{L\alpha}(\xi)$. З огляду на те, що в лабораторній системі відліку частинка дуже швидко долає відстань $[\xi,\xi_\alpha]$ (зі швидкістю близько Δ^{-1}), можна припустити, що її швидкість на цьому інтервалі змінюється слабо і тому величину $\nu_{L\alpha}(\xi)$ можна розкласти в ряд по $\xi_\alpha-\xi$ щодо поточного стану частинки:

$$\nu_{L\alpha}(\xi)\approx\nu_\alpha+\frac{d\nu_\alpha}{d\xi}(\xi-\xi_\alpha)+\ldots=\nu_\alpha+\frac{\Delta}{1+\Delta\cdot\nu_\alpha}\frac{d\nu_\alpha}{d\tau}(\xi-\xi_\alpha)+\ldots\ . \tag{2.15}$$

Підставляючи це розкладання в (2.14), отримаємо

$$\tau_{L\alpha}(\xi)=\tau-\Delta\int_{\xi}^{\xi_\alpha}\frac{d\xi}{1+\Delta\cdot\nu_{L\alpha}(\xi)}\approx\tau-\Delta(1-\Delta\cdot\nu_\alpha)(\xi-\xi_\alpha)+\frac{1}{2}\Delta^3(1-\Delta\cdot\nu_\alpha)\frac{d\nu_\alpha}{d\tau}(\xi-\xi_\alpha)^2+\ldots. \tag{2.16}$$

Нескладно показати, що якщо в отриманому виразі утримати тільки лінійний по $\xi_\alpha-\xi$ доданок і підставити його в (2.13), то ми отримаємо рівняння (2.9). Різниця виникає під час обліку доданків більш високого порядку. Звідси випливає умова застосовності рівняння (2.9). Виходячи з вимоги малості квадратичного доданка

$$\frac{1}{2}\Delta^3(1-\Delta\cdot\nu_\alpha)\frac{d\nu_\alpha}{d\tau}(\xi-\xi_\alpha)^2<<1, \tag{2.17}$$

отримаємо кордон просторової області застосовності цієї моделі:

$$\xi>>\xi^*=-\frac{2}{\Delta^{3/2}|E|}\,. \tag{2.18}$$

Оскільки $|E|\le 2$, а для релятивістських згустків $\Delta<<1$, то можна вважати, що запропонована нами спрощена модель може бути застосована в досить широкій просторовій області і може бути використана в практичних розрахунках. Виходячи з (2.18), межі застосування моделі можна визначити так: якщо навіть $|E|\approx 2$, то для $\Delta=0$: $\xi^*\to-\infty$ –

еквівалентно нескінченній швидкості пучка, тому що швидкість пучка в обраних одиницях пропорційна до Δ^{-1}, для $\Delta = 0.1$: $\xi^* \approx -31$ – найпроблемніший випадок, для $\Delta = 0.01$: $\xi^* \approx -1000$.

2.5. ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ CUDA ДЛЯ МОДЕЛЮВАННЯ ПРОЦЕСУ

Програма, що реалізує математичну модель задачі, створена з використанням технології JCUDA. JCUDA забезпечувала виконання з Java-програми програмного коду на GPU.

Розглянемо рух згустку з N частинок протягом деякого періоду часу (рис. 2.1), який розбитий на часові відрізки з кроком dt.

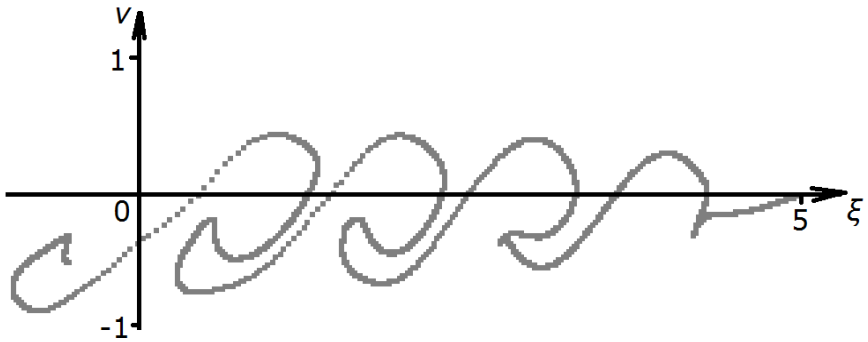

Рис. 2.1. Згусток з 10240 частинок у момент часу t = 4

Для кожної частинки на GPU розраховуються її координати ξ і v на кожному кроці у вигляді вирішення задачі Коші методом Ейлера:

$$\frac{dv_\beta}{dt} = -\frac{2}{N} \sum_{\alpha \mid \xi_\alpha > \xi_\beta}^{N} \cos[2\pi \cdot (\xi_\beta - \xi_\alpha)], \tag{2.19}$$

$$\frac{d\xi_\beta}{d\tau} = v_\beta. \tag{2.20}$$

В обчислювальному експерименті використовується схема взаємодії CPU частини і GPU частини програми з використанням двох циклів «for» під час розрахунку рівнянь для часових кроків. У рамках вкладеного циклу обчислення відбуваються без отримання результатів із пам’яті GPU. Для підвищення точності обчислень необхідно зменшити крок за часом dt, але за малого кроку положення згустку частинок змінюється повільно і тому кожну зміну положення згустку зберігати не потрібно. В цьому випадку тільки результат одного з десяти часових кроків зберігається, тобто переноситься з пам’яті GPU в оперативну пам’ять (ОЗП). Далі обчислення тривають за тією ж схемою, якщо користувач їх не зупиняє. Алгоритм обчислень на GPU показаний на рис. 2.2.

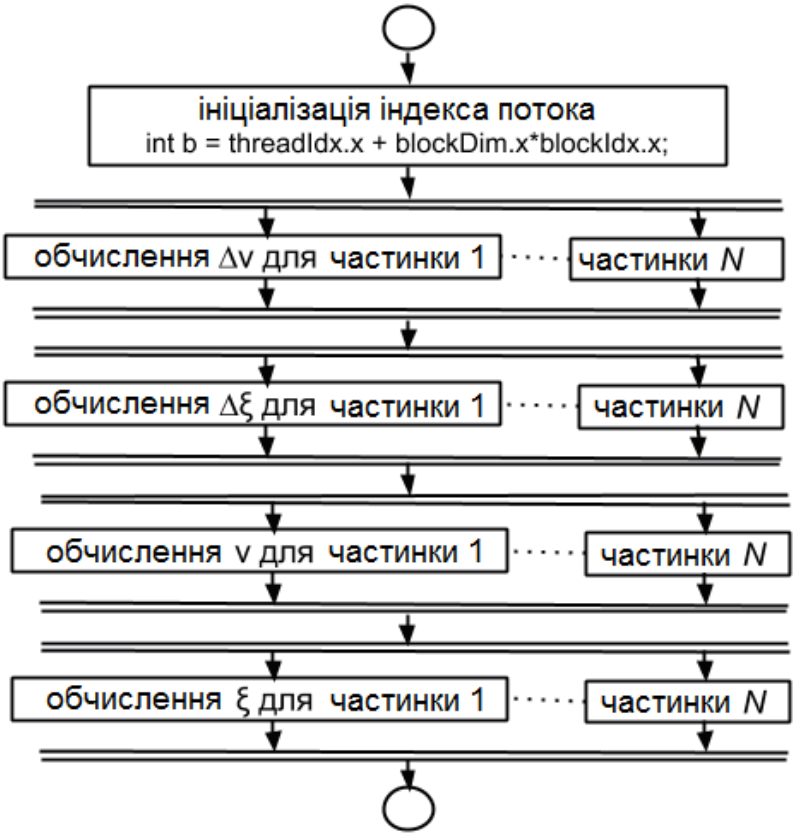

Рис. 2.2. Алгоритм обчислень на GPU

У головній частині програми («host») створюються масиви розміром N для зберігання координат ξ і v. Задаються початкові значення координат за $L = 5$:

```
for(int β=0;β<N;β++){
        ξ[β]=L*(β+1)/(double)N;
        v[β]=0;
}
```

На GPU також створюються ці масиви виділенням пам’яті для них. Масиви «ξ» і «v» з ініціалізованими значеннями в початковий момент часу копіюються з ОЗП в пам’ять GPU перед початком обчислень на GPU.

У головній частині програми задаються параметри розпаралелювання – кількість блоків і кількість потоків у блоці. Наприклад, за N = 10240 і кількості потоків у блоці nThreads = 32 вийде кількість блоків nBlocks = 10240/32 = 320. Код обчислень на GPU:

```
__device__ double complexFunction(int b, int N, double dt, double pi, double *ksi){
        double sum=0;
        for(int a=0;a<N;a++){
```

```
                if(ksi[a]>=ksi[b]){
                        sum+=cos(2.0*pi*(ksi[b]-ksi[a]));
                }
        }
        return -2.0*sum/(double)N*dt;
}
__global__ void func(int N, double dt, double pi, double *ksi, double *v, double *dksi, double *dv){
        int b = threadIdx.x + blockDim.x*blockIdx.x;
        dv[b] = complexFunction(b,N,dt,pi,ksi);
        dksi[b] = v[b]*dt;
        v[b] = v[b] + dv[b];
        ksi[b]=ksi[b]+ dksi[b];
}
```

Обчислення проводилися на відеокарті GeForce GT 610 і на одному ядрі двоядерного процесора AMD Athlon64 X2 2.2Ггц. Спочатку було визначено розмір блоку, за якого швидкість обчислень максимальна (табл. 2.1).

Таблиця 2.1

Час розрахунку на GPU залежно від розміру блоку

Розмір блоку	Час розрахунку на GPU одного часового кроку для 10240 часток, с
8	0.412
16	0.209
32	0.105
40	0.155
64	0.098
80	0.116
128	0.097
160	0.097
256	0.097
512	0.098
1024	0.098

З таблиці видно, що за декількох розмірів блоку швидкість обчислень максимальна; для подальшого використання був обраний розмір блоку 128.

Під час розрахунку на CUDA всі обчислення виконуються на GPU. На CPU виконується розрахунок координат частинок для початкового моменту часу і обробка отриманих на GPU результатів обчислень –

малювання графіка руху частинок після отримання результатів із пам'яті GPU. Проте основний час витрачається саме на обчислення на GPU. Наприклад, за 10240 частинок час обчислення на GPU становить 90 % від часу, витраченого на розрахунки, а за 20480 частинок – 98 %. Частка часу обчислення на GPU наближається до 100 % за подальшого зростання кількості частинок, а час, витрачений на отримання результатів з GPU і малювання графіка, не збільшується. Це показує, що в створеному алгоритмі розрахунку на CUDA немає резервів для підвищення швидкості виконання програми шляхом перенесення обчислень на GPU.

Для перевірки результатів проводяться також розрахунки на CPU. Відбувається порівняння результатів і визначення відхилень. Обчислення на GPU виконуються в подвійній точності. Щоб прискорити обчислення на GPU, можна проводити їх в одинарній точності. Можна також частину обчислень проводити в подвійній точності, а частину в одинарній, наприклад робити частину використовуваних функцій функціями зниженої точності.

Для того щоб програма проводила обчислення в одинарній точності, необхідно змінити тип даних простих змінних і масивів з double на float. Оскільки використовується функція cos( ), вона замінюється на менш точну функцію cosf( ) або __cosf( ).

За 10240 частинок і подвійній точності швидкість обчислення одного кроку за часом складає 0,097 с. Якщо використовувати функцію cosf( ) замість cos( ), а решту розрахунків залишити в подвійній точності, то час обчислення становить 0,031 с. Якщо всі розрахунки проводити в одинарній точності, то час обчислення складе 0,025 с. Швидкість виконання програми, в першу чергу, залежить від кількості частинок. На основі отриманих результатів розрахунків для різної кількості частинок будується графік часу виконання програми на GPU і CPU (рис. 2.3). Обчислення на GPU виконуються в подвійній точності.

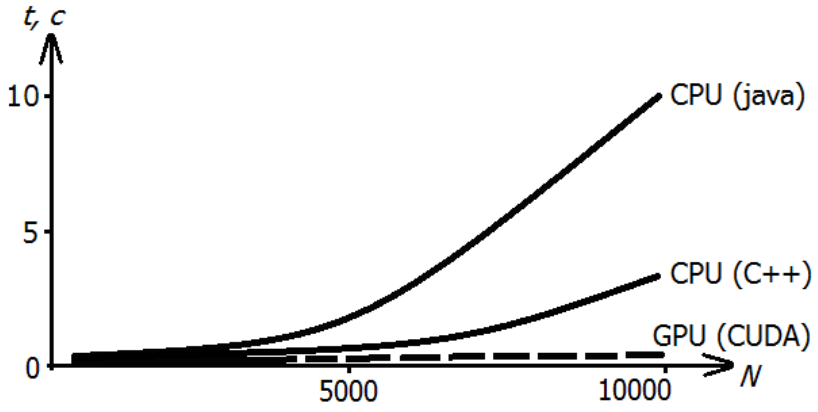

Рис. 2.3. Час розрахунку рівнянь для одного кроку за часом на GPU і CPU залежно від кількості частинок N

2.6. РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ: ВИПРОМІНЮВАННЯ ЗГУСТКУ МАЛОЇ ЩІЛЬНОСТІ ЗА РАХУНОК РОЗВИТКУ ДИСИПАТИВНОЇ НЕСТІЙКОСТІ

Розглянемо в цьому розділі згусток частинок малої щільності такий, щоб зміни швидкості виявилися незначними і не приводили до помітних змін довжини випромінюваного поля ($\Delta = 0$). Короткий пучок-згусток із 1000 частинок, рівномірно розподілених на довжині $\xi \subset (0, L)$ з нульовими початковими швидкостями. Відзначимо, що якщо між кожними двома такими частинками (назвемо їх опорними) рівномірно розташувати 9 або 99 додатково, то, як показано на рис. 2.4, на розглянутих часових відрізках динаміка основних (опорних) частинок не змінюється.

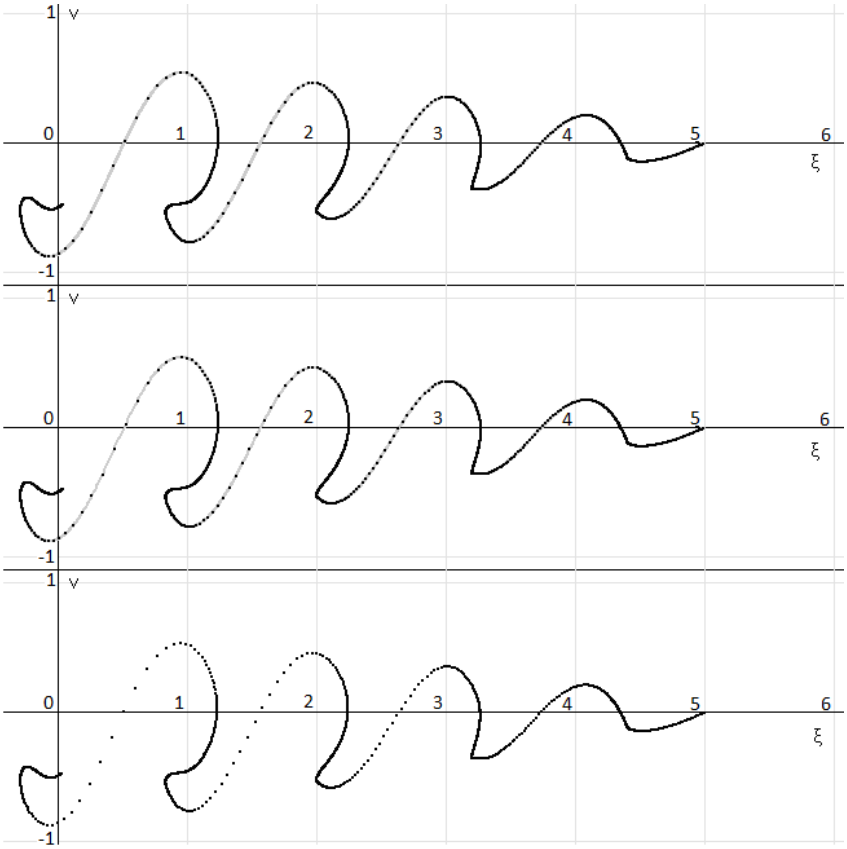

Рис. 2.4. Фазова площина частинок короткого пучка з кількістю частинок $N = 10^5$ *(верхній графік),* $N = 10^4$ *(середній графік),* $N = 10^3$ *(нижній графік) за* $\tau = 3$. *Опорні частки представлені чорним кольором, інші – сірим*

Тому в подальшому ми будемо розглядати згустки, в яких кількість моделюючих частинок (квазічастинок) дорівнюватиме $N = 10^3$. Пучки, розмір яких перевищує в кілька разів довжину випромінюваної хвилі, нестійкі. Через великий рівень випромінювання з об'єму пучка нестійкість набуває яскраво вираженого дисипативного характеру. Амплітуда поля

швидко зростає і обмеження цього зростання обумовлене ефектом захоплення частинок до потенційної ями хвилі (рис. 2.5). При цьому частота осциляцій захоплених частинок $\Omega_{TR} = \sqrt{eK_0 E/m_e}$ виявляється порядку величини лінійного інкремента нестійкості $\gamma_L = \sqrt{e^2 K_0 M/m_e}$ [59].

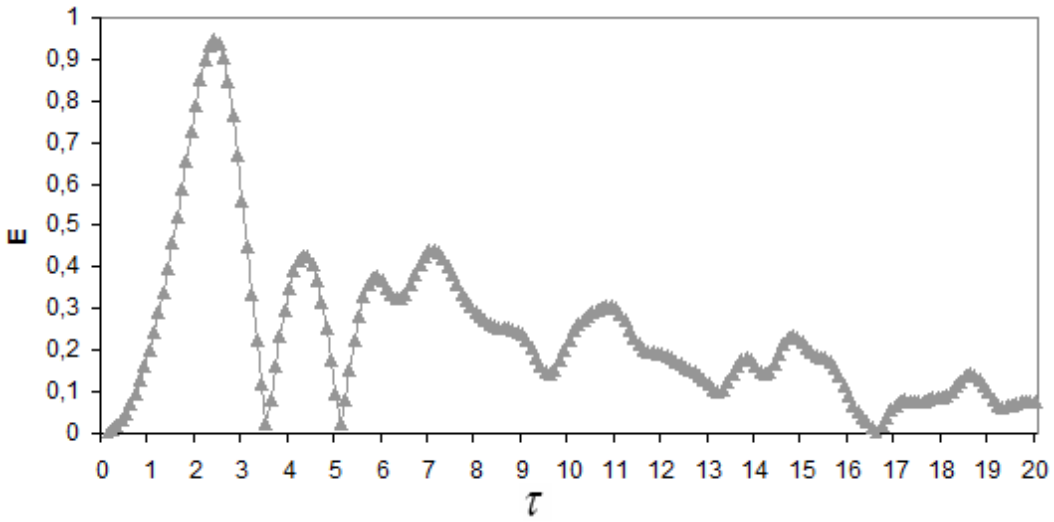

Рис. 2.5. Максимальна амплітуда поля E за згустком L = 5, $\Delta = 0$

На рис. 2.6, 2.7 представлено положення частинок короткого пучка-згустку з 1000 частинок на фазовій площині $(v\xi)$, поле і щільність згустку для двох моментів часу $\tau = 2.39$ та $\tau = 5.12$, коли досягається максимум і мінімум амплітуди поля позаду згустку.

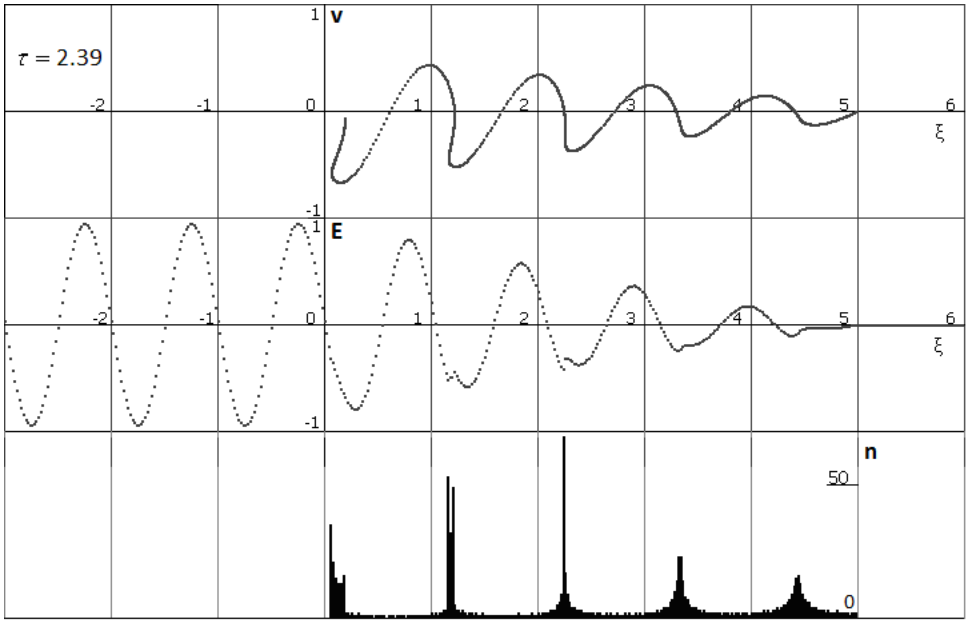

Рис. 2.6. Фазова площина $(v\xi)$, поле $E(\xi)$ і щільність $n(\xi)$ за досягнення максимуму поля поблизу згустку за $\tau = 2.39$, $L = 5$, $\Delta = 0$

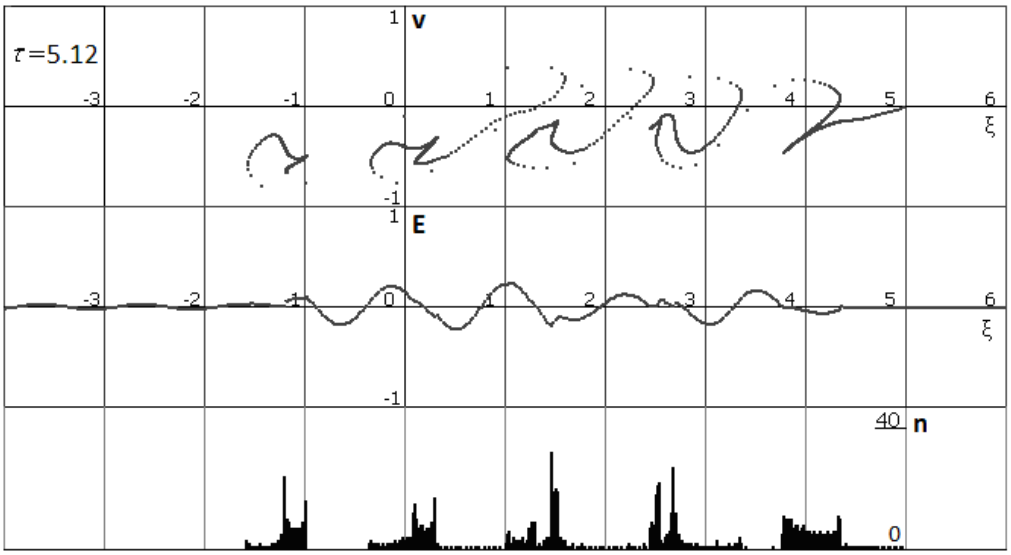

Рис. 2.7. Фазова площина $(v\xi)$*, поле* $E(\xi)$ *і щільність* $n(\xi)$
за досягнення максимуму поля поблизу згустку за $\tau = 5.12$*,* $L = 5$*,* $\Delta = 0$

Видно, що на довжині такого короткого пучка-згустку виникає нестійкість із формуванням його тонкої структури. Ця нестійкість є дисипативною пучковою нестійкістю, яка призводить до просторової модуляції пучка і формування п'яти досить компактних бунчів. В результаті нестійкості максимальна амплітуда поля досягає значень порядку одиниці в цьому нормуванні. Відзначимо, що випромінювання компактної квазічастинки, що представляє собою зібрані в одну точку всі частинки згустку, генерувало би кільватерний слід із максимально можливою амплітудою поля, яка дорівнює $E = E_{LIM} = 2$ в цьому нормуванні, що з очевидністю випливає з виразу (2.7).

2.7. РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ: ВИПРОМІНЮВАННЯ ЗГУСТКУ ВЕЛИКОЇ ЩІЛЬНОСТІ ЗА РАХУНОК РОЗВИТКУ ДИСИПАТИВНОЇ НЕСТІЙКОСТІ

У разі збільшення щільності частинок короткого пучка-згустку, кожна моделююча частка є компактною квазічастинкою з масою Nm_e і зарядом, який дорівнює Ne. При цьому замість $\gamma_L^2 = e^2 K_0 M/m_e$ слід записати $\gamma_L^2 = N^2 e^2 K_0 (M/N)/Nm_e$. Іншими словами, система рівнянь залишається незмінною. Однак тепер через значно збільшену кількість частинок M щільного згустку відношення лінійного інкремента до частоти може бути не настільки мале і $\Delta \neq 0$. Обговоримо випадок досить щільних пучків $\Delta = 0.1$, розміри яких помітно перевершують довжину випромінювання, яка в цьому описі прийнята за одиницю. Особливість динаміки такого короткого пучка обумовлена зміною довжини хвилі випромінювання через істотно більше гальмування і прискорення

частинок. У режимі розвиненої нестійкості максимально досяжна амплітуда кільватерного поля з часом представлена на рис. 2.8.

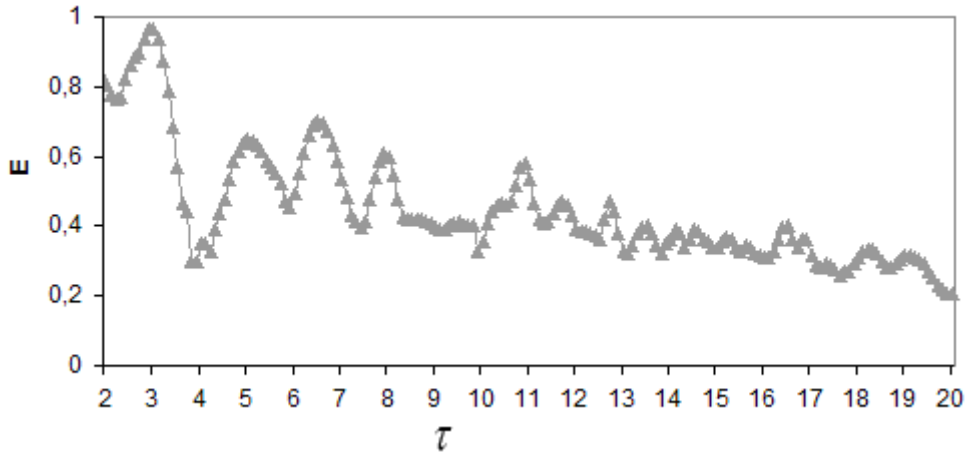

Рис. 2.8. Максимальна амплітуда поля E *за згустком* $L = 5$, $\Delta = 0.1$ *в режимі розвиненої нестійкості*

Однак найбільший максимум обвідної поля при цьому швидко відстає від згустку. Залежність відстані L_{MOD} від згустку до області цього максимуму поля показана на рис. 2.9. Можна показати, що в лабораторній системі відліку виділений максимум поля практично не переміщується (окремі викиди на графіках рис. 2.9 обумовлені неточністю методики визначення максимуму через наявність безлічі локальних екстремумів), що свідчить про «заморожування» поля після прольоту згустку. Оскільки групова швидкість ленгмюрівських коливань холодної плазми дорівнює нулю, подібні результати свідчать про якісне узгодження цієї моделі з відомими фізичними уявленнями.

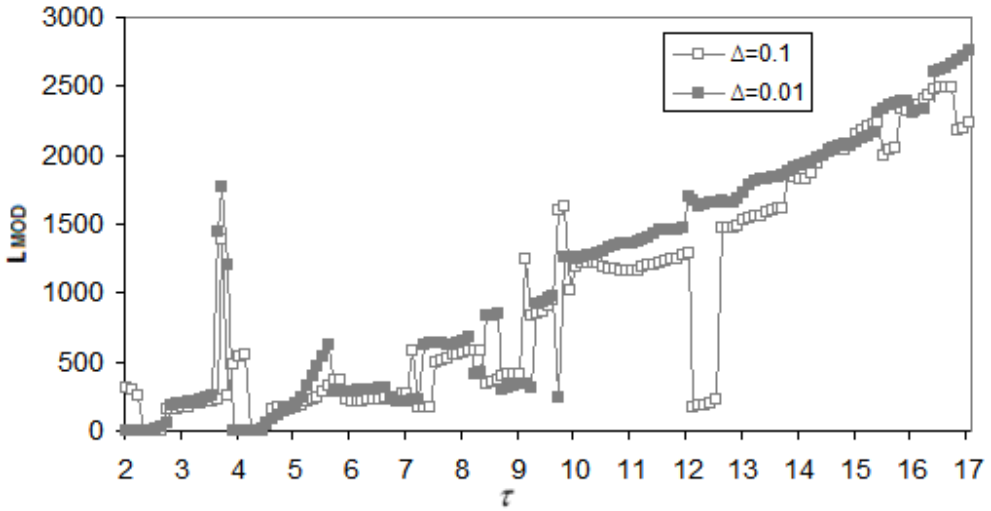

Рис. 2.9. Зміна положення L_{MOD} *максимуму кільватерного поля відносно до протяжних* $(L = 5)$ *згустків заряджених частинок різної щільності* $(\Delta = 0.01$ *та* $\Delta = 0.1)$ *від часу у всій області розгляду. Для* $\Delta = 0.1$ *значення помножені на 10*

На рис. 2.10, 2.11 представлено положення частинок короткого пучка-згустку з 1000 квазічасток на фазовій площині (ν,ξ), поле і щільність пучка для двох моментів часу $\tau = 2.85$ та $\tau = 3.91$, коли досягається максимум і мінімум амплітуди поля безпосередньо позаду щільного згустку. Виходячи з (2.18), межі застосування моделі можна визначити так: $|E|<1$, для $\Delta = 0.1$: $\xi^* \approx -60$, для $\Delta = 0.01$: $\zeta^* \approx -2000$. Оскільки з ростом τ амплітуда поля падає, то графік для $\Delta = 0.01$ практично на всьому інтервалі часу попадає в межі застосування, а для $\Delta = 0.1$ враховуємо, що графік розтягнутий в 10 разів і кордон застосовності можна рахувати для амплітуди поля $|E| \approx 0.5$. Тож $\zeta^* \approx -1200$, тобто в цілому модель працює до $\tau \approx 10$.

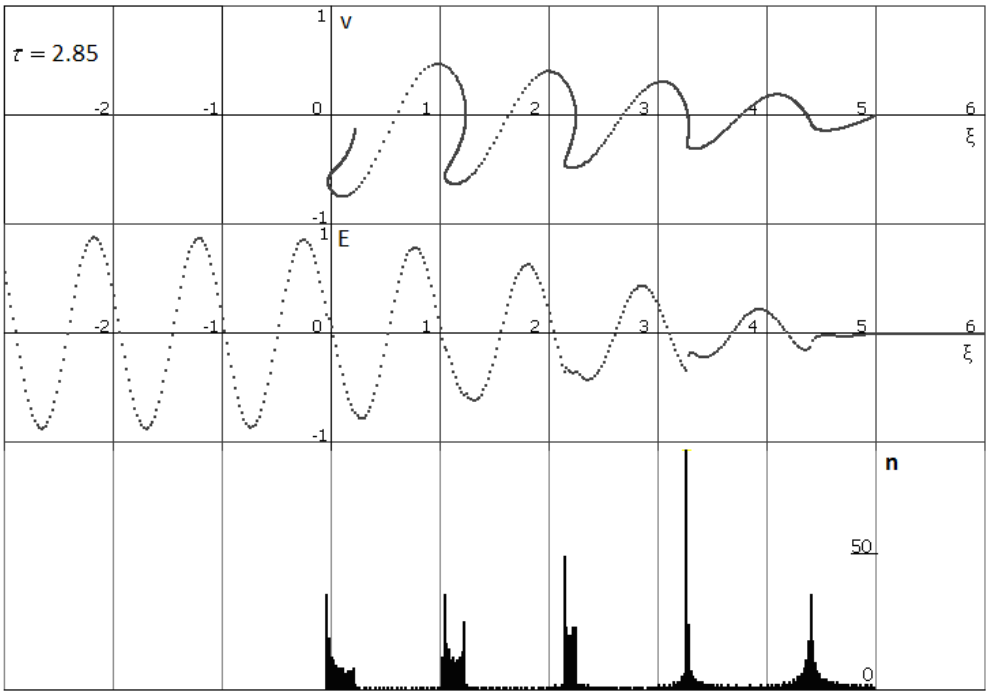

Рис. 2.10. Фазова площина (ν,ξ), поле $E(\xi)$ і щільність $n(\xi)$ за досягнення максимуму поля поблизу згустку за $\tau = 2.85$, $L = 5$, $\Delta = 0.1$

Є сенс розглянути характер фазової синхронізації частинок пучка, для чого можна для кожної частинки обчислити значення $\phi_\alpha = g_\alpha(\xi_\alpha - \xi)$, при цьому видаляючи цілу частину цієї величини. Множення на g_α враховує, що поле кожної частки має різну довжину хвилі випромінювання. Точку спостереження ξ виберемо в околиці максимуму поля за згустком. Результати обчислення $\phi_\alpha = g_\alpha(\xi_\alpha - \xi)$ повинні потрапляти в інтервал $(0-1)$. При цьому можна знайти розподіл часток по цих фазах $n(\phi)$ (рис. 2.12, 2.13). Крім того, з огляду на те, що в результаті інтерференції поля випромінюючих частинок з ϕ та $\phi + \pi$ взаємно

пригнічуються, то має сенс визначити величину $n(\phi)-n(\phi+\pi)$, зрозуміло, якщо величина $n(\phi+\pi)$ не дорівнює нулю.

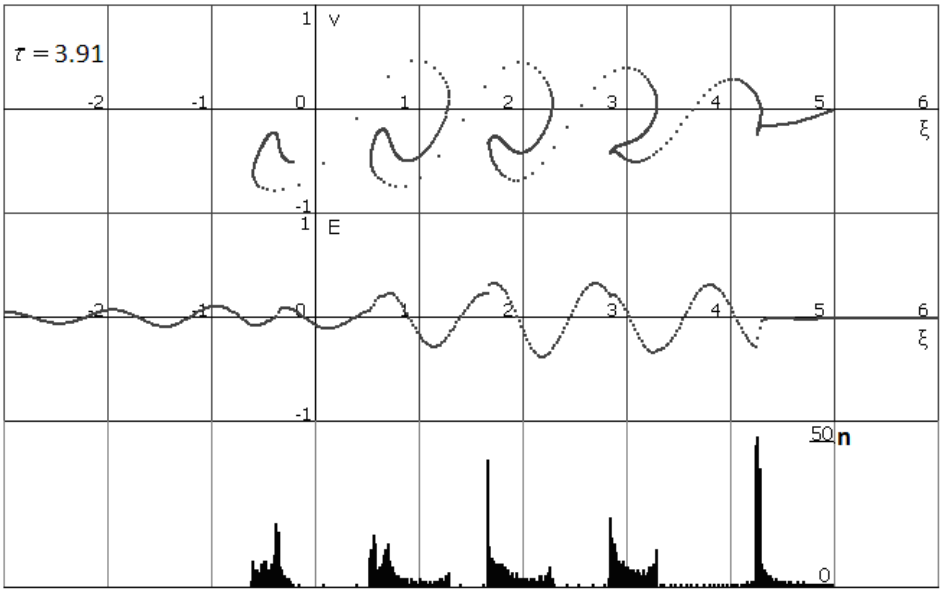

Рис. 2.11. Фазова площина (v,ξ), поле $E(\xi)$ і щільність $n(\xi)$ за досягнення максимуму поля поблизу згустку за $\tau = 3.91$, $L = 5$, $\Delta = 0.1$

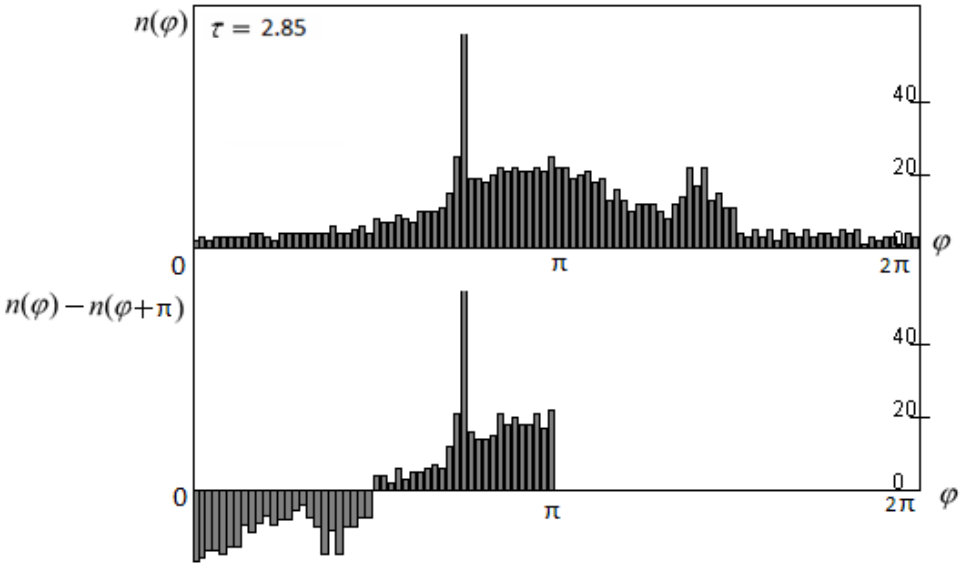

Рис. 2.12. Розподіл часток по фазах випромінюючих полів поблизу максимуму поля для всіх 1000 частинок на верхньому графіку. На нижньому графіку за тих же умов $\sum_{\alpha}\{n(\phi_\alpha)-n(\phi_\alpha+\pi)\} = 642$, $L = 5$, $\Delta = 0.1$

Спектр випромінювання можна визначити, розглядаючи розподіл часток за величиною хвильового вектора поля $g = (1+\Delta\cdot\nu)^{-1}$, що випромінюється ними.

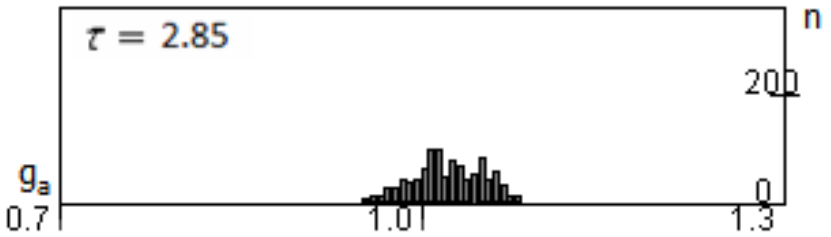

Рис. 2.13. Розподіл часток згустку по хвильових числах випромінюючих полів g поблизу максимуму поля за $L = 5$, $\Delta = 0.1$

Легко побачити, що в процесі нестійкості відбувається інтенсивна модуляція щільності частинок пучка, збільшується розкид частинок за швидкостями, обумовлений як квазіосциляторним рухом у потенційних ямах поля, так і загальним гальмуванням пучка як цілого під час випромінювання. В результаті нестійкості максимальна амплітуда поля досягає значень порядку одиниці в цьому нормуванні, так само як і в разі згустку малої щільності.

ВИСНОВКИ

Розглянуто 1D модель згустку заряджених частинок, що рухається в плазмі. У початковий момент всі частинки рівномірно розподілені на довжині L і мають одну і ту ж швидкість. Обговорюється випадок протяжного згустку, довжина якого в кілька разів перевершує довжину хвилі кільватерного поля.

Особливості застосування технології CUDA. У цій моделі використовується досить велика кількість частинок, що еквівалентно використанню подвоєного числа рівнянь руху.

Для кожної частки на GPU за допомогою CUDA розраховувалися її координати ξ і v на кожному кроці часу у вигляді розв'язання задачі Коші методом Ейлера.

Було визначено розмір блоку, який дорівнює 128 потоків, за якого швидкість обчислень максимальна. Частка часу обчислень на GPU наближається до 100 % за зростання кількості частинок, а час, витрачений на отримання результатів з GPU і їх обробку, не збільшується, що свідчить про відсутність в алгоритмі резервів для підвищення швидкості обчислень шляхом перенесення обчислень на GPU.

Для перевірки результатів проводилися також розрахунки на CPU. Обчислення можна прискорити без втрати точності, використовуючи функцію косинуса й інші змінні в меншій точності, ніж double.

Показано, що швидкість виконання програми, в першу чергу, залежить від кількості частинок. За зростання кількості частинок обчислення на CPU сповільнюються щодо обчислень на GPU.

Результати застосування моделі. Показано, що в об’ємі згустку виникає нестійкість і амплітуда поля в певній галузі, яка відстає від пучка, досягає значень, вдвічі менших за максимально можливу амплітуду випромінювання дуже компактного згустку з тією ж кількістю частинок. Область максимуму поля формується за згустком і в лабораторній системі відліку залишається в місці її утворення. Розмір цієї області в разі досить протяжних згустків займає кілька довжин хвиль, а в разі компактних згустків може виявитися більше.

Запропонована в роботі модель (2.9)–(2.10) має важливу перевагу порівняно з більш строгою моделлю (2.11)–(2.13) з точки зору обчислювальних ресурсів, тому що містить тільки рівняння, що залежать від часу, а модель (2.11)–(2.13) являє собою більш складну просторово-часову задачу. Є й інші особливості моделі (2.9)–(2.10), які роблять її застосування кращим під час вивчення динаміки пучків із відносно невеликою щільністю, розгляд яких дозволяє використовувати лінійний опис збурень навколишньої плазми. Перш за все, можна не враховувати магнітне поле, створене струмом пучка, бо струми в системі спокою пучка-згустку знехтувано малі. Більш того, всі обурення в системі можна вважати потенційними, ігноруючи електромагнітні ефекти, що неприпустимо під час опису динаміки рухомих згустків заряджених частинок у лабораторній системі відліку. Ця обставина набуває ще більшого значення під час переходу до тривимірного моделювання.

РОЗДІЛ 3

МОДЕЛЮВАННЯ СТРУКТУРНО-ФАЗОВИХ ПЕРЕХОДІВ У ТОНКИХ КОНВЕКТИВНО НЕСТІЙКИХ ШАРАХ РІДИНИ ТА ГАЗУ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ CUDA

3.1. АКТУАЛЬНІСТЬ ПРОБЛЕМИ В ПРЕДМЕТНІЙ ОБЛАСТІ

Питання структурних перетворень, структурно-фазових переходів другого роду, в результаті яких змінюється симетрія і частково характерні масштаби просторових структур, завжди цікавили дослідників і творців технологій. Під час розгляду різних процесів у суцільних середовищах необхідно враховувати динаміку спектрів збурень не тільки різного просторово-часового масштабу, а й різної просторової орієнтації [74–84]. Останнє відповідає в звичайному геометричному сенсі формуванню симетрій просторових структур, що мають не тільки ближній, а й далекий порядок [85–86]. Нелінійність середовища проявляється в наявності певних механізмів взаємодії таких збурень. Різні підходи до опису таких взаємодій у нерівноважних середовищах представлені, наприклад, в роботах [87–94]. Процеси за участю великої кількості збурень різного масштабу або орієнтації часто називають багатомодовими, або багатохвильовими, якщо мова йде про хвильові середовища або періодичні системи.

На сьогодні найбільший інтерес представляє з'ясування природи появи просторових структур, виділення фізично прозорих механізмів розвитку таких процесів і потім створення адекватних (що мають прозоре фізичне походження) математичних моделей опису цих явищ. Розглядаючи поведінку таких багатомодових або багатохвильових процесів формування просторових структур, можна побачити появу їх специфічних особливостей, таких як зміна динаміки нестійкості (зокрема, затримки або навіть її придушення [95]) під час формування нестійких нелінійних утворень. Відіграє значну роль і кількість ступенів свободи, кількість мод спектра (що призводить до появи малого параметра $\eta \propto 1/N$, обернено пропорційного кількості мод спектра N). Існування щільного спектра збурень здатне формувати довгоживучі квазістійкі нелінійні стани, здатне затримати розвиток перехідних процесів і процесів структурно-фазових переходів між цими станами [96; 97].

Для моделювання процесів структуроутворення в середовищах, де є виділений характерний розмір взаємодії між елементами майбутньої структури, виявилася досить привабливою модель Проктора–Сівашинського [98; 99], яка була використана для опису процесу розвитку конвекції в тонкому шарі рідини з межами, що погано проводять тепло. Автори роботи [100] виявили стаціонарні рішення з малою кількістю просторових мод, одне з яких (конвективні осередки) виявилося стійким, а друге (конвективні вали) – нестабільним. Особливість моделі в тому, що вона виділяє один просторовий масштаб взаємодії, залишаючи для еволюції системи можливість вибрати характер симетрії. Виявилося, що наявність мінімумів потенціалу взаємодії мод, абсолютна величина векторів хвильових чисел яких незмінна, і визначає вибір симетрії та характеристик просторової структури.

Перспективність моделі Проктора–Сівашинського проявилася у можливості під час обліку полоідальних вихорів всередині тонкого шару сформувати узагальнену модель Проктора–Сівашинського Письмена [101], яка, як показали подальші дослідження, була здатна коректно описати процес трансформації енергії періодичної структури тороїдальних вихорів Проктора–Сівашинського в енергію великомасштабного полоідального вихрового руху [102; 103]. Це явище «гідродинамічного динамо» відповідає за формування великих вихорів в конвективних шарах, зокрема в атмосфері планет. Однак навіть у моделі Проктора–Сівашинського не всі процеси і явища були детально вивчені.

Дослідження характеру розвиненого режиму нестійкості квазістаціонарної структури (валів) було розглянуто пізніше. Вивчення моделі [96] з використанням багатомодового опису дозволило з'ясувати, що спочатку виникає квазістійкий довгоживучий стан (спотворені квазіодномірні вали), який за досить великий час, значно більший за зворотний лінійний інкремент процесу, переходить у стійкий стан (квадратні осередки). Коректний аналіз математичної моделі Проктора–Сівашинського, представлений нижче в цій роботі, дозволив переконатися, що структурний перехід має властивості структурно-фазового переходу (значення функції стану дорівнюють сумі квадратів амплітуд мод спектра певної величини $I = \sum_j a_j^2 \equiv \sum_{k_j} |a_{k_j}|^2$ і відповідають певній топології просторової структури; час переходу між станами обумовлений різницею значень функції стану). Важливою проблемою, що обговорюється в цій роботі, є також визначення рівня дефектності виникаючих регулярних структур і з'ясування можливості знайти кореляцію між інтегральними спектральними характеристиками і часткою дефектних осередків структури, що виникає. Дефектність структур проявляється, зокрема, в проміжних, перехідних режимах процесів і обумовлена

вимушеною (нав'язаною нерівноважністю) інтерференцією зростаючих мод спектра [104]. У разі зовнішнього впливу шум в змозі підтримувати існування безлічі колись депресивних просторових мод, інтерференція яких із домінуючими модами спектра також здатна забезпечити інтерференційну картину, що відповідає дефектній просторовій решітці. Розуміння процесів, які призводять до порушень просторової періодичності структур, дозволило б за їхніми спектрами, експериментальне визначення яких цілком може бути доступним, судити про рівні дефектності просторових структур. Особливо слід з'ясувати вплив зовнішнього шуму на стійкість станів і структурно-фазові переходи.

Саме наявність виділеного масштабу (відстані між регулярними просторовими збуреннями) і можливість підбору потрібної стійкої симетрії (регулярної просторової конфігурації) і визначила інтерес до цієї фізичної моделі. Особливо для опису процесів у фізиці твердого тіла, де характерна відстань між елементами (атомами, молекулами) просторових структур у конденсованому їх стані практично незмінна. Нижче також показано, що проміжні стани з порушеним ближнім порядком, але зі збереженням далекого порядку також здатні формуватися в результаті структурно-фазових переходів (фазових переходів другого роду) і демонструвати ту ж динаміку появи, що і регулярні просторові утворення.

Нижче буде з'ясовано механізм формування просторових структур конвекції. Детально розглянемо динаміку і характер фазових переходів між структурами різної топології. Обговоримо дефектність просторових структур.

3.2. ОПИС МОДЕЛІ

Якщо число Релея Ra перевищує критичне значення Ra_{thr}, тобто $Ra = Ra_{thr}(1+\varepsilon)$, в шарі рідини між горизонтальними поверхнями, що погано проводять тепло (уздовж осі z), виникає тривимірна конвекція (див., наприклад, [75]), що описується рівнянням Проктора–Сівашинського [97; 98], яке визначає динаміку температурного поля цього процесу в горизонтальній площині (X,Y):

$$\dot{\Phi} = \varepsilon^2\Phi + \gamma\cdot\nabla(\Phi\nabla\Phi) - (1-\nabla^2)^2\Phi + \frac{1}{3}\nabla\left(\nabla\Phi\left|\Phi\right|^2\right) + \varepsilon^2 f\,, \qquad (3.1)$$

де f – зовнішній адитивний шум, ε – параметр, що визначає перевищення порога розвитку конвекції, вважається досить малою і позитивно визначеною величиною. У рівнянні використовується двовимірний

оператор $\nabla\varphi = \vec{i}\cdot\frac{\partial\varphi}{\partial X} + \vec{j}\cdot\frac{\partial\varphi}{\partial Y}$, причому $\vec{i}$, $\vec{j}$ – ортогональні один до одного одиничні вектори в площині (X,Y) розділу середовищ. Доданок виду $\gamma\cdot\nabla(\Phi\nabla\Phi)$ відповідає за температурну залежність в'язкості. Тут $X \approx 4x\sqrt{\varepsilon}$, $Y \approx 4y$, $F \approx 4\sqrt{3}\cdot\Phi$ – безрозмірна температура в одиницях різниці між підтримуваною температурою нижньої площини T_d і верхньої площини T_u у відсутності конвекції; ψ – безрозмірна швидкість в одиницях коефіцієнта температуропровідності χ (який дорівнює коефіцієнту теплопровідності λ, поділеному на щільність ρ і на питому теплопровідність C_p) рідини; координати x, y в одиницях товщини шару d, часовий інтервал d^2/χ, число Релея – число, що визначає поведінку рідини під впливом градієнта температури (за перевищення його критичного значення виникають конвективні потоки) $Ra = \sigma g(T_d - T_u)\cdot d^3/\nu\chi$, ν – кінематична в'язкість (динамічна в'язкість дорівнює кінематичній, що помножена на щільність), σ – коефіцієнт теплового розширення рідини, g – прискорення вільного падіння. Відзначимо, що перехід від використовуваних змінних до реальних фізичних величин представлений у табл. 3.1.

Таблиця 3.1

Зв'язок змінних з реальними фізичними величинами

Фізична величина	Подання в явному вигляді
Температура $T(x\sqrt{\varepsilon}, y)$	$T_d + (T_d - T_u)(-y + F(x\sqrt{\varepsilon}, y))$
Горизонтальна швидкість ψ_y	$60\sqrt{\varepsilon}\cdot F_{x\sqrt{\varepsilon}}\cdot(2y^3 - 3y^2 + y)$
Вертикальна швидкість $-\psi_x$	$-30\varepsilon\cdot(F_{x\sqrt{\varepsilon}})_{x\sqrt{\varepsilon}}\cdot(y^4 - 2y^3 + y^2)$

У цих умовах рішення можна шукати в формі

$$\Phi = \varepsilon\sum_j a_j \exp(i\vec{k}_j\vec{r}) \tag{3.2}$$

з $|\vec{k}_j| = 1$. Перенормовуючи одиницю часу $\propto \varepsilon^2$ для повільних амплітуд a_j, отримаємо еволюційне рівняння [103]:

$$\dot{a}_j = a_j - 2\gamma\cdot a_{j+j_0}a_{j+2j_0} - \sum_{m=1}^{N} V_{mj}\,|a_m|^2\,a_j + f, \tag{3.3}$$

де коефіцієнти взаємодії визначені співвідношеннями

$$V_{jj} = 1, \tag{3.4}$$

$$V_{ij} = (2/3)\left(1 - 2\left(\vec{k}_i \vec{k}_j\right)^2\right) = (2/3)\left(1 + 2\cos^2 \vartheta\right), \tag{3.5}$$

де ϑ – кут між векторами $\vec{k}_i$ та $\vec{k}_j$. Покладемо $\vartheta_{j_0} = 2\pi/3$, при цьому $\vartheta_{j+j_0} = \vartheta_j + 2\pi/3$ та $\vartheta_{j+2j_0} = \vartheta_j + 4\pi/3$.

Ширина інтервалу нестійкості в *k*-просторі представляє собою кільце, середній радіус якого дорівнює одиниці, а ширина дорівнює порядку величини відносної надпороговості ε, тобто багато менше одиниці. Під час розвитку нестійкості через зростання нелінійних членів ефективний інкремент мод, що лежать поза вельми малою околицею поблизу одиничного кола, буде спадати і може змінити знак. Найбільший інтерес представлятимуть моди, що знаходяться саме на цьому одиничному колі, тобто розглянемо спрощену модель явища, вважаючи, що спектр коливань вже розташовується на одиничному колі в *k*-просторі.

Відзначимо, що, незважаючи на простоту формального уявлення, задача є в фізичному просторі тривимірною, а в математичній моделі – двовимірною, тобто розглядаються температурні плями виходу конвективних потоків на верхню межу шару. Двовимірність задачі створює значні труднощі в візуалізації та аналізі поведінки системи.

3.3. ОСОБЛИВОСТІ ЗАСТОСУВАННЯ ТЕХНОЛОГІЇ CUDA ДЛЯ ОПИСУ ПРОЦЕСУ КОНВЕКЦІЇ

Програма, що реалізує математичну модель задачі, створена з використанням технології CUDA. Основним рівнянням для моделювання процесу конвекції було еволюційне рівняння. Головною обчислювальною складністю було збільшення кількості мод, що беруть участь у процесі, однак, незважаючи на великі обсяги обчислювальних даних, час обчислення основного рівняння на CPU був достатньою мірою маленьким. Втрати в часі починалися під час розрахунків поведінки температурного поля.

Температурне поле є сумою всіх мод у просторі, обмеженому граничними умовами. Нехай $\vartheta_n(t=0)$ для кожної моди задані початкові значення $a_n(t=0) = 0{,}01/N$ випадковим чином від нуля до π (від π до 2π все подібно – симетрія), причому розбити інтервал треба на *N* (це кількість мод). Тоді кожна *n*-на мода буде представляти собою

$$a_{n,m} Sin(2\pi nx) Sin(2\pi my), \tag{3.6}$$

де $a_n = a_{n,m}$, *n, m* (їх можна представити як $n = NCos\,\vartheta_s$, $m = NSin\,\vartheta_s$) – цілі числа, причому $N^2 = n^2 + m^2$. Під час розрахунку достатньо підсумувати за *n*, оскільки *m* визначається зі співвідношення $m^2 = N^2 - n^2$.

Картина температурного поля в інтервалі *-Lx/2 < x < Lx/2, -Ly/2 < y < Ly/2* є простою сумою всіх мод (підсумовування тільки за *n)*. Обчислення температурного поля $A_{x,y}$ (рис. 3.1) для одного часового кроку представлено рівнянням

$$A_{x*y+x} = \sum_{n=-N}^{n=+N} a_n Sin(2\pi nx) Sin(2\pi y\sqrt{N^2 - n^2}), \qquad (3.7)$$

де індекс A_{x*y+x} означає використання одновимірного масиву для зберігання значень температурного поля. Температурне поле повинно зберігатися в двовимірному масиві $A_{x,y}$, але вимушено використовується одновимірний масив, тому що під час написання власних функцій на CUDA в пам'яті GPU потрібно використовувати одновимірні масиви. Значення *x*, *y* змінюються від 0 до бажаної деталізації картини температурного поля, що далі позначається як «detalization».

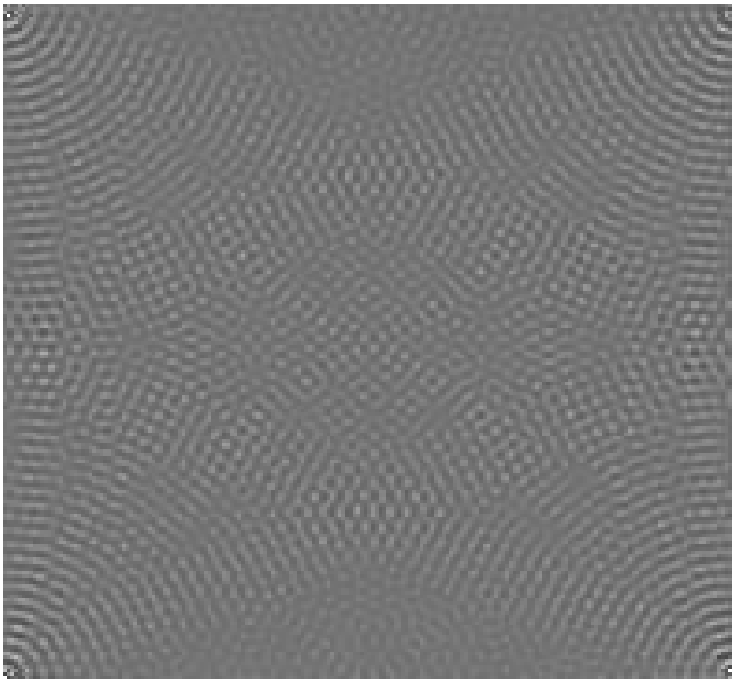

Рис. 3.1. Температурне поле, просторова структура – спотворені вали

На схемі алгоритму (рис. 3.2) показаний алгоритм проведення обчислень на GPU.

Код основної частини програми мовою Java, який керує обчисленнями на GPU:

dim3 nThreads = new dim3(detalization, 1, 1); // установка розміру блоку, який дорівнює detalization

dim3 nBlocks = new dim3(1, 1, 1); // встановлюємо розмір сітки величиною в 1 блок

KernelLauncher kernelLauncher=KernelLauncher.create(folder,"func",""); // folder – адреса файлу cuda-модуля, func – функція, яка запускається під час розрахунку на GPU

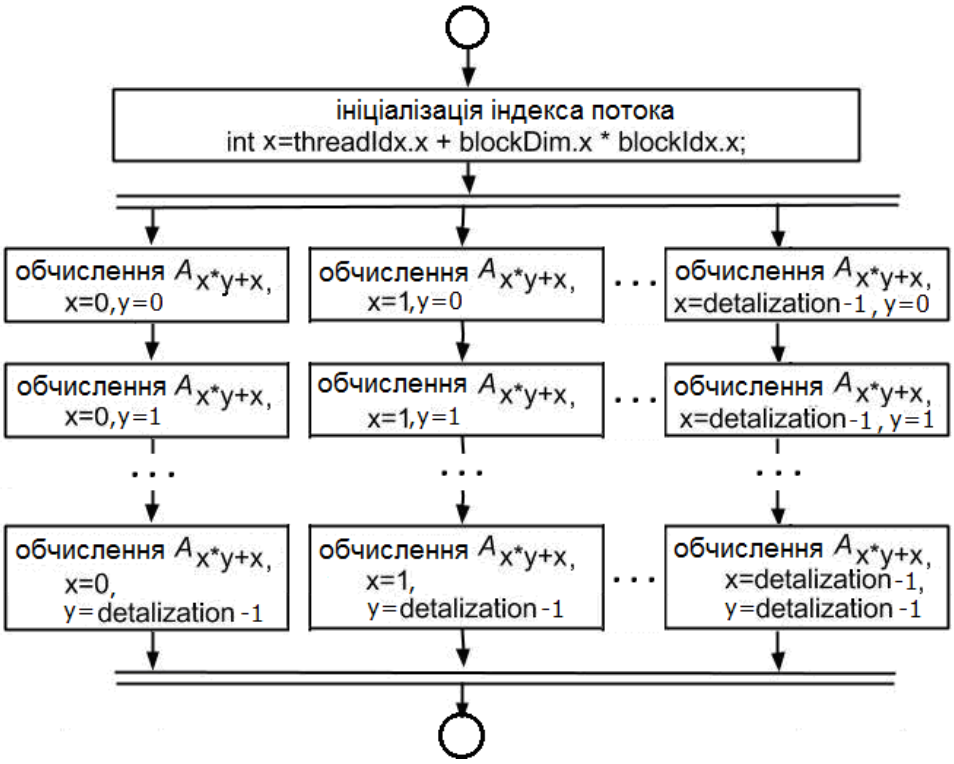

Рис. 3.2. Схема алгоритму розрахунку температурного поля на GPU

CUdeviceptr a_array = new CUdeviceptr(); // створення покажчика на пам'ять, яка буде зберігати амплітудний вектор

cuMemAllocal(a_array, a.length); // виділення пам'яті на GPU для a_array

cuMemcpyHtoD(a_array, Pointer.to(a), a.length); // поміщення масиву в пам'ять GPU

CUdeviceptr A_array = new CUdeviceptr(); // створення покажчика на пам'ять, яка буде зберігати температурне поле

cuMemAlloc(A_array, A.length); // виділення пам'яті на GPU для A_array

kernelLauncher.setup(nBlocks,nThreads).call(detalization,N,Math.PI,a_array.getPtr(),A_array.getPtr()); // запуск обчислень на GPU, де N – кількість мод, Math.PI – число пі

cuMemcpyDtoH(Pointer.to(A), A_array, A.length); // отримання температурного поля з пам'яті GPU

A_array.memFree(); // очищення пам'яті GPU

a_array.memFree(); // очищення пам'яті GPU

Код обчислень на GPU:

```
extern "C" __device__ void threadFunc(int detalization,int N, double *a, double pi,int x, double *A){
        for(int y=0;y<detalization;y++){
            double sum=0;
            for(int n=-N;n<=N;n++) {

sum+=a[n+N]*sin(2.0*pi*x*n)*sin(2.0*pi*y*sqrt((double)N*N-n*n));
            }
            A[x*y+x]=sum;
        }
}

extern "C" __global__ void func(int detalization,int N, double pi, double *a, double *A){
        int x = threadIdx.x + blockDim.x*blockIdx.x;
        threadFunc(detalization,N,a,pi,x,A);
}
```

Обчислення проводяться на відеокарті GeForce GTS 450 (обчислювальна потужність 601 GFlops) і на одному ядрі двоядерного процесора AMD Athlon X2 250 (6,0 GFlops на ядро).

Під час розрахунку температурного поля всі обчислення виконуються на GPU. На CPU виконується підготовка розрахунку і малювання температурного поля на основі даних, отриманих із GPU.

Швидкість виконання програми залежить від кількості мод і величини деталізації. На основі отриманих результатів розрахунків будується графік часу виконання програми на GPU і CPU (рис. 3.3).

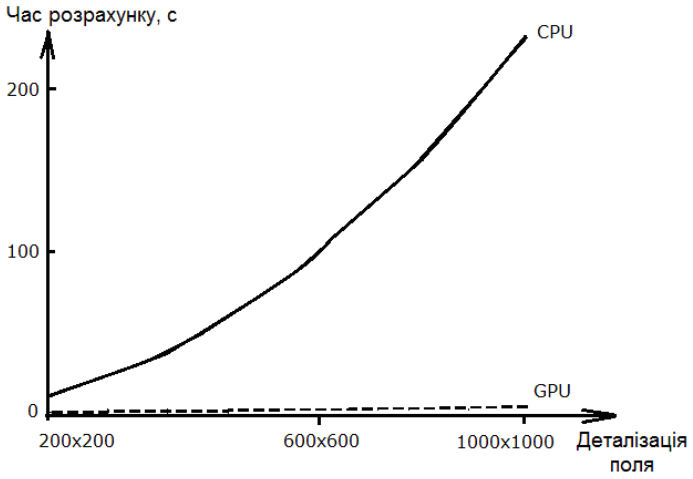

Рис. 3.3. Час виконання програми на GPU и CPU залежно від деталізації картини температурного поля. Кількість мод $N = 200$

За 200 мод і деталізації поля 1000x1000 точок обчислення на GPU відбуваються в 67 разів швидше, ніж на CPU.

3.4. РЕЗУЛЬТАТИ ЧИСЕЛЬНОГО МОДЕЛЮВАННЯ

Конвекція у відсутності температурної залежності в'язкості. У відсутності температурної залежності в'язкості $\gamma = 0$ в [96; 97] показано, що в разі великої кількості мод за високої точності розрахунків система затримується в своєму розвитку, залишаючись у динамічній рівновазі. Розвиток збурень у системі, як показує чисельний аналіз рівняння (3.3), відбувається наступним чином [96].

З початкових флуктуацій швидко збуджується широкий спектр за ϑ. По досягненні першого піку похідної виникає так званий «аморфний» стан системи. Значення квадратичної форми цього спектра температурного поля $I = \sum_j a_j^2$ можна оцінити, прирівнявши праву частину (3.3) до нуля, при цьому отримаємо значення близьке до 0,75.

Можна переконатися в тому, що функцією стану є величина $I = \sum_i a_i^2$. Для кожної структури є певне рівноважне значення функції стану, що відповідає її топології. Проте, не дивлячись на тривалий час існування кожної квазістаціонарної структури, існує механізм її руйнування і формування структури з іншою топологією.

Для подальшого розвитку – «кристалізації» – одна з мод повинна отримати порцію енергії, що перевищує певний поріг. Тобто в цих умовах необхідна наявність певного рівня шуму – флуктуацій. Це досягається за кінцевого значення шуму $f \neq 0$ або під час зменшення точності розрахунків, що, як зазначається в [97] еквівалентно. Подібні випадки, коли шум здатний спровокувати або прискорити процес нестійкості, зібрані в книзі [95].

Якщо одна з мод отримує потрібну порцію енергії, розвивається процес формування найпростішої конвективної структури – валів (рис. 3.4). Величина I при цьому досягає значень, близьких до одиниці ($I \to 1$). Однак цей стан не є стійким і спостерігається структурний перехід: конвективні вали зазнають модуляцію вздовж осі обертання рідини, характерний розмір якої скорочується. У цьому перехідному стані система знаходиться досить великий час (який незначно зростає в певних межах під час збільшення кількості мод), при цьому зберігається значення $I \approx 1.07$. Через досить великий час, що в десятки разів перевершує зворотний інкремент початкової лінійної нестійкості, з новоутвореного «бічного» спектра «виживає» лише одна мода, амплітуда якої порівнюється з амплітудою первісної лідируючої моди. В кінцевому

підсумку формуються стійка конвективна структура – квадратні осередки, за якої квадратична форма системи досягає значення $I = 1.2$.

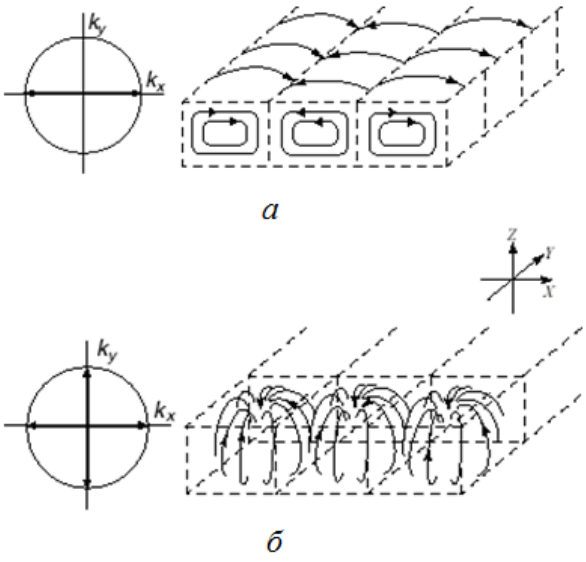

Рис. 3.4. Конвективні вали (а), квадратні осередки (б)

Подальші дослідження процесу виявили наступну динаміку зміни інтегральної квадратичної форми $I = \sum_j a_j^2$ з часом (рис. 3.5).

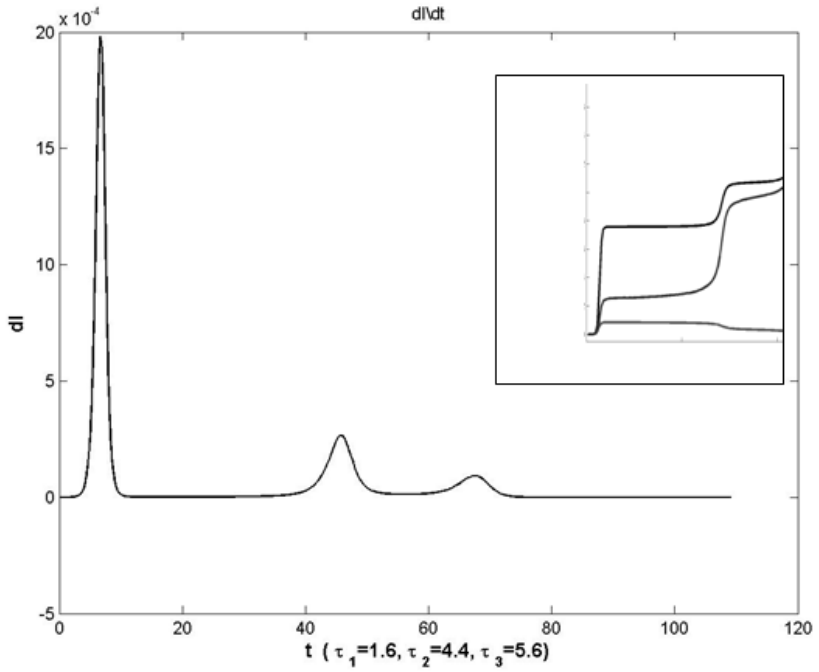

Рис. 3.5. Динаміка похідної dI/dt *(у відносних одиницях) функції стану* $I = \sum_j a_j^2$.
На вкладці верхня крива демонструє поведінку I,
середня – поведінку середньоквадратичного відхилення $\sqrt{\sum_i (A_i - \bar{A})^2}$
і нижня – поведінку середньої амплітуди $\frac{1}{N}\sum_i A_i = \bar{A}$ *як функцій часу*

Саме після другого сплеску похідної формується вторинна метастабільна структура з новим значенням $I \approx 1.07$. Після третього сплеску похідної квадратичної форми починає формуватися стабільна структура конвективних осередків. Подібна поведінка системи переконує в існуванні структурно-фазових переходів в цій системі.

Часи розвитку релаксаційних процесів під час руху системи до більш рівноважного стану зазвичай визначаються різницею значень функції стану після переходу і до нього. Чим більше ця різниця, тим швидше відбувається процес переходу з одного стану в інший.

Важливо відзначити й інше: послідовність зміни станів визначається часом розвитку нестійкостей (виконують роль релаксаційних процесів), що забезпечують перехід до все більшого наближення до рівноважного стану системи. Причому раніше проявляють себе більш швидкі релаксаційні процеси, зумовлені великими перепадами рівноважних значень функції стану. Переконаємося, що і в цьому випадку всі явища відбуваються в подібній послідовності і в рамках подібного сценарію. Саме чисельний аналіз моделі дозволяє підтвердити ці міркування.

Можна переконатися в тому, що часові терміни формування станів τ_n зворотно пропорційні різниці між значеннями $I = \sum_i a_i^2$ після n-го структурно-фазового переходу $I_n^{(+)} = (\sum_i a_i^2)_n^{(+)}$ і до цього переходу $I_n^{(-)} = (\sum_i a_i^2)_n^{(-)}$. Тобто $\tau_n \propto \{(\sum_i a_i^2)_n^{(+)} - \sum_i a_i^2)_n^{(-)}\}^{-1} = \Delta I_n^{-1}$, $\tau_3 / \tau_2 \approx \Delta I_2 / \Delta I_3$.

Таким чином, на основі чисельного дослідження моделі Проктора–Сівашинського показано, що функцією стану, що володіє певною топологією, є сума квадратів амплітуд мод $I = \sum_i a_i^2$, тому що структури, які виникають під час переходів, описуваних сплесками похідною цієї функції, мають відмінні топології і характеризуються її фіксованими значеннями.

Розглянемо більш докладно формування стабільної структури конвективних осередків. Позначимо амплітуди мод, що формують у кожній реалізації просторову структуру квадратних конвективних осередків, як a_1 та a_2. На інтервалі між другим і третім сплеском похідної квадратичної форми (рис. 3.6) вивчимо динаміку «спектральної дефектності» структури $D = \sum_{j \neq 1,2} a_j^2 / \sum_j a_j^2$, заснованої на відношенні квадратів амплітуд мод спектра, що не відповідає системі квадратних осередків, до повної суми квадратів мод, а також так звану «візуальну дефектність» $d = N_{def} / N$, де N_{def} – кількість дефектних просторових осередків (площа структури, зайнята нерегулярними осередками)

і N – кількість осередків в ідеальній регулярній структурі (повна площа структури).

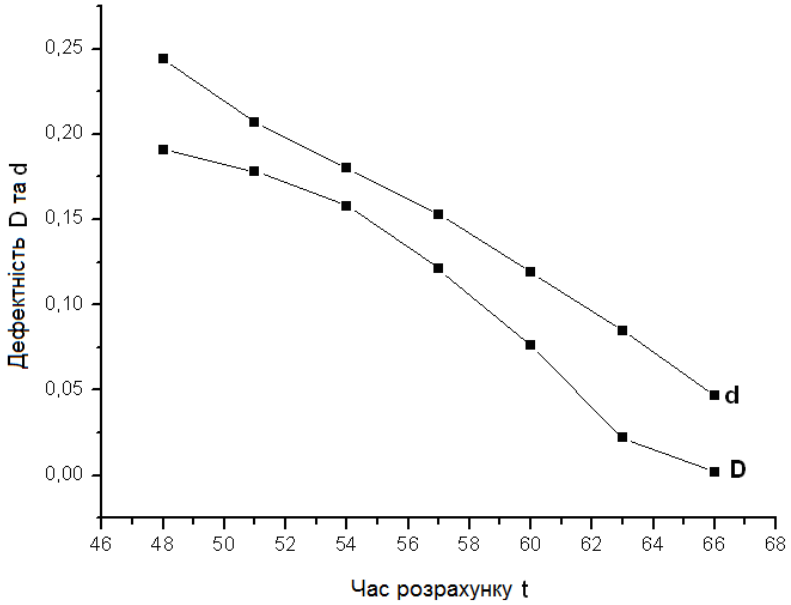

Рис. 3.6. Порівняльний аналіз спектральної D і візуальної d дефектності. Кількість мод 50

Критерії, за якими вважати осередок правильним і метод, що дозволяє підрахувати кількість цих осередків, наступні. Отримана картина для поля переводиться в режим 8-бітного зображення. Тобто максимальна кількість кольорів зменшується до 256. Тим самим сформована структура стає більш явно помітною. За збільшення подібного зображення можна досить явно розрізнити, яка зі структурних одиниць є необхідним нам правильним осередком, а яка – ні. Правильний осередок має правильну геометричну форму з рівномірно темним центром і порівняними з центром за розміром чотирма височинами світлішого рівномірного забарвлення.

Незважаючи на якісний характер опису величин, що характеризують спектральну і візуальну дефектність структури, можна відзначити подібну їх поведінку (рис. 3.7) під час наближення до завершення структурного переходу.

У разі досить великого рівня шуму, як адитивного ($f \neq 0$), так і мультиплікативного (випадкова складова в ε^2 в першому члені правої частини (3.1)), рівень амплітуд мод із самого початку може виявитися не малим. Початкові умови також можуть бути такими, що стартовий стан системи можна вважати «аморфним» з високим ступенем невпорядкованості, тобто якщо амплітуди збурень досить великі і відрізняються одна від одної за випадковим законом. Такий стан може підтримуватися в подальшому випадковим шумом. Важливо з'ясувати, за яких рівнів шуму вдається зберегти «аморфний» стан, що характеризується наявністю великої кількості просторових мод.

Шум дуже великої інтенсивності здатний утримати систему від формування конвективних структур, однак шум меншої інтенсивності не здатний перешкодити утворенню послідовно метастабільного (вали) і стабільного (квадратні осередки) станів. За зменшення амплітуди шуму процес переходу з метастабільного в стабільний стан здатний сповільнитися, коли система надовго затримується («заморожується») в метастабільному стані.

Конвекція за обліку температурної залежності в'язкості. Облік залежності в'язкості від температури визначається складовою виду $\gamma\nabla(\Phi\nabla\Phi)$ в (3.3), що пропорційна γ. Для $\gamma > 0$ (конвекція газу) потік газу піднімається до центру осередку, для $\gamma < 0$ (конвекція рідини) потік рідини рухається у напрямку від центру осередку вниз (див., наприклад, [105]). За $|\gamma| << 1$ вплив цього доданка на характер процесу невеликий. Процес конвекції відбувається за сценарієм, що обговорюється вище. Але за наближення величини γ до одиниці додатковий механізм обміну енергії між кожною трійкою мод, що орієнтовані по сторонах рівностороннього трикутника, руйнує механізм багатохвильової взаємодії кубічної векторної нелінійності. Причому наслідки цього руйнування в разі різного знака γ виявляються практично однаковими. Перш за все, швидкий ріст мод спектра на лінійній стадії формує квазістійку структуру з досить вибагливою топологією, що залежить від початкових умов. Однак після невеликого проміжку часу відбувається другий структурний перехід (рис. 3.7), в результаті якого формуються стійкі протяжні і чітко виражені вали, структура яких представлена на рис. 3.1, а просторовий розподіл температурного поля структури на рис. 3.8.

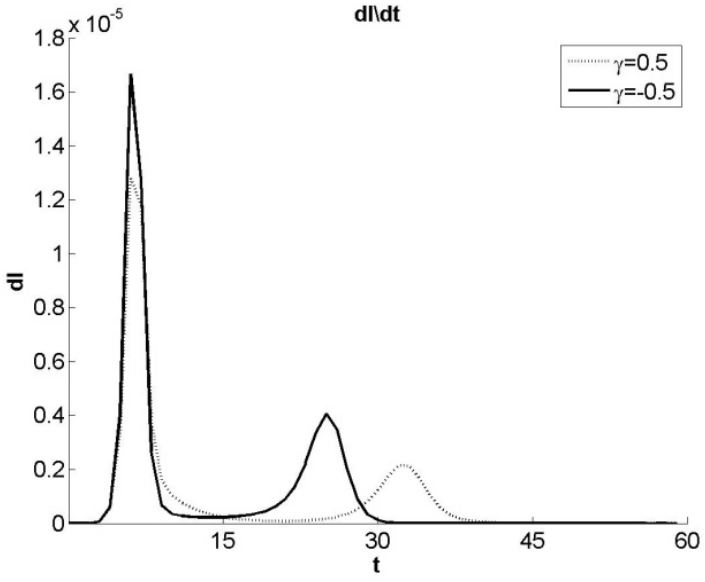

Рис. 3.7. Зміна похідної $\partial I / \partial t$ під час розвитку процесу за $|\gamma| = 0.5$

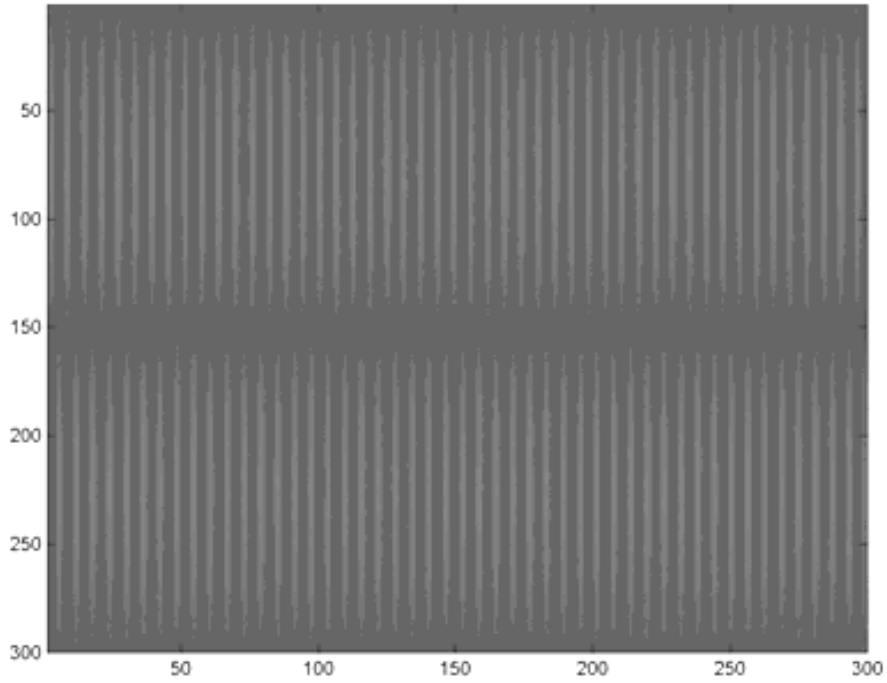

Рис. 3.8. Температурне поле, що відповідає формуванню конвективних валів за $|\gamma| = 0.5$

Характер структурних переходів зрозумілий з рис. 3.9, де можна бачити безперервність функції $I = \sum_j a_j^2$, що характеризує стан системи.

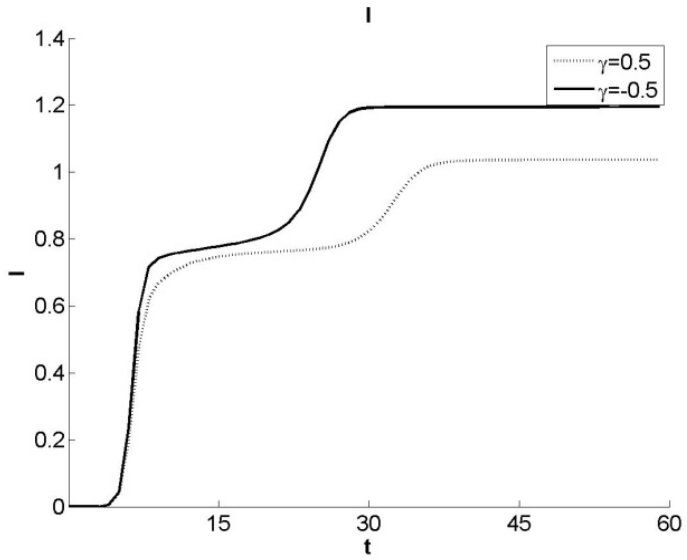

Рис. 3.9. Поведінка функції $I = \sum_j a_j^2$ *,*
що характеризує стан системи за $|\gamma| = 0.5$

Таким чином, помітна залежність в'язкості від температури здатна формувати стійкі конвективні вали. Такі конвективні вали можуть формуватися в областях тонкої хмарності (рис. 3.10).

Рис. 3.10. Формування конвективних валів довжиною в сотні кілометрів на північ від Австралії на початку періоду дощів

ВИСНОВКИ

Застосування технології CUDA: обчислення диференційних рівнянь у цій математичній моделі не займає багато часу, паралельні обчислення не потрібні. Технологія CUDA виявилася корисною для розрахунку картини температурного поля. Зважаючи на специфіку роботи з матрицями, ця задача легко розпаралелюється і приріст швидкості складає в середньому 50–60 разів для обраних початкових даних – 200 мод, деталізації решітки до 1000x1000 точок. Показаний алгоритм для одного часового кроку, але фізика процесу має на увазі, що виклик цього рахунку буде відбуватися багато разів, що передбачає можливість спостереження зміни динаміки поведінки температурного поля і формування просторових структур.

Результати моделювання: особливістю моделі Проктора–Сівашинського для опису конвекції ***за відсутності температурної залежності в'язкості*** є наявність трьох станів. Саме з чисельних розрахунків стало ясно, що часові терміни структурних переходів між метастабільними станами набагато менші за час їх існування. Кожний стан, що має певну топологію, характеризується певним рівноважним значенням функції стану $I = \sum_i A^2{}_i$. Квазістійкі стани руйнуються через нестійкості, час розвитку яких можна оцінити за величиною імпульсу (сплеску) в часі похідної функції стану. Показано, що характерні часові терміни нестійкостей, що руйнують старий і формують новий стан зворотно пропорційні різниці між значеннями функції стану після і до структурно-

фазового переходу. Також відзначено, що більш швидкі релаксаційні процеси, тобто структурно-фазові переходи, передують більш повільним. Характерний розмір конвективних утворень у режимі розвиненої нестійкості і в прийнятих одиниць виміру порядку $2\pi / k \propto 2\pi$, а довжина хвильових векторів порядку одиниці. Потенціал взаємодії просторових мод $V_{ij} = (2/3)\left(1 + 2\cos^2 \vartheta_{ij}\right)$ має глибокий мінімум для кутів $\vartheta_{ij} = \vartheta_i - \vartheta_j$ між векторами $\vec{k}_i$ та $\vec{k}_j$ двох просторових мод $\vartheta_{ij} = \pm\pi / 2$. Саме ці мінімуми породжують нестійкість структури валів [95; 96]. Бо існування мінімуму V_{ij} для мод із відносно невеликими амплітудами дозволяє їм продовжувати своє зростання, пригнічуючи при цьому збурення, які виникли раніше. За наближення до стабільного стану просторова структура позбавляється від безлічі дефектів, причому спостерігається кореляція між відносною часткою спостережуваних візуально (геометрично) дефектів структури і величиною дефектності, яка визначається як відношення квадратів амплітуд мод спектра, що не відповідає системі квадратних осередків до повної суми квадратів мод.

За урахування залежності в'язкості від температури в моделі Проктора–Сівашинського, що описує конвекцію тонкого шару рідини або газу з межами, що погано проводять тепло, виявляється придушення структурно-фазових переходів із формуванням конвективних квадратних осередків. Як і у відсутності залежності в'язкості від температури, в еволюції системи простежується поява довгоживучих квазістійких станів із топологією, яка визначається початковими умовами. Деякі відмінності для газу і рідини змінюють лише амплітуду кінцевої структури конвективних валів, не змінюючи характеру структурно-фазових переходів.

РОЗДІЛ 4

ОСНОВИ ВИКОРИСТАННЯ ОБЧИСЛЮВАЛЬНОЇ ТЕХНОЛОГІЇ CUDA

4.1. ВИКОРИСТАННЯ GPU ДЛЯ ОБЧИСЛЕНЬ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ

На відміну від ***центральних процесорів*** (central processing unit, CPU), ***графічні процесори*** (graphics processing unit, GPU) мають продуктивність паралельної архітектури – велику кількість ядер невеликої частоти, що означає виконання багатьох паралельних потоків повільно, а не виконання одного потоку швидко (рис. 4.1) [106]. Такий підхід обчислень загального призначення на GPU (обчислень, які зазвичай проводяться на CPU) відомий як ***general purpose computing on graphics processing units*** (GPGPU) [107]. Завдяки паралельній архітектурі швидкість обчислень на GPU вища, ніж на CPU, якщо порівнювати зіставні за ціною або енергоспоживанням пристрої.

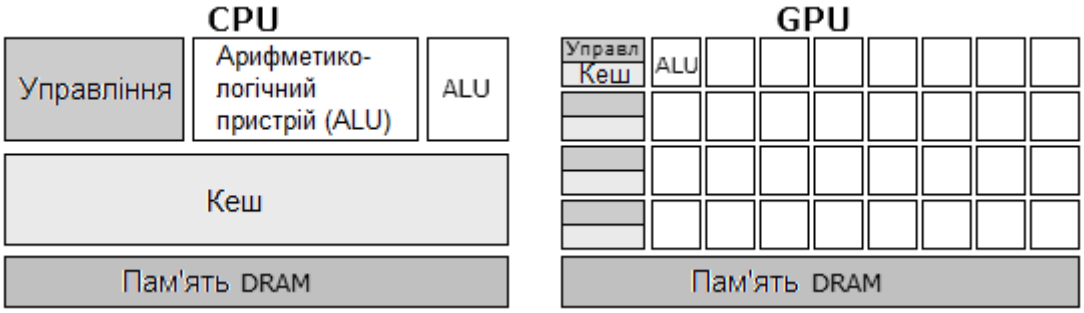

Рис. 4.1. Архітектури CPU та GPU

Такий принцип обчислень називається «***одиночний потік команд, множинний потік даних***» («single instruction, multiple data», SIMD). SIMD дозволяє забезпечити паралелізм на рівні даних. Завдяки очікуваним у майбутньому операційним системам (ОС) GPU може стати універсальним процесором для паралельних обчислень, який буде працювати з будь-яким додатком [108].

Основні GPGPU технології: ***CUDA*** та ***OpenCL***. Їх порівняння наведено, наприклад, у [109–112]. CUDA і OpenCL схожі у багатьох відношеннях:

- засновані на моделі SIMD;
- припускають поділ програмного коду на CPU і GPU частину;

- засновані на мові програмування С;
- моделі обчислень і пам'яті подібні.

Основні відмінності CUDA і OpenCL:

- CUDA підтримується тільки GPU від Nvidia, а OpenCL є відкритим із метою підтримки різними пристроями;
- CUDA включає в себе, окрім програмної, ще й апаратну складову;
- CUDA має велику підтримку з боку розробника (Nvidia), більше документації, більше поширення;
- обчислення на CUDA відбуваються швидше (зазвичай до 10 %), ніж на OpenCL.

4.2. ВСТУП ДО CUDA

Compute unified device architecture (CUDA) – архітектура паралельних обчислень, розроблена компанією Nvidia. CUDA забезпечує можливість обчислень загального призначення на GPU від Nvidia [113]. CUDA з'явилася в 2006 році і присутня в наступних серіях графічних чіпів, які використовуються в родинах GPU від Nvidia [114]:

- GeForce – GPU для домашніх комп'ютерів і ноутбуків;
- Quadro – GPU з поліпшеною обробкою графіки;
- Tesla – GPU для високопродуктивних обчислень;
- Tegra – GPU для мобільних платформ (смартфонів, планшетів та ін.);
- Jetson – вбудовані GPU для систем робототехніки, автомобілів і ін.

Найбільш важливі параметри пристроїв CUDA: обчислювальна потужність – кількість операцій із плаваючою крапкою (операцій над числами) в секунду (FLOPS), обсяг глобальної пам'яті, версія архітектури CUDA.

Програмісти використовують мову програмування «С для CUDA» (мова програмування С з розширеннями і обмеженнями) для написання алгоритмів, що виконуються на GPU. Також CUDA надає обчислювальні інтерфейси для Python, Perl, Fortran, Java, Mathematica, MATLAB та інших мов програмування і прикладних програм [108].

Переваги CUDA:

- висока швидкість обчислень (на GPU порівняно з CPU від декількох до кількох десятків разів);
- наявність інтерфейсів для різних мов програмування;
- безкоштовність;
- постійний розвиток;
- поширеність порівняно з іншими технологіями GPGPU.

Недоліки CUDA:

- додаткова складність написання програм для GPU порівняно з CPU;
- не підтримуються алгоритми, які не піддаються розпаралелюванню;

- архітектуру CUDA підтримує тільки виробник Nvidia;
- GPU, що мають старі версії архітектури CUDA, не підтримують можливості нових версій;
- швидкість обчислень у подвійній точності менше, ніж в одинарній точності, на відміну від CPU, які оптимізовані під обчислення в подвійній точності (на CPU швидкість обчислень у подвійній точності і в одинарній збігаються);
- висока вартість т. зв. «професійних GPU» – сімейств GPU Quadro і Tesla, оптимізованих для обчислень загального призначення, тобто які мають високу швидкість обчислень у подвійній точності.

4.3. МОДЕЛЬ ПРОГРАМУВАННЯ CUDA

Nvidia оперує визначеннями для CUDA, які відрізняються від визначень, що застосовуються для роботи з CPU [115]:
- потік (thread) – набір даних, який необхідно обробити;
- блок (block) – сукупність потоків;
- сітка (grid) – сукупність блоків;
- ядро (kernel) – програма, яка виконується однією сіткою блоків;
- хост (host) – CPU;
- девайс (device) – GPU.

Для управління потоками Nvidia створила модель «***одиночний потік команд, безліч ниток***» («single instruction, multiple thread», SIMT), котра подібна до SIMD [113]. Відмінність між моделями в тому, що в SIMT потоки об'єднуються в групи по 32 потоки, які називаються ***варпи*** (warp). Блоки потоків поділяються на варпи, а варпи виконуються фізично одночасно. Потоки, що належать одному варпу, також належать одному блоку. Тому розмір блоку впливає на кількість варпів і, відповідно, на швидкість обчислень.

GPU є обчислювальним пристроєм – співпроцесором (device) для центрального процесора (host) [116]. Ядром (kernel) називається код програми, що виконується паралельно на GPU. Модель програмування в CUDA передбачає групування потоків. Потоки об'єднуються в блоки потоків. Програма (kernel) виконується над сіткою (grid) блоків потоків (рис. 4.2). Сітки і блоки бувають одно-, дво- і тривимірними за формою.

CUDA включає в себе два інтерфейси програмування додатків (application programming interface, API): високого рівня (CUDA Runtime API) і низького (CUDA Driver API) (рис. 4.3) [116].

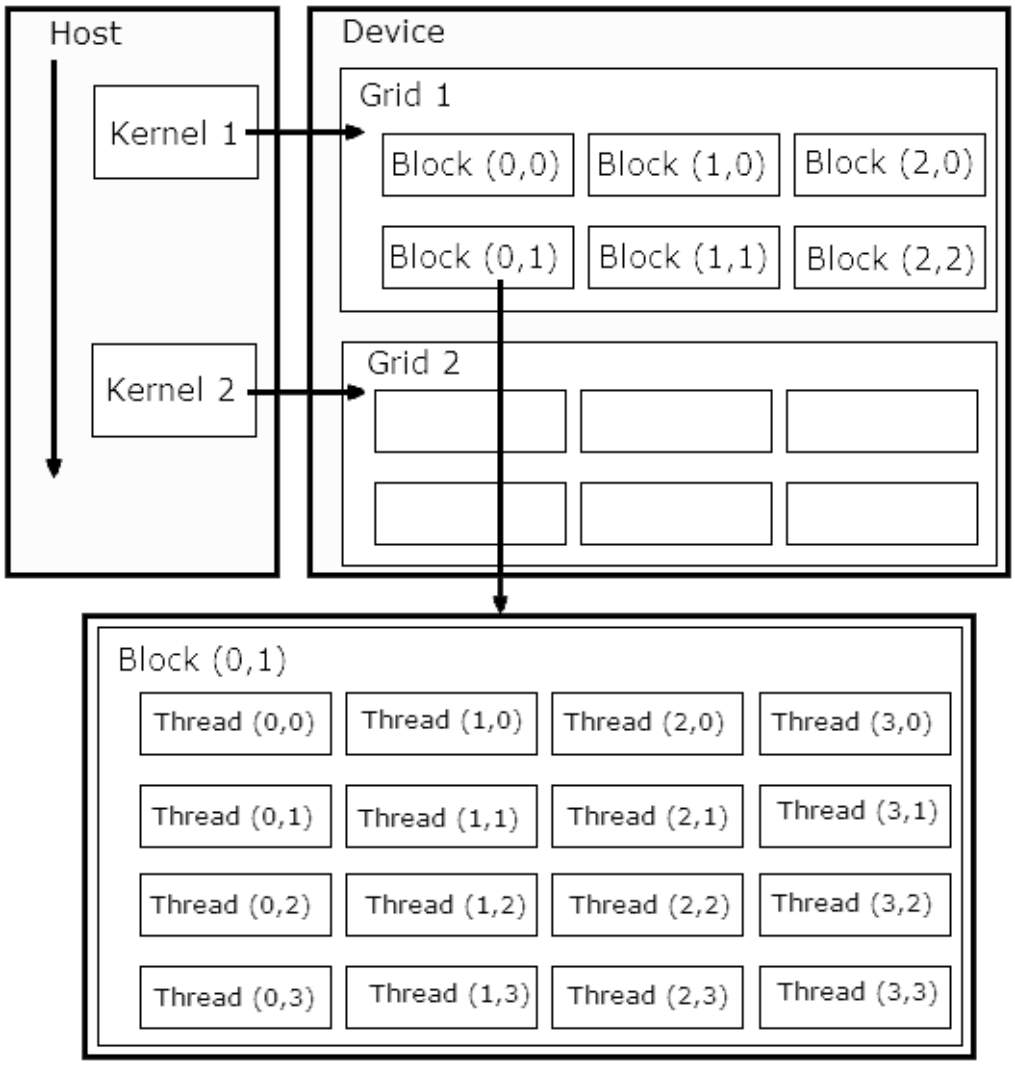

Рис. 4.2. Групування потоків

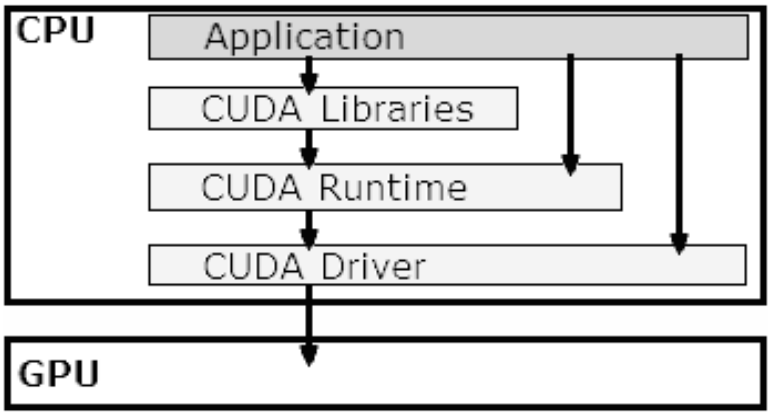

Рис. 4.3. Зв'язок CUDA API з прикладними програмами

Етапи виконання CUDA-програми (рис. 4.4):

1) підготовчий – завдання параметрів розпаралелювання (розмірів сітки і блоку), ініціалізація змінних, які будуть використовуватися в host- і device-частинах програми. Змінні host-частини програми заповнюються даними для обчислень;

2) виділення пам'яті на GPU і копіювання в неї даних для проведення обчислень. Дані зі змінних з host-частини програми копіюються в пам'ять GPU;

3) передача параметрів розпаралелювання і змінних у функцію і виконання цієї функції на GPU;

4) отримання результатів обчислення із пам'яті GPU назад у змінні в host-частину програми і звільнення пам'яті GPU.

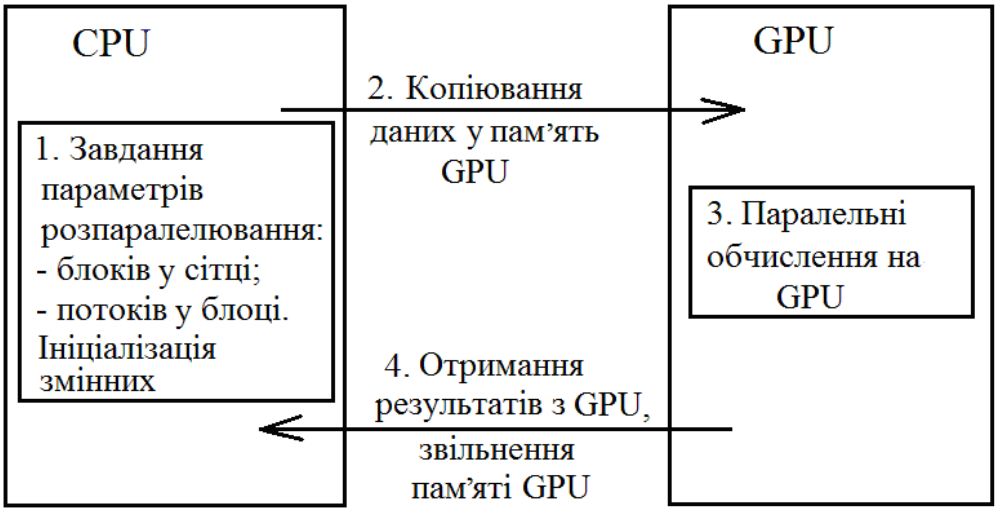

Рис. 4.4. Етапи виконання CUDA-програми

4.4. ВЕРСІЇ ОБЧИСЛЮВАЛЬНИХ МОЖЛИВОСТЕЙ CUDA ТА CUDA TOOLKIT

Пристрої CUDA мають різні версії архітектури GPU, які називаються версіями обчислювальних можливостей (compute capability) [106]. CUDA Toolkit містить драйвери, компілятори, відладчики, бібліотеки та інші інструменти для розробки програм (табл. 4.1).

Таблиця 4.1

Зв'язок версій compute capability та CUDA Toolkit

Рік	Версії compute capability	Версії CUDA Toolkit	Назва архітектури GPU	Основне нововведення
2006	1.0	-	Tesla	одинарна точність
2007	1.1	1.0, 1.1		атомарні функції
2008	1.2, 1.3	2.0		подвійна точність
2009	-	2.1, 2.2, 2.3		
2010	2.0, 2.1	3.0, 3.1, 3.2	Fermi	

Продовження табл. 4.1

2011	-	4.0, 4.1		
2012	3.0, 3.2	4.2, 5.0	Kepler	
2013	3.5, 3.7	5.5		динамічний паралелізм
2014	5.0, 5.2	6.0, 6.5	Maxwell	загальна віртуальна пам'ять GPU та CPU
2015	-	7.0, 7.5		
2016	6.0, 6.1	8.0	Pascal	багатошарова пам'ять, обчислення змішаної точності
план. 2018			Volta	

Розвиток архітектури GPU йде в бік підвищення обчислювальної потужності, швидкості передачі даних і енергоефективності GPU, а також у бік підтримки більшої кількості можливостей використання GPU і в бік зниження складності програмування на CUDA.

Основні нововведення версії обчислювальних можливостей 3:

- динамічний паралелізм. Потоки на GPU можуть динамічно народжувати нові сітки потоків [117]. Зі зведенням до мінімуму пересилання даних в CPU і назад спрощується паралельне програмування. Це також дозволяє застосовувати CUDA до більш широкого спектру алгоритмів;

- Hyper-Q – можливість до 32 звернень до GPU одночасно з передачею даних і команд [118]. Це дозволяє зробити рівномірне навантаження на GPU і прискорити обчислення. Корисно під час використання message passing interface (MPI) з CUDA;

- виклик бібліотек із коду, що виконується на GPU, покращує зв'язування об'єктів і спрощує написання програм [117];

- GPU-direct – прямий зв'язок між GPU і іншими пристроями PCI-E і прямий доступ до пам'яті між мережевими картами і GPU без звернення до CPU [117]. Це зменшує затримки між вузлами GPU в кластерах;

- Nvidia Nsight Eclipse Edition забезпечує швидке написання коду в середовищі Eclipse за допомогою вбудованого редактора CUDA [117].

Основні нововведення версії обчислювальних можливостей 5:

- уніфікована пам'ять (загальна віртуальна пам'ять GPU і CPU) дозволяє CUDA-додаткам отримувати доступ до пам'яті ОЗП і пам'яті GPU без необхідності вручну копіювати дані з одного виду пам'яті в інший;

- заміна CPU-бібліотек BLAS і FFT на GPU-еквівалент і їх автоматичне масштабування на вісім GPU.

Найновішими архітектурами CUDA є Pascal і Volta. У них очікується поява шини NVLINK для зв'язку GPU-GPU і GPU-CPU з більшою

пропускною здатністю, ніж PCIe, використання багатошарової (3D) пам'яті і технології Stacked Memory, яка передбачає розміщення динамічної пам'яті (DRAM) безпосередньо на чіпі GPU для збільшення її пропускної здатності, можливість обчислень змішаної точності [119].

ВИСНОВКИ

На відміну від центральних процесорів, графічні процесори мають продуктивність паралельної архітектури – велику кількість ядер невеликої частоти, що означає можливість одночасного виконання багатьох паралельних потоків. Графічні процесори можуть використовуватися для обчислень загального призначення завдяки таким технологіям як CUDA.

CUDA – архітектура паралельних обчислень, розроблена компанією Nvidia. За технологією CUDA графічний процесор є обчислювальним пристроєм – співпроцесором для центрального процесора. Модель програмування в CUDA передбачає групування потоків. Потоки об'єднуються в блоки потоків, а блоки – в сітки.

Найбільш важливі параметри пристроїв CUDA: обчислювальна потужність, обсяг глобальної пам'яті, версія архітектури CUDA. Програмісти використовують мову програмування Сі з розширеннями і обмеженнями для написання алгоритмів, що виконуються на GPU.

Етапи виконання CUDA-програми: ініціалізація змінних, виділення пам'яті на GPU і копіювання в неї даних для проведення обчислень, передача параметрів розпаралелювання і змінних у функцію і виконання цієї функції на GPU, отримання результатів обчислення із пам'яті GPU назад у змінні і звільнення пам'яті GPU.

Пристрої CUDA мають різні версії архітектури GPU, які називаються версіями обчислювальних можливостей. CUDA Toolkit містить драйвери, компілятори, відладчики, бібліотеки та інші інструменти для розробки програм.

РОЗДІЛ 5

НАВЧАЛЬНІ ПРИКЛАДИ ВИКОРИСТАННЯ CUDA

5.1. ДВА ОСНОВНІ ЗАСТОСУВАННЯ CUDA

CUDA забезпечує взаємодію з CUDA API і драйверами [120]. Перше застосування CUDA – взаємодія зі спеціалізованими бібліотеками, побудованими на основі CUDA API:

- cuBLAS – CUDA-бібліотека підпрограм основної лінійної алгебри (Basic Linear Algebra Subprograms);
- cuFFT – CUDA-бібліотека швидкого трансформування Фур'є (Fast Fourier Transforms);
- cuDPP – CUDA-бібліотека примітивів паралельних даних (Data Parallel Primitives);
- cuRAND – генератор випадкових чисел;
- cuSPARSE – CUDA-бібліотека для розріджених матриць (sparse matrix);
- AmgX, CULA, OpenCV, Nvidia Video Codec SDK і ін.

Під час використання спеціалізованих бібліотек написання коду програми майже не відрізняється від написання стандартних програм [120].

Однак спеціалізовані бібліотеки часто не пропонують необхідну функціональність. Тому друге застосування CUDA полягає в створенні власних функцій і їх виконанні на GPU викликом з основної (з CPU) частини програми [120].

5.2. СТВОРЕННЯ ВЛАСНИХ ФУНКЦІЙ

Код, що виконується на GPU, пишеться мовою програмування Сі у вигляді функцій, які викликаються з основної частини програми або з інших функцій CUDA [106].

Функції бувають трьох типів [106]:

- «__global__» – виконується на GPU, викликається з CPU;
- «__device__» – виконується на GPU, викликається з GPU;
- «__host__» – виконується на CPU, викликається з CPU.

Наприклад, необхідно виконати на GPU функцію «func», що викликається з основної частини програми:

```
__global__void func(<...список параметрів...>){
        <... тіло функції....>
}
```

Вхідними змінними є змінні основної частини програми. Під час опису вхідних змінних функції вказується їх тип і ім'я через кому (якщо вхідною змінною є масив, то перед його ім'ям вказується «*»).

Потім задається сітка й індекси потоків за допомогою змінних «threadIdx.x» (індекс потоку в блоці), «blockDim.x» (розмір блоку), «blockIdx.x» (індекс блоку). Ці змінні дійсні тільки в межах функції, яка виконується на GPU. Змінна 'i' є індексом потоку; такий запис забезпечує охоплення всіх потоків у сітці:

```
int i = threadIdx.x + blockDim.x*blockIdx.x;
```

Зручно розпаралелювати однакові операції над елементами масиву, коли для операції над окремим елементом масиву буде створюватися окремий потік, і він буде виконуватися паралельно з іншими потоками.

Наступна функція обчислює косинус для кожного елемента масиву «input» і записує результат в масив «output»

```
__global__ void func(double *input, double *output){
        int i = threadIdx.x + blockDim.x*blockIdx.x;
        output[i]=cos(input[i]);
}
```

Косинус обчислюється паралельно, для обчислення кожного елемента масиву створюється окремий потік. Можливо кілька викликів функцій із «__global__» функції. Обчислення синуса буде паралельним, але воно почнеться тільки після обчислення косинуса:

```
__global__ void func(double *input, double *output){
        int i = threadIdx.x + blockDim.x*blockIdx.x;
        output[i]=cos(input[i]);
        output[i]=sin(output[i]);
}
```

За більш складної структури функції не можна описати паралельні обчислення в простій функції, такій як синус або косинус, тому

створюється складна допоміжна функція типу «__device__», яка викликається з «__global__» функції:

```
__device__ double complexFunction(int b, int N, double *input){
        double sum=0;
        for(int a=0;a<N;a++){
                if(input[a]>=input[b]){
                        sum+=cos(input[b]-input[a]);
                }
        }
        return -2.0*sum/(double)N;
}

__global__ void func(int N, double *input, double *output){
        int i = threadIdx.x + blockDim.x*blockIdx.x;
        output[i]=complexFunction(i, N, input);
}
```

5.3. ВИКЛИК ВЛАСНИХ ФУНКЦІЙ

Виклик власних функцій CUDA відбувається в основній частині програми і має вигляд [106]:

```
func<<<numBlocks, threadsPerBlock>>>(N, ptrInput, ptrOutput);
```

де «func» – ім’я функції;
«numBlocks» – кількість блоків;
«threadsPerBlock» – кількість потоків в блоці;
«N, ptrInput, ptrOutput» – вхідні параметри функції.

Прості змінні можна безпосередньо вказувати в списку параметрів функції, для масивів необхідні покажчики.

Завдання розмірів сітки, тобто кількості блоків і потоків у блоці відбувається за допомогою змінних типу dim3 [106]. При цьому результат ділення *N/blockSize* повинен бути цілим числом. Якщо результат ділення не є цілим числом, він обов’язково повинен округлятися до більшого цілого, інакше потоки, що залишилися, не будуть охоплені індексом потоку:

```
int N=1000;
int blockSize = 50;
dim3 threadsPerBlock(blockSize, 1);
dim3 numBlocks(N/blockSize, 1);
```

де «blockSize» – розмір блоку – кількість потоків у блоці;
«N» – розмір масиву, який обробляється на GPU.

Для масивів «arrInput», «arrOutput» перед початком обчислень необхідно виділення пам'яті на GPU і поміщення цих масивів у пам'ять GPU [106]. Код на GPU проводить обчислення над масивом «arrInput» і результат записує в «arrOutput», тому «arrOutput» перед початком обчислень порожній і досить виділити для нього пам'ять GPU без копіювання «arrOutput» в цю пам'ять:

```
size_t size = N * sizeof(double);
double* arrInput=new double[N];

for(int i=0;i<N;i++){
        arrInput[i]=i;
}

double* arrOutput=new double[N];
double* ptrInput = (double*)malloc(size);
double* ptrOutput = (double*)malloc(size);
cudaMalloc((void**)&ptrInput, size);
cudaMalloc((void**)&ptrOutput, size);
cudaMemcpy(ptrInput, arrInput, size, cudaMemcpyHostToDevice);
```

де «size» – обсяг пам'яті для зберігання масиву;

«arrInput», «arrOutput» – масиви, які беруть участь в обчисленнях. При цьому масив «arrInput» необхідно заповнити значеннями;

«ptrInput», «ptrOutput» – покажчики на масиви в пам'яті GPU;

«cudaMalloc» – виділення пам'яті на GPU для масиву;

«cudaMemcpy» – копіювання масиву між ОЗП і пам'яттю GPU;

«cudaMemcpyHostToDevice» – копіювання масиву з ОЗП в пам'ять GPU.

Після завершення обчислень необхідно повернути результати в пам'ять ОЗП і очистити пам'ять GPU:

```
cudaMemcpy(arrOutput, ptrOutput, size, cudaMemcpyDeviceToHost);
cudaFree(ptrInput);
cudaFree(ptrOutput);
```

де «cudaMemcpyDeviceToHost» – копіювання масиву з пам'яті GPU в ОЗП;

«cudaFree» – очищення пам'яті на GPU для масиву.

5.4. ВИКЛИК БІБЛІОТЕЧНИХ ФУНКЦІЙ

Виклик бібліотечних функцій подібний до виклику власних функцій. При цьому відсутня необхідність завдання параметрів розпаралелювання [120].

Розглянемо виклик функції «Sscal» з бібліотеки cuBLAS. Ця функція помножує елементи матриці на число. Спочатку створюється змінна статусу і покажчик бібліотеки cuBLAS, а також покажчик на матрицю:

```
cublasStatus_t stat;
cublasHandle_t handle;
stat = cublasCreate(&handle);
float* ptrMatrix;
```

Потім матриця поміщується в пам'ять GPU, відбувається виклик функції «Sscal» і отримання результатів обчислення з пам'яті GPU. Список параметрів для виконання функції можна знайти в документації cuBLAS.

```
cudaMalloc ((void**)&ptrMatrix, size);
stat = cublasSetMatrix (<…список параметрів…>);
cublasSscal (handle, <…список параметрів…>);
stat = cublasGetMatrix (<…список параметрів…>);
```

Потім звільняється пам'ять GPU:

```
cudaFree(ptrMatrix);
cublasDestroy(handle);
```

5.5. ДИНАМІЧНИЙ ПАРАЛЕЛІЗМ

Динамічний паралелізм підтримується GPU Nvidia з версіями обчислювальних можливостей 3.5 і вище.

Потоки, що виконуються паралельно на GPU, можуть динамічно народжувати нові сітки потоків, в яких потоки будуть виконуватися паралельно [117].

Головний потік є батьківським, він запускає нову сітку потоків «child_func<<<10,100>>>», які є дочірніми [106]. Батьківський потік не вважається завершеним, поки всі дочірні потоки не завершаться (гарантується неявна синхронізація між батьківським і дочірніми потоками).

```
__global__ void child_func(double *array){
        array[threadIdx.x] = array[threadIdx.x]+1;
}

__global__ void parent_func(double *array){
        child_func<<<10,100>>>(array);
}
```

5.6. УНІФІКОВАНА ВІРТУАЛЬНА ПАМ'ЯТЬ

Уніфікована віртуальна пам'ять підтримується GPU Nvidia з версіями обчислювальних можливостей 3.0 і вище, а також CUDA Toolkit 6.0 і вище. Уніфікована пам'ять передбачає єдиний простір пам'яті ОЗП і GPU [106], усувається необхідність копіювання пам'яті командою «cudaMemcpy». При цьому замість команди «cudaMalloc» необхідно використовувати семантично подібну команду «cudaMallocManaged». Розглянутий вище приклад буде виглядати так:

```
int N=1000;
int blockSize = 50;
dim3 threadsPerBlock(blockSize, 1);
dim3 numBlocks(N/blockSize, 1);
size_t size = N * sizeof(double);
double* arrInput=new double[N];

for(int i=0;i<N;i++){
        arrInput[i]=i;
}

double* arrOutput=new double[N];
double* ptrInput = (double*)malloc(size);
double* ptrOutput = (double*)malloc(size);
cudaMallocManaged ((void**)&ptrInput, size);
cudaMallocManaged ((void**)&ptrOutput, size);
func<<<numBlocks, threadsPerBlock>>>(N, ptrInput, ptrOutput);
```

Під час використання уніфікованої пам'яті дані переміщуються між пам'яттю ОЗП і GPU залежно від того, який пристрій отримує доступ до пам'яті [106]. Це робиться для забезпечення швидкого доступу до пам'яті. Однак за наявності декількох GPU в системі дані не переміщуються між ними, а доступ до пам'яті інших GPU здійснюється по шині PCIe.

5.7. ВИКОРИСТАННЯ КІЛЬКОХ GPU ДЛЯ ОБЧИСЛЕНЬ

Обчислення можна розпаралелювати на кілька GPU. Отримати кількість GPU в системі можна командою [106]:

```
int deviceCount;
cudaGetDeviceCount(&deviceCount);
```

Отримати властивості всіх GPU в системі можна через цикл:

```
for (int device = 0; device < deviceCount; device++) {
        cudaDeviceProp deviceProp;
        cudaGetDeviceProperties(&deviceProp, device);
}
```

Перед виконанням обчислень на певному GPU необхідно його вибрати командою «cudaSetDevice(1)», в якій вказується індекс GPU. Індекс за замовчуванням дорівнює 0. Далі для обраного GPU необхідно виконати типові операції, такі як операції з пам’яттю і виклик функцій, які виконуються на GPU.

5.8. АТОМАРНІ ОПЕРАЦІЇ

Можуть бути випадки, коли більше одного потоку намагаються записати значення в одну комірку пам’яті (таких випадків краще уникати). Запис значення на прикладі операції «x++» складається з трьох етапів – зчитування «x», підсумовування «x+1» і запис в «x» нового значення. Припустимо, два потоки намагаються одночасно виконати операцію «x++» за «x = 0». Може вийти, що вони одночасно прочитають початкове значення «x = 0» і після підсумовування буде записаний обома потоками результат «x=1», а повинно вийти «x = 2». Правильно буде, якщо операцію виконає спочатку один потік, який прочитає «0» і запише «1», а потім операцію виконає другий потік, який прочитає «1» і запише «2».

Для того щоб уникнути подібних помилок використовуються атомарні операції. Атомарна операція гарантує, що тільки один потік зможе здійснювати операції читання і запису над осередком пам’яті до завершення атомарної операції. Атомарна операція може бути використана тільки в функції типу «__device__». Список атомарних операцій представлений в [106]. Вид атомарної операції, яка читає значення за адресою «address» і додає до нього «value»:

```
atomicAdd(address, value);
```

Наприклад, до «i» елементу масиву «array» додається 1:

```
atomicAdd(&array[i],1);
```

Для деяких атомарних операцій необхідно явно написати код, який їх виконує (код можна знайти в [106]), наприклад, для операції з double-числами «atomicAdd(double address, double value)» пишеться окрема «__device__» функція. У той же час для операції з int-числами «atomicAdd(int address, int value)» цього робити не потрібно.

Недоліком використання атомарних операцій є зниження швидкості обчислень через обмеження доступу потоків до пам’яті.

5.9. БАГАТОВИМІРНІ СІТКИ І БЛОКИ

Сітки блоків і блоки потоків можуть бути 1-, 2-, 3-вимірними (x-, y-, z-вимірювання) [106]. Для доступу до вимірів використовуються змінні «threadIdx.x», «threadIdx.y», «threadIdx.z» (індекс потоку в блоці), «blockDim.x», «blockDim.y», «blockDim.z» (розмір блоку), «blockIdx.x», «blockIdx.y», «blockIdx.z» (індекс блоку).

Розмір блоків і сіток задається змінними типу «dim3», наприклад:

```
dim3 threadsPerBlock(4, 5, 6);
dim3 numBlocks(5, 6, 7);
```

При цьому масиви, що відправляються в пам’ять GPU, повинні бути тільки одновимірними. Тому дво- і тривимірні масиви повинні бути переведені програмістом в одновимірні. Наприклад, для перекладу двовимірного масиву в одновимірний необхідно послідовно пройти всі рядки двовимірного масиву і переписати їх значення в одновимірний масив.

Розглянемо приклад зведення в квадрат елементів тривимірного масиву розміром 4x4x4. Для цього використана сітка з одного тривимірного блоку:

```
int size = 4;
dim3 threadsPerBlock(size, size, size);
dim3 numBlocks(1, 1, 1);
```

Під час розрахунку індексу потоку на GPU враховуються три виміри, масив залишається одновимірним:

```
__global__ void func(float *array, int size){
    int thread = threadIdx.x + threadIdx.y*size + threadIdx.z*size*size;
    array[thread]=array[thread]*array[thread];
}
```

5.10. ТОЧНІСТЬ ОБЧИСЛЕНЬ НА CUDA

Після обчислень на CUDA бажано перевіряти правильність отриманих результатів на GPU з результатами на CPU. Це пов'язано з більшою складністю програмування на CUDA і, відповідно, з більшою ймовірністю здійснення помилок. Потім можна проводити обчислення тільки на GPU [121].

На CUDA обчислення в одинарній точності (тип змінних float) відбуваються швидше, ніж в подвійній точності. Однак одинарної точності може бути недостатньо – з'являються похибки в результатах. Тоді обчислення переводяться в подвійну точність (тип змінних double) шляхом зміни типів змінних float на double [121].

У CUDA кожна математична функція має кілька реалізацій, що відрізняються за точністю результату і швидкістю обчислення: double-функції (наприклад, sin(x), cos(x)), float-функції (наприклад, sinf(x), cosf(x)), функції зниженої точності (наприклад, __sinf(x), __cosf(x)). Таблиці функцій різної точності наведені в [106]. Точність функцій описується стандартом IEEE-754. Максимальний розмір помилки визначається як абсолютна величина різниці між правильно округленим результатом і результатом функції на CUDA (ulp error).

Деякі вбудовані в CUDA функції забезпечують недостатню точність обчислень, наприклад функції Бесселя. У цьому випадку на CUDA можна явно написати код, який розраховує значення функцій Бесселя [121].

За впевненості в правильності результатів для прискорення обчислень розрахунки або тільки їх частину можна переводити в одинарну точність. Наприклад, обчислення тригонометричних функцій на GPU можна виконувати в одинарній точності, використовуючи float-функції, а інші розрахунки в подвійній точності [121].

5.11. ОПТИМІЗАЦІЯ ОБЧИСЛЕНЬ НА CUDA

Під час використання технології CUDA важливу роль відіграє оптимізація, завдяки якій швидкість обчислень можна збільшити в кілька разів. Для цього необхідно врахувати основні рекомендації щодо оптимізації обчислень, детально описані в [113; 122; 123]:

- перенесення основних обчислень на GPU і їх максимальне розпаралелювання. Бажано перенести навіть ті ділянки коду, що не розпаралелюються або обчислюються на CPU швидше, ніж на GPU, для зниження обсягу передачі даних між пам'яттю ОЗП і GPU і, відповідно, зниження витрат часу на передачу даних [121]. Наприклад, у задачі є система із сотень чи тисяч однотипних рівнянь, що описують рух кожної

частинки, а також у задачі присутні окремі рівняння, що описують окремі показники процесу. Обчислення рівнянь проводяться багаторазово (задача Коші). Обчислення однотипних рівнянь добре розпаралелюється на CUDA, а обчислення окремих рівнянь складно розпаралелити. Але обчислення окремих рівнянь також доцільно перенести на GPU, де вони виконуються одним потоком, тобто непаралельно;

- завантаження вихідних даних для розрахунку в пам'ять GPU один раз [121]. Далі обмін даними між CPU і GPU повинен бути відсутнім, крім тих випадків, коли необхідно отримати результати з пам'яті GPU. Додатковою перевагою такого підходу є те, що частина даних існує тільки в пам'яті GPU і не дублюється в ОЗП для економії пам'яті. Така рекомендація добре підходить для задачі Коші і буде розглянута пізніше;

- використання швидких типів пам'яті GPU – розділеної (shared) і константної (constant), які мають невеликий обсяг і високу швидкість доступу [123]. У цій роботі для операцій на CUDA не вказуються специфікатори типів пам'яті, а значить, використовується основна пам'ять GPU – глобальна (global memory);

- підбір розміру блоку (кількості потоків у блоці). Рекомендується, щоб розмір блоку був не менший 32 і був кратний числу 32 (це пов'язано з розміром варпа). Також розмір блоку не може бути більше 1024 – це обмеження GPU [123];

- використання float-функцій і функцій зниженої точності;

- попередній розрахунок часто повторюваних ділянок коду [121]. Наприклад, у задачі доводиться обчислювати кілька разів функції з одним і тим же аргументом. Тоді розрахунок таких функцій можна провести попередньо, зберігши результати в окремому масиві і використовувати ці результати пізніше;

- відхід від випадків, коли більше одного потоку намагаються записати результат в одну комірку пам'яті. Крім уповільнення обчислень, це може призводити до помилок (для подолання помилок у таких випадках використовуються атомарні операції);

- використання сучасних GPU з підтримкою динамічного паралелізму і інших можливостей.

Корисно оцінити витрати часу на різні етапи роботи програми і під час оптимізації приділити увагу найбільш тривалим етапам [121]. При цьому проведення оптимізації не повинно сильно впливати на точність обчислень і не має ускладнювати код програми, тому що моделювання фізичних задач вимагає частої зміни параметрів і рівнянь.

ВИСНОВКИ

При створенні прикладних програм CUDA забезпечує взаємодію з CUDA API і драйверами через два основних застосування.

Перше застосування CUDA – взаємодія зі спеціалізованими бібліотеками, побудованими на основі CUDA API. Під час використання спеціалізованих бібліотек написання коду програми майже не відрізняється від написання стандартних програм.

Однак спеціалізовані бібліотеки часто не пропонують необхідну функціональність. Тому друге застосування CUDA полягає в створенні власних функцій і їх виконанні на GPU викликом з основної частини програми.

Код, що виконується на GPU, пишеться мовою програмування Сі у вигляді функцій, які викликаються з основної частини програми або з інших функцій CUDA. Вхідними змінними функцій є змінні основної частини програми. Під час опису вхідних змінних функції вказується їх тип і ім’я. Потім задається сітка й індекси потоків.

Зручно розпаралелювати однакові операції над елементами масиву, коли для операції над окремим елементом масиву буде створюватися окремий потік.

Після обчислень на CUDA бажано перевіряти правильність отриманих результатів на GPU з результатами на CPU. На CUDA обчислення в одинарній точності відбуваються швидше, ніж в подвійній точності. Однак одинарної точності може бути недостатньо – з’являються похибки в результатах. Тоді обчислення переводять в подвійну точність.

Під час використання технології CUDA важливу роль відіграє оптимізація, завдяки якій швидкість обчислень можна збільшити в кілька разів. Для цього необхідно врахувати основні рекомендації щодо оптимізації обчислень. Корисно оцінити витрати часу на різні етапи роботи програми і під час оптимізації приділити увагу найбільш тривалим етапам.

РОЗДІЛ 6

РОЗРОБКА ПАРАЛЕЛЬНИХ АЛГОРИТМІВ ДЛЯ ЗАДАЧІ КОШІ

6.1. МОЖЛИВІСТЬ РОЗПАРАЛЕЛЮВАННЯ ОБЧИСЛЕНЬ НА РІВНІ ДАНИХ

Багато процесів у фізиці і, зокрема, в цій роботі описуються диференційними рівняннями. Розв'язання диференційного рівняння з заданими початковими умовами називається задачею Коші [124].

Математичні моделі фізичних процесів є системами диференційних рівнянь. Кожне рівняння описує одну з характеристик фізичного процесу, наприклад швидкість згустку частинок. Для розрахунку швидкості кожної частинки згустку використовується окреме рівняння, при цьому рівняння швидкості однотипні, тобто складаються з однакових обчислювальних операцій. Але дані, які використовуються для розрахунку швидкості частинок, в якійсь мірі відрізняються для кожної окремої частинки. Тому, відповідно до моделі SIMD, обчислювати однотипні рівняння можна паралельно (рис. 6.1), тобто паралелізм в цьому випадку полягає у використанні однакових обчислювальних операцій до безлічі даних (розпаралелювання обчислень на рівні даних).

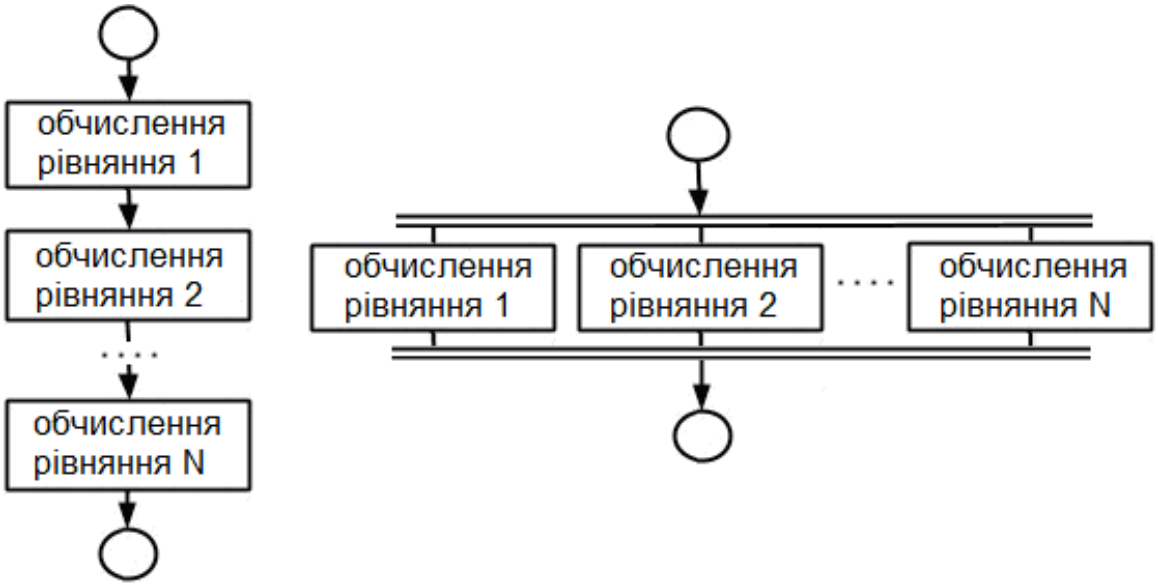

Рис. 6.1. Послідовне (праворуч) і паралельне (зліва) обчислення рівнянь

Інші рівняння також можна обчислювати паралельно (рис. 6.2).

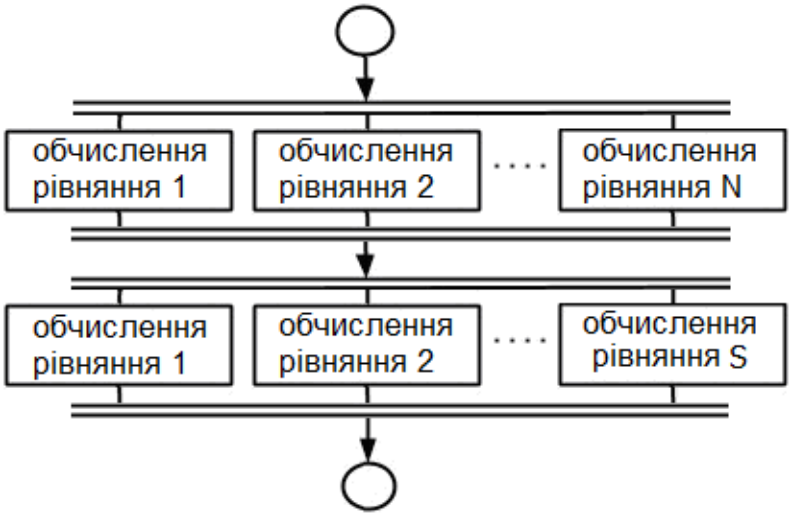

Рис. 6.2. Паралельний алгоритм для різних рівнянь

6.2. РОЗПАРАЛЕЛЮВАННЯ ОБЧИСЛЕНЬ З УРАХУВАННЯМ ОСОБЛИВОСТЕЙ ЗАДАЧІ КОШІ

Обчислювальний експеримент відбувається як розв'язання задачі Коші – встановлення значень параметрів у початковий момент часу, потім приріст часу і перерахунок усіх рівнянь, і т.д. багаторазово [121]. Тобто дані задаються перед початком обчислень, а потім у кожний момент часу використовуються дані, пораховані для попереднього моменту часу (рис. 6.3).

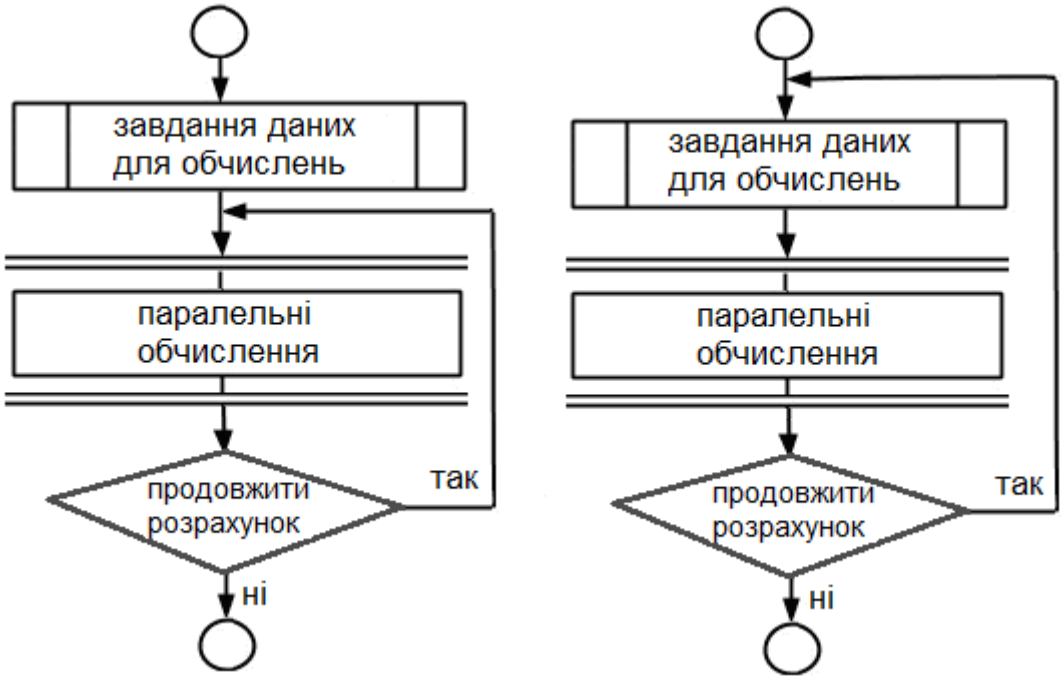

Рис. 6.3. Відмінність завдання даних для обчислень під час розв'язання задачі Коші (зліва) й інших задач (праворуч)

Алгоритм обчислювального експерименту під час розв'язання задачі Коші представлений на рис. 6.4. На схемі алгоритму показаний постійно повторюваний запуск обчислень на GPU з використанням двох циклів. У рамках вкладеного циклу обчислення відбуваються без отримання результатів із пам'яті GPU. Для підвищення точності обчислень зменшується крок за часом Δt, але за малого кроку розвиток фізичного процесу відбувається повільно і тому результати обчислень на кожному кроці за часом отримувати не потрібно. Тільки результати одного з певної кількості кроків за часом зберігаються, тобто переносяться з пам'яті GPU в ОЗП. Далі обчислення тривають за тією ж схемою, поки вони не завершаться. З урахуванням етапів виконання CUDA-програми в алгоритм додано завдання параметрів розпаралелювання, поміщення даних у пам'ять GPU, отримання результатів із пам'яті GPU, звільнення пам'яті GPU та ін.

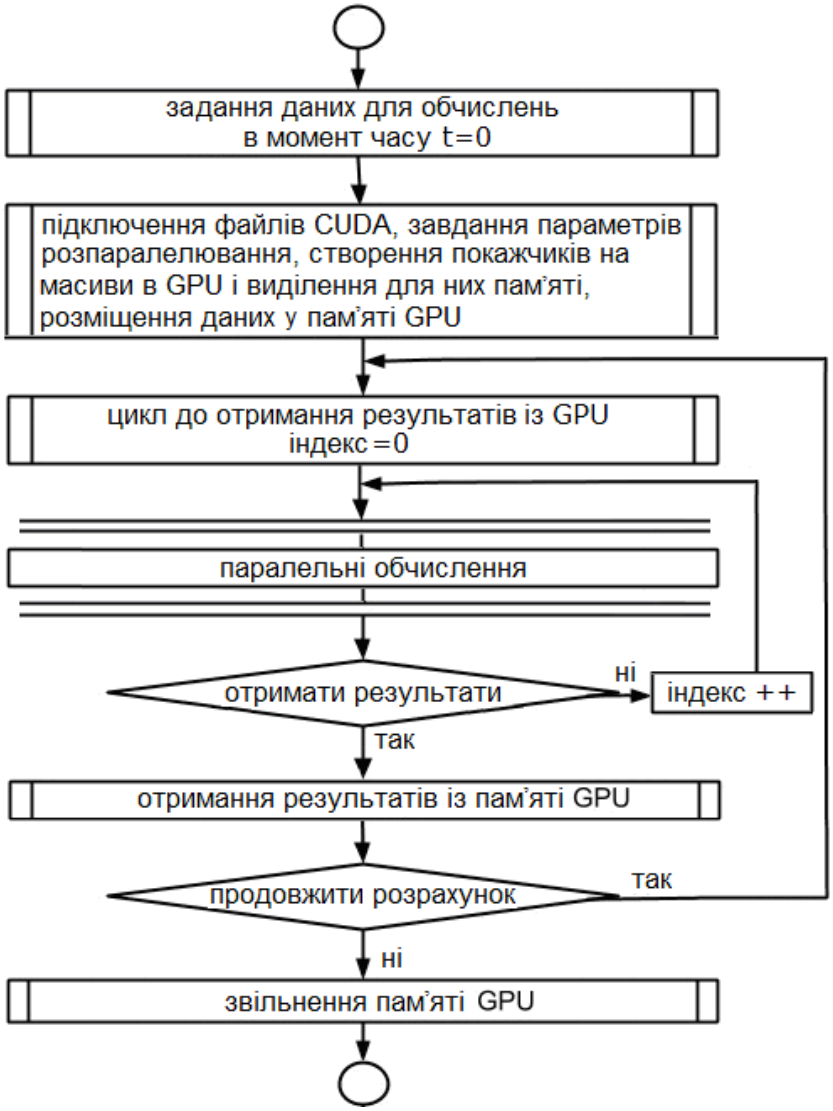

Рис. 6.4. Паралельний алгоритм та етапи виконання CUDA-програми

Як говорилося раніше, в CUDA програмний код ділиться на CPU- і GPU-частини. Блок паралельних обчислень належить до GPU-частини. Інші блоки алгоритму і цикли належать до CPU-частини, тому що їх виконання запускається з CPU.

Можна співвіднести структуру алгоритму (рис. 6.4) з етапами виконання CUDA-програми. На першому і другому етапах задаються дані для обчислень, ці дані поміщуються в пам'ять GPU, а також задаються параметри розпаралелювання для кожної функції, виконуваної на GPU. На третьому етапі відбуваються обчислення з використанням двох вкладених циклів, де «t» – змінний час, «T» – кінцевий час, «currentTime» – поточний час, «saveTime» – час, через який відбувається отримання результатів із пам'яті GPU. На четвертому етапі звільняється пам'ять GPU.

```
<задання даних для обчислень>
<задання параметрів розпаралелювання>
<переміщення даних у пам'ять GPU>
for (double t = 0; t <T; t + = saveTime) {
        double currentTime = t;
        for (int i = 0; i <(int) (saveTime / Δt); i ++) {
                function1 <<< numBlocks1, threadsPerBlock1 >>> (<вхідні змінні>);
                function2 <<< numBlocks2, threadsPerBlock2 >>> (<вхідні змінні>);
                ...
                functionN <<< numBlocksN, threadsPerBlockN >>> (<вхідні змінні>);
                currentTime + = Δt;
        }
        <отримання результатів з пам'яті GPU>
}
<звільнення пам'яті GPU>
```

Отримання результатів із пам'яті GPU зазвичай передбачає розрахунок додаткових параметрів на CPU, графічне відображення результатів і їх збереження в текстовому або графічному вигляді в файловій системі. Розрахунок додаткових параметрів потрібний для повноти результатів перед їх збереженням або відображенням, наприклад, для підготовки графіків. Тому такий розрахунок не потрібно проводити в кожен момент часу, і він виноситься з вкладених циклів і виконується на CPU.

Власні функції CUDA під час розв'язання задачі Коші реалізують один із чисельних методів розв'язання диференційних рівнянь. Далі буде розглянута реалізація найбільш простого чисельного методу – явного методу Ейлера 1-го порядку точності і найбільш поширеного чисельного методу – явного методу Рунге–Кутти 4-го порядку точності.

6.3. РЕАЛІЗАЦІЯ ЯВНОГО МЕТОДУ ЕЙЛЕРА НА CUDA

Згідно з методом Ейлера в кожен момент часу за допомогою кожного рівняння розраховується приріст функції $\Delta y = f(...) \cdot \Delta t$, а потім розраховується значення функції додаванням приросту до значення функції в попередній момент часу $y(t) = y(t - \Delta t) + \Delta y$ (рис. 6.5).

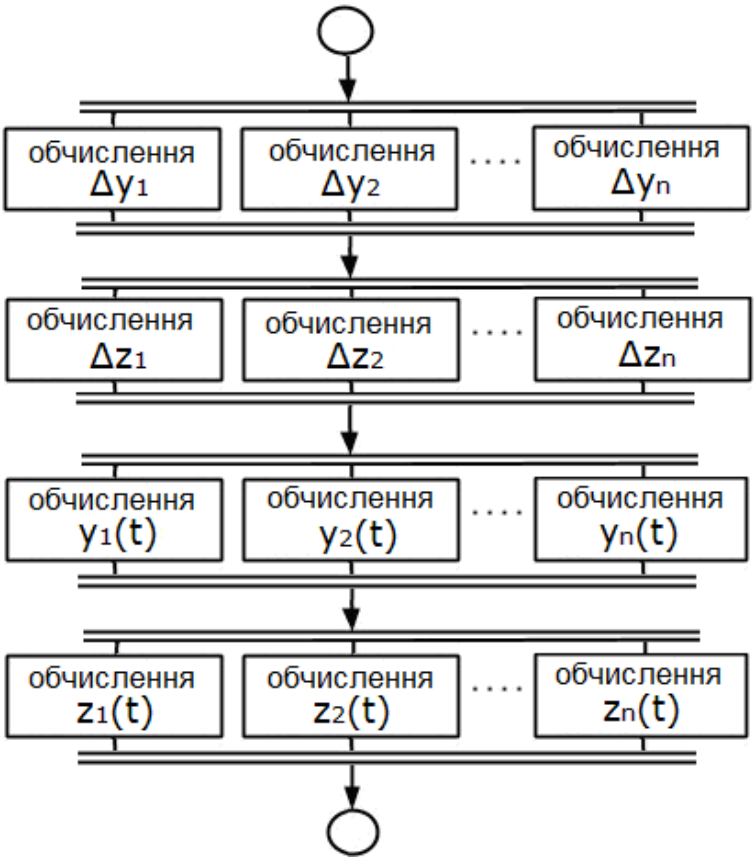

Рис. 6.5. Розрахунок рівнянь y_n, z_n в момент часу t методом Ейлера

Програмний код для алгоритму (рис. 6.5) поданий нижче. Обрані параметри розпаралелювання: кількість блоків – 1, число потоків в блоці дорівнює розміру масиву. При цьому в CUDA є обмеження до 1024 потоків на блок, тому якщо розмір масиву більше 1024, потрібно використовувати більше блоків.

```
__device__ double function2(int i, double *y, double *z, double *dt){
    return <результат обчислення> * dt;
}
__device__ double function3(int i, double *y, double *z, double *dt){
    return <результат обчислення> * dt;
}
__global__ void function1(double *dy, double *y, double *dz, double *z,
double *dt){
    dy[threadIdx.x] = function2(threadIdx.x, y, z, dt);
    dz[threadIdx.x] = function3(threadIdx.x, y, z, dt);
```

```
        y[threadIdx.x] = y[threadIdx.x] + dy[threadIdx.x];
        z[threadIdx.x] = z[threadIdx.x] + dz[threadIdx.x];
}
```

6.4. ПОЛІПШЕННЯ АЛГОРИТМУ ЯВНОГО МЕТОДУ ЕЙЛЕРА НА CUDA ШЛЯХОМ ОБ'ЄДНАННЯ СІТОК ПОТОКІВ

Обчислення прирощень $\Delta y_n, \Delta z_n$ для рівнянь y_n, z_n відбувається незалежно одне від одного, тому обчислення можна проводити одночасно і паралельно, при цьому кількість рівнянь y_n, z_n має бути однаковою (рис. 6.6). З точки зору моделі програмування CUDA таке поліпшення алгоритму є об'єднанням сіток блоків у більші сітки [125]. Паралелізм у цьому випадку полягає в деякій подобі використання безлічі обчислювальних операцій до безлічі даних (модель multiple instruction multiple data (MIMD)). Об'єднання сіток може бути застосовано за наявності однакової кількості рівнянь n, наприклад, $x_n, y_n ... z_n$, які можна обчислювати незалежно один від одного. Якщо кількість рівнянь n набагато більша, ніж кількість ядер GPU, то прискорення обчислень буде невеликим, тому що невикористання ядер GPU буде малим (за рахунок скорочення цього невикористання буде мале прискорення).

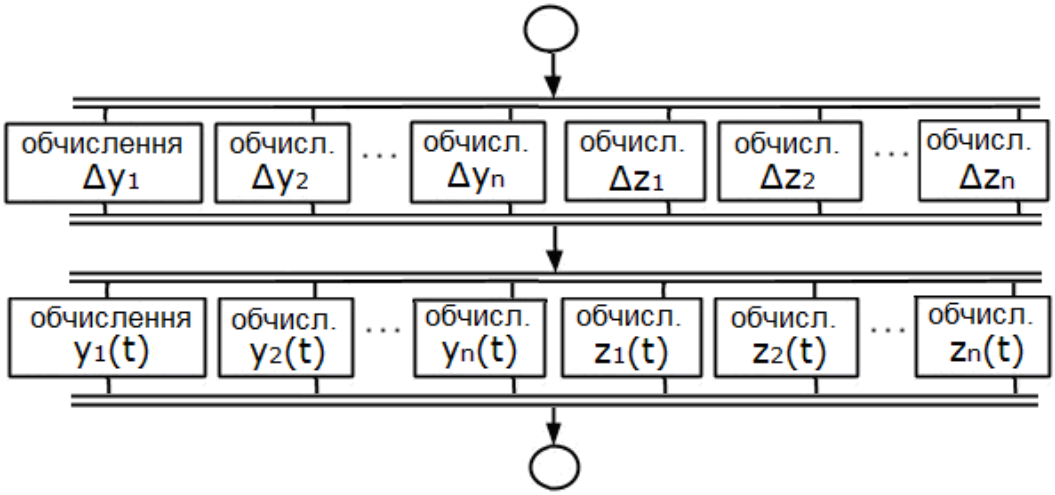

Рис. 6.6. Розрахунок рівнянь y_n, z_n у випадку об'єднання сіток

У випадку об'єднання сіток кожна сітка поміщується в блок [125]. Нехай буде по 150 рівнянь y_n, z_n, тобто $N = 150$. Тоді для одночасного обчислення $\Delta y_n, \Delta z_n$ створюється одна сітка з двох блоків (тому що є рівняння y_n, z_n і за допомогою індексу блоку можна отримати доступ до відповідного рівняння y_n або z_n) по 150 потоків. Спочатку на GPU запускається функція «function1», яка запускає обчислення $\Delta y_n, \Delta z_n$ (функцію «*function2*») за заданими параметрами розпаралелювання, а потім обчислення y_n, z_n (функцію «*function3*»). Всі потоки блоку

з індексом 0 будуть обчислювати рівняння Δy_n, потоки блоку з індексом 1 будуть обчислювати рівняння Δz_n.

```
__device__ void function2(int n, int block, double *dy, double *y, double *dz, double *z, double *dt){
        if(block == 0) dy[n] = <результат обчислення> * dt;
        else if(block == 1) dz[n] = <результат обчислення> * dt;
}
__device__ void function3(int n, int block, double *dy, double *y, double *dz, double *z, double *dt){
        if(block == 0) y[n] = y[n] + dy[n];
        else if(block == 1) z[n] = z[n] + dz[n];
}
__global__ void function1(double *dy, double *y, double *dz, double *z, double *dt){
        function2(threadIdx.x, blockIdx.x, dy, y, dz, z, dt);
        function3(threadIdx.x, blockIdx.x, dy, y, dz, z, dt);
}
```

Математичні операції під час обчислення y_n, z_n зазвичай набагато простіші і виконуються швидше, ніж під час обчислення $\Delta y_n, \Delta z_n$. Тому, щоб уникнути ускладнення коду, обчислення y_n, z_n можна робити без об'єднання сіток.

Об'єднання сіток потоків вирівнює навантаження на ядра GPU і збільшує швидкість обчислень [125]:

- без об'єднання сіток під час обчислення y_n створюється 150 потоків. З кількістю ядер GPU, яка дорівнює, наприклад, 384 (GeForce GT630 GV-N630D3-2GL) велика частина ядер залишається не використаною. Потім відбувається обчислення z_n з тією ж кількістю потоків;

- у випадку об'єднання сіток під час одночасного обчислення y_n, z_n створюється 300 потоків, тобто велика частина ядер GPU використовується. Таким чином, за одиницю часу можна обчислити рівняння y_n, z_n.

6.5. РЕАЛІЗАЦІЯ ЯВНОГО МЕТОДУ РУНГЕ–КУТТИ НА CUDA

Згідно з методом Рунге–Кутти, в кожен момент часу за допомогою кожного рівняння розраховується 4 наближення функції $y_{k1}, y_{k2}, y_{k3}, y_{k4}$, а потім розраховується значення функції $y(\mathrm{t}) = y(\mathrm{t}-\Delta t) + \frac{\Delta t}{6}(y_{k1} + 2\cdot y_{k2} + 2\cdot y_{k3} + y_{k4})$. Тобто приріст функції Δy в методі Ейлера відповідає приросту функції $\frac{1}{6}(y_{k1} + 2\cdot y_{k2} + 2\cdot y_{k3} + y_{k4})$ в методі

Рунге–Кутти. Нехай значення y_n залежить від y_n, z_n, тобто за методом Ейлера $\Delta y_n = f(y_n, z_n) \cdot \Delta t$. У методі Рунге–Кутти 1-ше наближення функції дорівнюватиме $y_{nk1} = f(y_n, z_n)$, 2-ге наближення залежить від 1-го (треба попередньо порахувати також z_{nk1}) і т. д.: $y_{nk2} = f(y_n + y_{nk1} \cdot \frac{\Delta t}{2}, z_n + z_{nk1} \cdot \frac{\Delta t}{2})$, $y_{nk3} = f(y_n + y_{nk2} \cdot \frac{\Delta t}{2}, z_n + z_{nk2} \cdot \frac{\Delta t}{2})$, $y_{nk4} = f(y_n + y_{nk3} \cdot \Delta t, z_n + z_{nk3} \cdot \Delta t)$ (рис. 6.7).

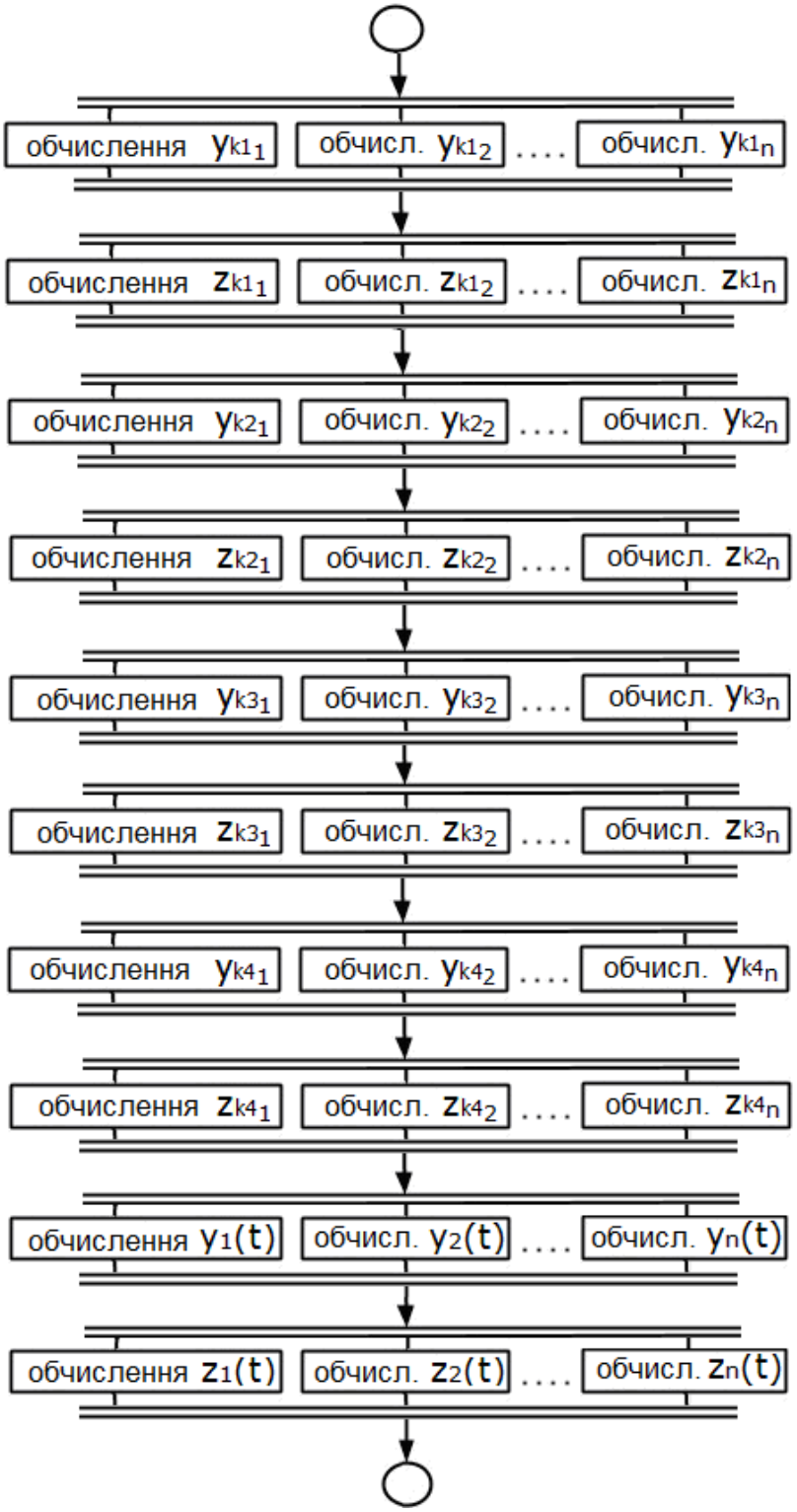

Рис. 6.7. Розрахунок рівнянь y_n, z_n в момент часу t методом Рунге–Кутти

Програмний код для алгоритму (рис. 6.7) представлений далі. Обрані параметри розпаралелювання: кількість блоків – 1, кількість потоків в блоці дорівнює розміру масиву. Для розрахунку $y_{k1}, y_{k2}, y_{k3}, y_{k4}$ можна використовувати одну і ту ж «__device__» функцію на CUDA (функцію «function2»). Тоді в якості параметрів у таку функцію передаються коефіцієнт dt і результат розрахунку попереднього наближення. Під час розрахунку наближення y_{k1} в якості коефіцієнта dt використовується 0.0, а в якості значень попереднього наближення використовується заповнений нулями допоміжний масив «zeroN» розміром N.

```
__device__ double function2(int n, double *y, double *z, double dt_koef, double *yk, double *zk){
    return <результат обчислення>;
}
__device__ double function3(int n, double *y, double *z, double dt_koef, double *yk, double *zk){
    return <результат обчислення>;
}
__global__ void function1(double *y, double *z, double *dt, double *yk1, double *zk1, double *yk2, double *zk2, double *yk3, double *zk3, double *yk4, double *zk4, double *zeroN){
    int n = threadIdx.x;
    yk1[n] = function2(n, y, z, 0.0, zeroN, zeroN);
    zk1[n] = function3(n, y, z, 0.0, zeroN, zeroN);
    yk2[n] = function2(n, y, z, dt/2.0, yk1, zk1);
    zk2[n] = function3(n, y, z, dt/2.0, yk1, zk1);
    yk3[n] = function2(n, y, z, dt/2.0, yk2, zk2);
    zk3[n] = function3(n, y, z, dt/2.0, yk2, zk2);
    yk4[n] = function2(n, y, z, dt, yk3, zk3);
    zk4[n] = function3(n, y, z, dt, yk3, zk3);
    y[n] = y[n] + (yk1[n] + 2.0*yk2[n] + 2.0*yk3[n] + yk4[n])*dt/6.0;
    z[n] = z[n] + (zk1[n] + 2.0*zk2[n] + 2.0*zk3[n] + zk4[n])*dt/6.0;
}
```

ВИСНОВКИ

Багато процесів у фізиці описуються диференційними рівняннями. Математичні моделі фізичних процесів в цій роботі є системами диференційних рівнянь. Кожне рівняння описує одну з характеристик фізичного процесу, при цьому частина рівняннь однотипні, тобто складаються з однакових обчислювальних операцій. Тому, відповідно до моделі SIMD, обчислювати однотипні рівняння можна паралельно.

Обчислювальний експеримент відбувався як розв’язання задачі Коші – встановлення значень параметрів у початковий момент часу, потім приріст часу і перерахунок усіх рівнянь, і т.д. багаторазово. В представленому алгоритмі показаний постійно повторюваний запуск обчислень на GPU з використанням двох циклів. У рамках вкладеного циклу обчислення відбуваються без отримання результатів із пам’яті GPU. Для підвищення точності обчислень зменшується крок за часом, але за малого кроку розвиток фізичного процесу відбувається повільно і тому результати обчислень на кожному кроці за часом отримувати не потрібно. Тільки результати одного з певної кількості кроків за часом зберігаються, тобто переносяться з пам’яті GPU в оперативну пам’ять.

Власні функції CUDA під час розв’язання задачі Коші реалізували один із чисельних методів розв’язання диференційних рівнянь. Була розглянута реалізація явного методу Ейлера 1-го порядку точності і явного методу Рунге–Кутти 4-го порядку точності.

Розглянуто поліпшення алгоритму розрахунку методом Ейлера шляхом об’єднання сіток блоків у більші сітки. Паралелізм у даному випадку полягав в деякій подобі використання безлічі обчислювальних операцій до безлічі даних.

СПИСОК ЛІТЕРАТУРИ

1. Lighthill M. J. Contribution to the theory of waves in nonlinear dispersive system / M. J. Lighthill // J. Inst. Math. Appl. – 1965. – V.1, № 2. – P. 269–306.

2. Zakharov V. E. Stability of nonlinear waves in dispersive media / V. E. Zakharov // J. Teor. Prikl. Fiz. – 1966. – V. 51. – P. 668–671.

3. Silin V. P. Parametric resonance in plasma / V. P. Silin // JETP. – 1965. – V. 48, № 6. – P. 1679–1691.

4. Aliev Ju. M. Oscillations theory of plasma, which situated in HF electromagnetic field / Ju. M. Aliev, V. P. Silin // JETP. – 1965. – V. 48, № 3. – P. 901–912.

5. Gorbunov L. M. On the plasma instability in strong HF field / L. M. Gorbunov, V. P. Silin // JETP. – 1965. – V. 49, № 6. – P. 1973–1981.

6. Benjamin T. B. The disintegration of wave trains on deep water / T. B. Benjamin, J. E. Feir // J. Fluid Mech. – 1967. – P. 417–430.

7. Васильев В. А. Автоволновые процессы / В. А. Васильев, Ю. М. Романовский, В. Г. Яхно ; под ред. Д. С.Чернавского. – М. : Наука. Гл. ред. физ.-мат. лит. 1987. – С. 240.

8. Автоволновые процессы в системах с диффузией / под ред. М. Т. Греховой. – Горький : ИПФ АН СССР, 1981. – С. 246.

9. Теория солитонов: метод обратной задачи / под ред. С. П. Новикова. – М. : Наука, 1980. – 327 с.

10. Групповой анализ дифференциальных уравнений / Л. В. Овсянников. – М. : Наука, 1978. – 240 с.

11. Chernousenko V. M. Space dissipative Structures / V. M. Chernousenko, V. M. Kuklin, I. P. Panchenko, V. M. Vorob'ev // Nonlinear World. – 1990. – V. 2 (IV International Workshop on Nonlinear and Turbulent Processes in Physics, 1989, Singapore). – P. 776–803.

12. Куклін В. М. Роль поглинання та дисипації енергії у формуванні просторових нелінійних структур у нерівноважних середовищах / В. М. Куклін // УФЖ. Огляди. – 2004. – Т. 1, № 1. – С. 49–81.

13. Kuklin V. M. Effect of induced interference and the formation of spatial perturbation fine structure in nonequlibrium open-ended system / V. M. Kuklin // Вопросы атомной науки и техники (ВАНТ). Сер. : Плазменная электроника и новые методы ускорения. – 2006. – № 5 (5). – С. 63–68.

14. Suhl H. Effective Nuclear Spin Interactions in Ferromagnets / H. Suhl // Phys Rev. – 1958. – V.109, № 2. – P. 606.

15. Schlomann E. B. Ferromagnetic Resonance Experiments at High Peak Power Levels / E. Schlomann, J. H. Saunder, M. H. Sirvets // J. Appl. Phys. – 1960. – V. 31, Suppl. – P. 386–395.

16. Захаров В. Е. Новый механизм ограничения амплитуды СВ при параллельной накачке / В. Е. Захаров, В. С. Львов, С. С. Старобинец // ФТТ. – 1969. – Т. 11. – С. 2047–2055.

17. Захаров В. Е. Турбулентность спиновых волн за порогом их параметрического возбуждения / В. Е. Захаров, В. С. Львов, С. С. Старобинец // УФН. – 1974. – Т. 114, № 4. – С. 609–654.

18. Львов В. С. Нелинейные спиновые волны / В. С. Львов. – М. : Наука. Гл. ред. физ.-мат. лит., 1987. – 272 с.

19. Белкин Е. В. Об интерференции в многомодовых режимах модуляционных неустойчивостей / Е. В. Белкин, А. В. Киричок, В. М. Куклин // Вопросы атомной науки и техники (ВАНТ). Сер. : Плазменная электроника и новые методы ускорения. – 2008. – № 4 (6). – С. 222–227.

20. Belkin E. V. The mathematical models of the modulation instability processes of waves in media with cubic nonlinearity : manuscript, PhD-thesis by speciality 01.05.02 – mathematical modeling and computational methods / E. V. Belkin. – Kharkiv. : V. N. Karasin's Kharkiv National University. : 2010. – 150 p.

21. Белкин Е. В. Модуляционная неустойчивость волн, поддерживаемых внешним источником в среде с поглощением / Е. В. Белкин, А. В. Киричок, В. М. Куклин // ВАНТ. Сер. : Плазменная электроника и новые методы ускорения. – 2010. – № 4(68). – С. 291–295.

22. Belkin E. V. Development of modulation instabilities in media with damping and forcing / E. V. Belkin, A. V. Kirichok, V. M. Kuklin // Мощная импульсная электрофизика. Международная конференция «XIV Харитоновские тематические научные чтения». International conference XIV Khariton's topical scientific readings. Сборник докладов. Digest of technical papers. – Саров : РФЯЦ-ВНИИЭФ, 2013. – 534 с., 7 с.ил.

23. Белкин Е. В. О верификации S-теории, используемой для описания модуляционных неустойчивостей волнового поля / Е. В. Белкин, А. В. Киричок, В. М. Куклин, А. В. Приймак // Вестник Харьковского национального университета им. В. Н. Каразина. – 2014. – № 1105. Сер. : Математическое моделирование. Информационные технологии. Автоматизированные системы управления. – Вып. 24. – С. 5–20.

24. Belkin E. V. Abnormal waves in wave field with modulation instability / E. V. Belkin, A. V. Kirichok, V. M. Kuklin, A. V. Pryimak // East European Journal of Physics. – 2014. – V.1, № 2. – P. 4–39.

25. Карпман В. И. Нелинейные волны в диспергирующих средах / В. И. Карпман. – М. : Наука, 1973. – 175 с.

26. Zakharov V. E. Statistical theory of gravity and capillary waves on the surface of a finite-depth fluid. Three-dimensional aspects of air-sea interaction / V. E. Zakharov // Eur. J. Mech. B Fluids. – 1999. – V.18(3). – P. 327–344.

27. Schwartz L. W. Strongly nonlinear waves / L. W. Schwartz, J. D. Fenton // Ann. Rev. Fluid. Mech. – 1982. – V.14. – P. 39–60.

28. Nonlinear Water Waves / L. Debnath // Academic Press, Boston. – 1994.

29. Куркин А. А. Волны-убийцы: факты, теория и моделирование / А. А. Куркин, Е. Н. Пелиновский. – Нижний Новгород : ННГУ, 2004.

30. Dyachenko A. I. Modulation instability of stokes wave – freak wave / A. I. Dyachenko, V. E. Zakharov // JETPLett. – 2005. – V. 81(6). – P. 255–259.

31. Kharif C. Physical mechanisms of the rogue wave phenomenon / C. Kharif, E. Pelinovsky // Eur. J. Mech. B-Fluid. – 2006. – V. 22(6) . – P. 603–633.

32. Yeom D.-I. Photonics: rogue waves surface in light / D.-I. Yeom, B. J. Eggleton // Nature. – 2007. – V. 450. – P. 953–962.

33. Solli D. R. Optical rogue waves / D. R. Solli, C. Ropers, P. Koonath, B. Jalali // Nature. – 2007. – V. 450. – P. 1054–1064.

34. Burlaga L. F. Linear magnetic holes in a unipolar region of the heliosheath observed by Voyager 1 / L. F. Burlaga, N. F. Ness, M. H. Acuna // J. Geophys. Res., 112, A07106. – 2007.

35. Kharif C. Rogue Waves in the Ocean / C. Kharif, E. Pelinovsky, A. Slunyaev // Springer-Verlag, Berlin, Heidelberg. – 2009. – P. 1–10.

36. Ruderman M. S. Freak waves in laboratory and space plasmas / M. S. Ruderman // Eur. Phys. J. Special Topics. – 2010. – V. 185. – P. 57–66.

37. Бадулин С. Влияние гигантских волн на безопасность морской добычи и транспортировки углеводородов / С. Бадулин, А. Иванов, А. Островский // Технологии, ТЭК. – 2005. – № 2. – С. 56–62.

38. Stansell P. Distributions of extreme wave, crest and trough heights measured in the North Sea / P. Stansell // Ocean Engineering. – 2005. – V. 32, № 8–9. – P. 1015–1036.

39. Chen P. Acceleration of electrons by the interaction of a bunched electron beam with plasma / P. Chen, J. M. Dawson, R. W. Huff, T. Katsouleas // Phys Rev. Lett. – 1985. – V. 54. – P. 693–696.

40. Katsouleas T. Physical Mechanisms in the plasma wake-field accelerator / T. Katsouleas // Phys. Rev. A. – 1986. – V. 33. – P. 2056–2064.

41. Bennet W. H. Magnetically self-focusing streams / W. H. Bennet // Phys. Rev. – 1934. – V.45. – P. 890–920.

42. Budker G. I. Relativistic stabilized electron beam / G. I. Budker // Proc. CERN Symp. of high energy accelerators. Geneva. – 1956. – V. 1. – P. 68–73.

43. Davidson R. C. Theory of Nonneutral Plasmas / R. C. Davidson // W. A. Benjamin Inc., Advanced Book Program Reading. Massachusets. – 1974. – V. 43. – 213 p.

44. Иванов А. А. Физика сильнонеравновесной плазмы / А. А. Иванов. – М. : Атомиздат, 1977.

45. Рухадзе А. А. Физика сильноточных релятивистких пучков / А. А. Рухадзе, Л. С. Богданкевич, С. Е. Росинский, В. Г. Рухлин. – М. : Атомиздат, 1980.

46. Fainberg Ya. B. Particle accelerators / Ya. B. Fainberg // Proc. VII Internat. Conference on Accelerators, Erevan. – 1969.

47. Красовицкий В. Б. Нелинейная радиальная самофокусировка электронного пучка в плазме / В. Б. Красовицкий // Письма в ЖЭТФ. – 1969. – № 9. – С. 679–684.

48. Дорофеенко В. Г. Самофокусировка модулированного электронного пучка в плазме / В. Г. Дорофеенко, В. Б. Красовицкий // Укр. Физ. Журнал. – 1984. – Т. 29, № 3. – С. 395–405.

49. Гладкий А. М. Свойства плазменных волн, возбуждаемых электронными сгустками / А. М. Гладкий, В. П. Коваленко, П. Н. Юсманов // Письма в ЖЭТФ. – 1976. – № 24. – С. 533–542.

50. Self-focusing of relativistic electron bunches in a plasma / V. B. Krasovitsky. – Kharkiv : Folio, 2000. – 196 p.

51. Батищев О. В. Самофокусировка ленточного РЭП в плотной плазме / О. В. Батищев, В. Б. Красовицкий, Ю. С. Сигов и др. // Физика плазмы. – 1993. – Т. 19. – С. 738–743.

52. Batishchev O. V. 2.5 Dimentional computer simulation of relativistic bunch propagation in tenuous and dence plasmas / O. V. Batishchev, V. I. Karas', Yu. S. Sigov, Ya. B. Fainberg // Plasma Physics Reports. – 1994. – V. 20. – P. 583–586.

53. Балакирев В. А. Модуляция релятивистких элетронных сгустков в плазме / В. А. Балакирев, Г. В. Сотников, Я. Б. Файнберг // Физика плазмы. – 1996. – Т. 22, № 2. – С. 165–169.

54. Balakirev V. A. Charged particle (CP) acceleration by an intense wakefield (WF) excited in plasma by either laser pulse (LP) or relativistic electron bunch (REB) / V. A. Balakirev, I. V. Karas', V. I. Karas' at al. // Вопросы атомной науки и техники. Сер. : Плазменная электроника и новые методы ускорения (3). – 2003. – № 4. – С. 29–32.

55. Onishenko N. I. Theoretical studies of the resonator concept of dielectric wakefield accelerator / N. I. Onishenko, G. V. Sotnikov // Вопросы атомной науки и техники. Сер. : Плазменная электроника и новые методы ускорения (3). – 2006. – № 5. – С. 203–207.

56. Файнберг Я. Б. Ускорение заряженных частиц в плазме / Я. Б. Файнберг // УФН. – 1967. – Т. 93. – С. 617–628.

57. Файнберг Я. Б. Некоторые вопросы плазменной электроники. Физика плазмы / Я. Б. Файнберг. – 1985. – Т. 11, вып.11. – С. 1398–1410.

58. Рабинович М. С. Принципы релятивистской плазменной электроники / М. С. Рабинович, А. А. Рухадзе // Физика плазмы. – 1976. – Т. 2., вып. 5. – С. 715–722.

59. Шапиро В. Д. Взаимодействие волна-частица в неравновесных средах / В. Д. Шапиро, В. И. Шевченко // Изв. ВУЗов. Радиофизика. – 1976. – Т. 19, № 5–6. – С. 787–791.

60. Гришин В. К. Устойчивость заряженного пучка малой длительности в плазменном волноводе / В. К. Гришин, Е. Н. Шапошникова // Физика плазмы. – 1982. – Т. 8, вып. 2. – С. 287–292.

61. Кондратенко А. Н. Эволюция сгустка заряженных частиц в поле собственного излучения / А. Н. Кондратенко, В. М. Куклин, Н. С. Репалов // Укр. физ. журнал. – 1982. – Т. 27, № 8. – С. 1159–1164.

62. Куклин В. М. Одномерные движущиеся сгустки заряженных частиц в плазме / В. М. Куклин // Укр. физ. журнал. – 1986. – Т. 31, № 6. – С. 853–857.

63. Куклин В. М. Роль поглощения и диссипации энергии в формировании пространственных нелинейных структур в неравновесных средах / В. М. Куклин // УФЖ. Обзоры. – 2004. – Т. 1, № 1. – С. 49–81.

64. Абрамович В. У. К нелинейной теории диссипативной пучковой неустойчивости релятивистского пучка в плазме / В. У. Абрамович, В. И. Шевченко // ЖЭТФ. – 1972. – Т. 62, вып. 4. – С. 1386–1391.

65. Кондратенко А. Н. Основы плазменной электроники / А. Н. Кондратенко, В. М. Куклин. – М. : Энергоатомиздат, 1988. – 320 с.

66. Киричок А. В. Об особенностях излучения движущихся одиночных электроннных сгустков в плазме / А. В. Киричок, В. М. Куклин, А. В. Мишин // ВАНТ. Сер. : Плазменная электроника и новые методы ускорения. – 2010. – № 4(68). – С. 58–61.

67. Куклин В. М. Многоволновые процессы в плазме / В. М. Куклин, И. П. Панченко, Ф. Х. Хакимов. – Душанбе : Дониш, 1999. – 175 с.

68. Kuklin V. M. 3-D short Beam Dynamics / V. M. Kuklin, S. S. Moiseev, I. P. Panchenko // Moscow. Reprint of Institute of Space Research. – 1989. – № 1619. – 11 p.

69. Kuklin V. M. Nonlinear structure formation in dissipative media / V. M. Kuklin, I. P. Panchenko // Plasma Physics Reports. – 1994. – 20, № 9. – P. 813–823.

70. Альтеркоп Б. А. Двумерная динамика компенсированного электронного сгустка в плотной плазме / Б. А. Альтеркоп, С. Р. Жексембин, В. Г. Рухлин, В. П. Тараканов // Препринт Ин-та Высоких температур АН СССР. – 1986. – № 6–193. – С. 35–45.

71. Корн Г. Справочник по математике для научных работников и инженеров / Г. Корн, Т. Корн ; под ред. И. Г. Арамановича. – М. : Наука, 1974. – 832 с.

72. Куклин В. М. О процессах излучения в неравновесных средах / В. М. Куклин // Вісник ХНУ ім. В. Н. Каразіна. Сер. : Ядра, частинки, поля. – 2010. № 933, вып. 4 (48). – С. 4–27.

73. Балакирев В. А. Теория черенковских усилителей и генераторов на релятивистских пучках / В. А. Балакирев, Н. И. Карбушев, А. О. Островский, Ю. В. Ткач. – К. : Наукова думка, 1993. – 208 с.

74. Busse F. H. Nonlinear convection in a layer with nearly insulating boundaries / F. H. Busse, N. Riahi // J. Fluid Mech. – 1980. – № 96. – P. 243–256.

75. Chandrasekhar S. Hydrodynamic and Hydromagnetic Stability / S. Chandrasekhar. – Oxford : Clarendon Press, 1970.

76. Getling A. V. Rayleigh–Bénard Convection: Structures and Dynamics / A. V. Getling // World Scientific. – 1998. – P. 59.

77. Cross M. C. Pattern formation outside of equilibrium / M. C. Cross, P. C. Hohenberg // Rev. Mod. Phys. 65. – 1993. – P. 851.

78. Karpman V. I. Nonlinear Waves in Dispersive Media / V. I. Karpman. – New York : Pergamon, 1975. – 185 p.

79. Engelbrecht J. K. Evolution equations and self-wave processes in the active medium / J. K. Engelbrecht // On nonlinear continuum mechanics. – Tallinn : Valgus, 1985. – P. 119–131.

80. Schwartz L. W. Strongly nonlinear waves / L. W. Schwartz, J. D. Fenton // Ann. Rev. Fluid. Mech. – 1982. – № 14. – P. 39–60.

81. Davydov A. S. Solitons in molecular systems / A. S. Davydov. – Kyiv : Naukova Dumka, 1984. – V. 1. – P. 731.

82. Yanovsky V. V. Lectures on nonlinear phenomena / V. V. Yanovsky // Kharkov Institute of Single Crystals. – 2007. – V. 2.

83. Zaslavsky G. M. Weak Chaos and Quasi-Regular Patterns / G. M. Zaslavsky, R. Z. Sagdeev, D. A. Usikov, A. A. Chernikov. – Cambridge : Cambridge University Press, 1991. – 257 p.

84. Kuklin V. M. The role of absorption and dissipation of energy in the formation of spatial structures in nonlinear nonequilibrium media / V. M. Kuklin // Ukr. J. Phys. Rev. 1. – 2004. – P. 49–81.

85. Nikolis G. Self-organization in nonequilibrium systems / G. Nikolis, I. Prigogine. – New York : Wiley, 1977. – 491 p.

86. Haken H. Synergetics: An Introduction / H. Haken. – Berlin : Springer-Verlag, 1978. – 412 p.

87. Tsytovich V. N. Nonlinear Effects in Plasma / V. N. Tsytovich. – New York : Plenum Press, 1970. – 350 p.

88. Zakharov V. E. Solitons and collapses: two evolution scenarios of nonlinear wave systems / V. E. Zakharov, E. A. Kuznetsov // Phys. Usp. – 2012. – № 55. – P. 535–556.

89. Petviashvili V. I. Solitary Waves in the Plasma and the Atmosphere / V. I. Petviashvili, O. A. Pokhotelov. – Moscow : Energoatomizdat, 1989. – 262 p.

90. Klimontovich L. Statistic Theory of Plasma / L. Klimontovich, H. Vilhelmson, I. P. Yakimenko // Molecular Systems. – Moscow : Mosc. University Press, 1990. – P. 263–401.

91. L'vov V. S. Wave Turbulence under Parametric Excitations. Applications to Magnetics / V. S. L'vov. – Berlin : Springer-Verlag. 1994. – 330 p.

92. Kuklin V. M. Multiwave processes in plasma physics / V. M. Kuklin, I. P. Panchenko, F. H. Khakimov. – Dushanbe : Donish, 1989.

93. Buts V. A. The Theory of Coherent Radiation by Intense Electron Beams / V. A. Buts, A. N. Lebedev, V. I. Kurilko. – Berlin; Heidelberg : Springer-Verlag, 2006. – 263 p.

94. Moiseev S. S. Vortex Dynamo in a Convective Medium with Helical Turbulence / S. S. Moiseev, P. B. Rutkevich, A. V. Tur, V. V. Yanovsky // JETP. – 1988. – V. 94(2). – P. 144–153.

95. Horsthemke W. Noise-Induced Transitions / W. Horsthemke, R. Lefever // Springer Series in Synergetics 15. – New York : Springer-Verlag, 1984. – P. 164–200.

96. Kirichok A. V. Allocated Imperfections of Developed Convective Structures / A. V. Kirichok, V. M. Kuklin // Phys. and Chem. of the Earth. – 1999. – Part A, 6. – P. 533–538.

97. Belkin E. V. Structure transitions in the Proctor–Sivashinsky model / E. V. Belkin, I. V. Gushchin, A. V. Kirichok, V. M. Kuklin // VANT. Series : Plasma electronics and new methods of acceleration. – 2010. – № 4. – P. 296–298.

98. Chapman J. Nonlinear Rayleigh-Benard convection between poorly conducting boundaries / J. Chapman, M. Proctor // J. Fluid Mech. – 1980. – № 101. – P. 759–765.

99. Gertsberg V. Large cells in nonlinear Rayleigh-Benard convection / V. Gertsberg, G. E. Sivashinsky // Prog. Theor. Phys. – 1981. – № 66. – P. 1219–1229.

100. Malomed B. A. Two-dimensional quasi-periodic structures in nonequilibrium systems / B. A. Malomed, A. A. Nepomniachtchi, M. P. Tribel'skii // JETP. – 1989. – V. 69. – P. 388–396.

101. Pismen L. Inertial effects in long-scale thermal convection / L. Pismen // Phys. Lett. – 1986. – № 116. – P. 241–243.

102. Kirichok A. V. Dynamics of large-scale vortices in the mode of convective instability / A. V. Kirichok, V. M. Kuklin, I. P. Panchenko at al. // Physics in Ukraine, Proc. of Int. Conference. – Kyiv : Inst. of Theor. Phys, 1993. – P. 76–80.

103. Kirichok A. V. On the possibility of dynamo mechanism in non-equilibrium convective environment / A. V. Kirichok, V. M. Kuklin, I. P. Panchenko // Dokl. Ukr. Acad. Nauk. – 1997. – № 4. – P. 87–92.

104. Kuklin V. M. Effect of induced interference and the formation of spatial perturbation fine structure in nonequlibrium open-ended system / V. M. Kuklin // Problems of Atomic Science and Technology, Ser. Plasma Electronics and New Methods of Acceleration. – 2006. – № 5(5). – P. 63–68.

105. Gushchin I. V. Pattern formation in convective media / I. V. Gushchin, A. V. Kirichok, V. M. Kuklin // Kharkiv Nat. Un., Phys. Ser. : Nuclei, Particles, Fields. – 2013. – № 1040. – P. 4–27.

106. CUDA C Programming Guide. – Nvidia Corporation [Electronic resource]. – Way of access : http://docs.nvidia.com/cuda/cuda-c-programming-guide/. – Electronic version, 2015. – HTML format.

107. CUDA. – Wikipedia [Electronic resource]. – Way of access : http://en.wikipedia.org/wiki/CUDA. – Electronic version, 2014. – HTML format.

108. Что такое CUDA? – Nvidia Corporation [Электронный ресурс]. – Режим доступа : http://www.nvidia.ru/object/what_is_cuda_new_ru.html. – Электрон. версія, 2014. – HTML формат.

109. Karimi K. A Performance Comparison of CUDA and OpenCL [Electronic resource] / K. Karimi, N. G. Dickson, F. Hamze. – arxiv.org. – Way of access : http://arxiv.org/ftp/arxiv/papers/1005/1005.2581.pdf – Electronic version, 2015. – PDF format.

110. Fang J. A Comprehensive Performance Comparison of CUDA and OpenCL [Electronic resource] / J. Fang, A. Varbanescu, H. Sips. – pds.ewi.tudelft.nl. – Way of access : http://pds.ewi.tudelft.nl/pubs/papers/ icpp2011a.pdf – Electronic version, 2015. – PDF format.

111. Rosendahl S. CUDA and OpenCL API comparison [Electronic resource] / S. Rosendahl. – wiki.aalto.fi. – Way of access : https://wiki.aalto.fi/download/ attachments/40025977/Cuda+and+OpenCL+API+comparison_presented.pdf – Electronic version, 2015. – PDF format.

112. Betz S. Comparison OpenCL and CUDA [Electronic resource] / S. Betz. – lrr.in.tum.de. – Way of access : http://www.lrr.in.tum.de/~gerndt/ home/Teaching/ ComputerArchitecture/StudentPresentations/CudaOpenCL.pdf – Electronic version, 2015. – PDF format.

113. Сандерс Д. Технология CUDA в примерах. Введение в программирование графических процессов / Д. Сандерс, Э. Кэндрот. ; пер. с англ. А. А. Слинкина ; науч. ред. А. В. Боресков. – М. : ДМК Пресс, 2011. – 232 с.

114. CUDA. – Wikipedia [Электронный ресурс]. – Режим доступа : http://ru.wikipedia.org/wiki/CUDA. – Электрон. версия, 2014. – HTML формат.

115. CUDA мы катимся: технология Nvidia CUDA. – Хакер [Электронный ресурс]. – Режим доступа : http://www.xakep.ru/post/47507/ default.asp. – Электрон. версия, 2014. – HTML формат.

116. Берилло А. Nvidia CUDA – неграфические вычисления на графических процессорах [Электронный ресурс] / А. Берилло. – ixbt.com. – Режим доступа : http://www.ixbt.com/video3/cuda-1.shtml. – Электрон. версия, 2014. – HTML формат.

117. Nvidia CUDA 5 еще больше упрощает программирование на GPU. – Nvidia Corporation [Электронный ресурс]. – Режим доступа : http://www. nvidia.ru/ object/nvidia-cuda-5-parallel-computing-20121015-ru.html. – Электрон. версия, 2014. – HTML формат.

118. Шиллинг А. GTC 2012: что скрывается за Hyper-Q и Dynamic Parallelism [Электронный ресурс] / А. Шиллинг. – hardwareluxx.ru. – Режим доступа : http://www.hardwareluxx.ru/index.php/news/hardware/ grafikkarten/ 22085-gtc-2012-hyper-q-dynamic-parallism.html. – Электрон. версия, 2015. – HTML формат.

119. Тестов А. NVIDIA планирует ускорить разработку графических архитектур [Электронный ресурс] / А. Тестов. – 3dnews. – Режим доступа : http://www.3dnews.ru/907507. – Электрон. версия, 2015. – HTML формат.

120. JCUDA / M. Hutter. – jcuda.org [Electronic resource]. – Way of access : http://jcuda.org/. – Electronic version, 2015. – HTML format.

121. Приймак А. В. Оптимизация вычислений на CUDA при моделировании неустойчивости ленгмюровских волн в плазме / А. В. Приймак // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. пр. – К. : Век+, 2013. – № 58. – С. 125–130.

122. Боресков А. В. Основы работы с технологией CUDA / А. В. Боресков, А. А. Харламов. – М. : ДМК Пресс, 2010. – 232 с.

123. CUDA C Best Practices Guide. – Nvidia Corporation [Electronic resource]. – Way of access : http://docs.nvidia.com/cuda/cuda-c-best-practices-guide. – Electronic version, 2013. – HTML format.

124. Цветков И. В. Применение численных методов для моделирования процессов в плазме : учебное пособие / И. В. Цветков. – М. : МИФИ, 2007. – 84 с.

125. Приймак А. В. Оптимизация вычислений на CUDA путем объединения сеток потоков в моделях взаимодействия пучка и плазмы / А. В. Приймак // Материалы XIII Международной научно-практической интернет-конференции «Проблемы и перспективы развития науки в начале третьего тысячелетия в странах СНГ» (29–30 апреля 2015 года, Переяслав-Хмельницкий) : сб. научных трудов / Переяслав-Хмельницкий ГПУ им. Г. Сковороди. – Переяслав-Хмельницкий, 2015. – С. 326–329.

ДОДАТКИ

Додаток А

ВІДПРАЦЮВАННЯ НАЛАШТУВАННЯ ІНТЕРВАЛУ ОЧІКУВАННЯ GPU ДЛЯ ОПЕРАЦІЙНОЇ СИСТЕМИ

Для зручності читача, який вирішив перевірити наведені в цій монографії приклади на своєму комп'ютері, наведемо в додатках А–В наше відпрацювання способів установки і налаштування CUDA.

Перевищення інтервалу очікування GPU веде до помилки CUDA_ ERROR_LAUNCH_TIMEOUT. Ця помилка означає, що GPU не може проводити розрахунки довше встановленого інтервалу очікування без відповіді в основну частину CUDA-програми. Наприклад, інтервал очікування для ОС WindowsXP дорівнює 20 с, для Windows7 – 5 с. Для вирішення проблеми необхідно відкрити редактор реєстру «Пуск» -> «Виконати» («regedit»). У редакторі реєстру знайти папку «Graphics Drivers» і додати в неї і в папку «DCI» (вкладену в «GraphicsDrivers») параметр «TdrLevel» зі значенням 0 (рис. А.1).

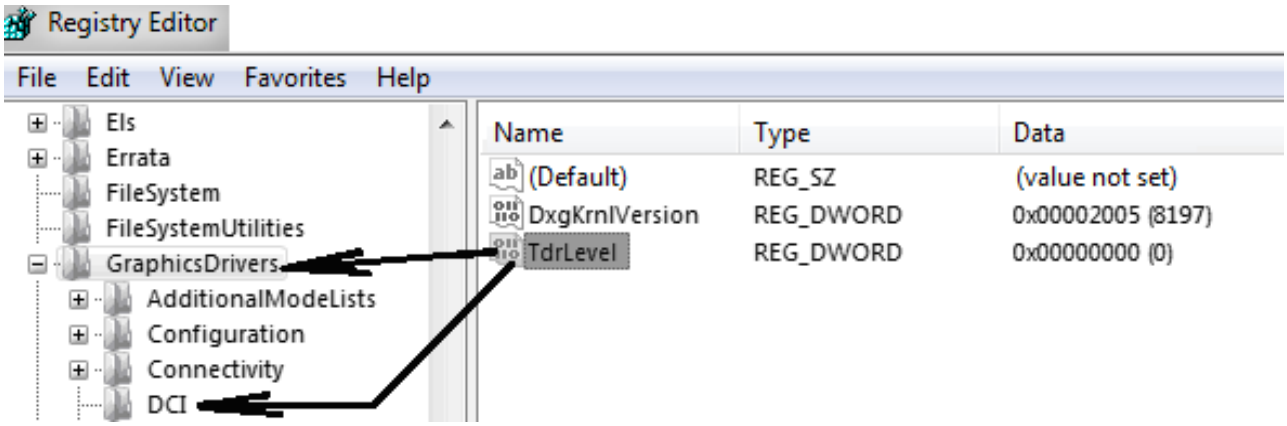

Рис. А.1. Додавання параметра «TdrLevel» в реєстр Windows

Додаток Б

ВІДПРАЦЮВАННЯ НАЛАШТУВАННЯ CUDA ДЛЯ СЕРЕДОВИЩА РОЗРОБКИ VISUAL STUDIO EXPRESS 2012 У WINDOWS X32

1. Для програмування на CUDA необхідно встановити CUDA Toolkit. Завантажити його можна, наприклад, за посиланням https://developer.nvidia.com/cuda-downloads

В цьому додатку описується використання CUDA Toolkit версії 6.5.

2. Після установки CUDA Toolkit в середовищі розробки Visual Studio необхідно створити порожній «Visual C ++ Win32 Project» через меню «FILE» – «NEW PROJECT» (рис. Б.1). Вибрати «WIN32 Console Application». Натиснути «ОК», а в наступному вікні натиснути «Finish».

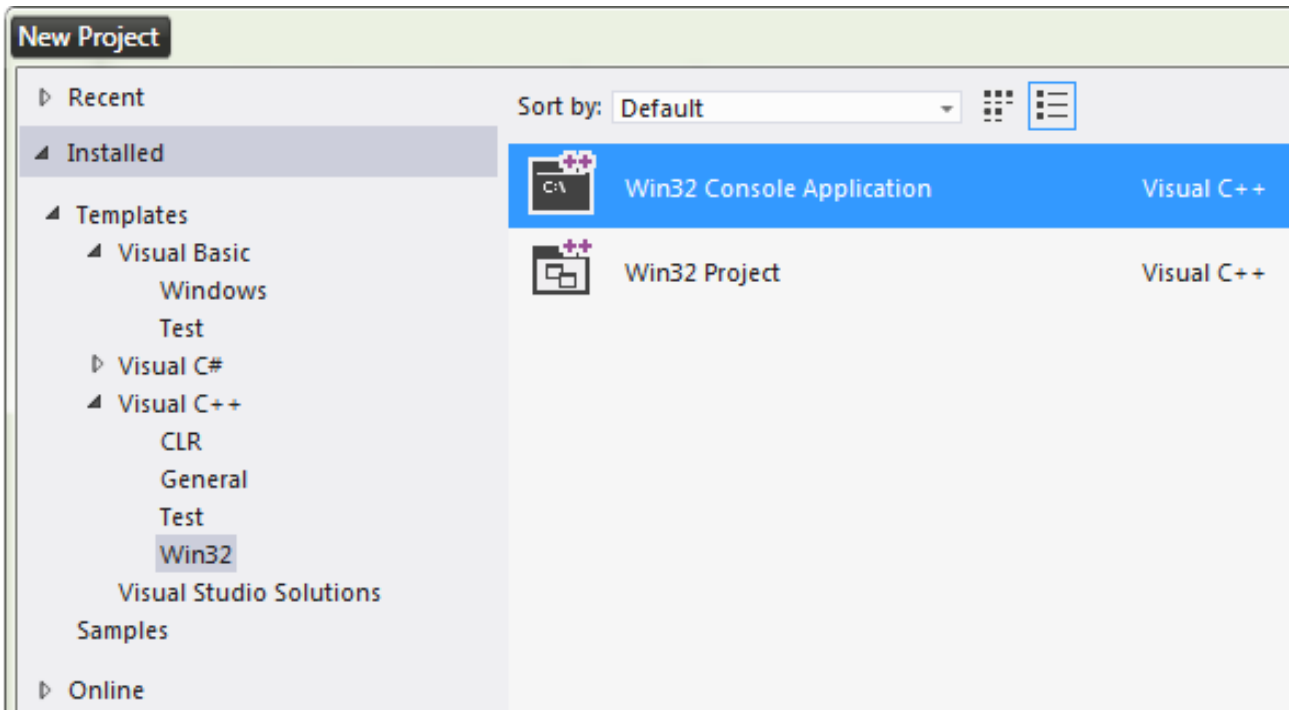

Рис. Б.1. Створення проекту в Visual Studio

3. В оглядачі рішень знайти .cpp файл з іменем проекту (у цьому випадку ConsoleApplication1) (рис. Б.2).

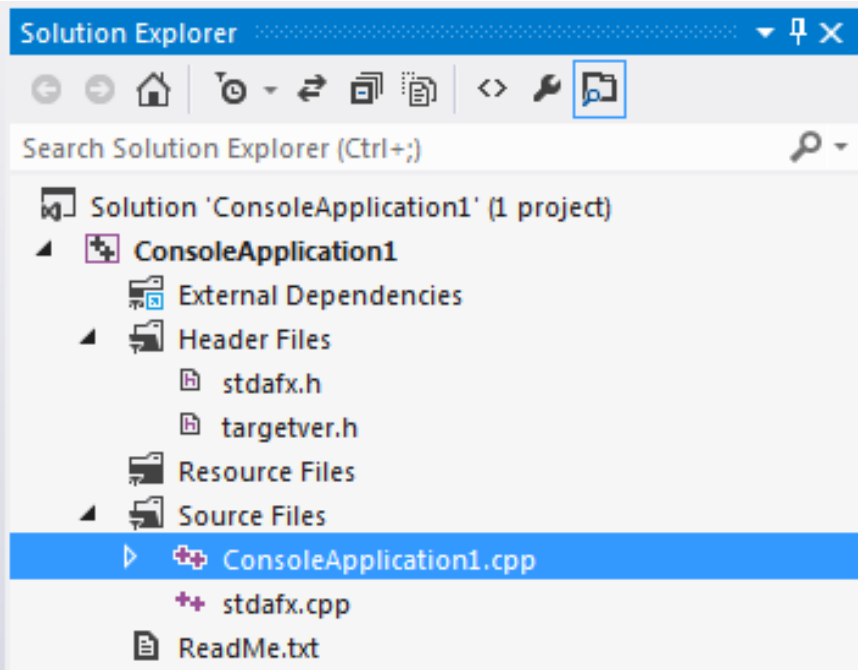

Рис. Б.2. .cpp-файл проекту

Натиснути на файлі правою кнопкою і в меню, що випадає, натиснути «Rename». Змінити розширення файлу .cpp на .cu (рис. Б.3).

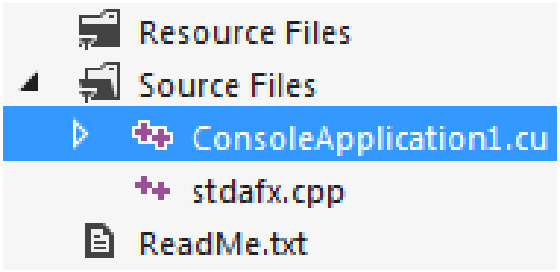

Рис. Б.3. Зміна розширення файлу .cpp на .cu

4. Знайти в установленому CUDA Toolkit наступні файли (рис. Б.4).

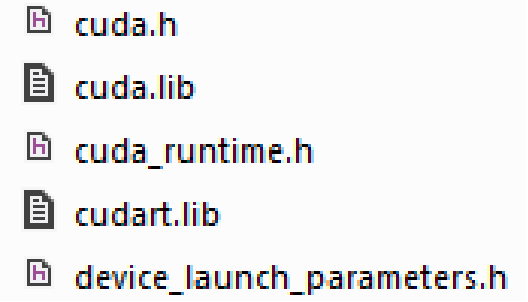

Рис. Б.4. Необхідні файли CUDA

Файли потрібно скопіювати в проект у папку ConsoleApplication1 (рис. Б.5).

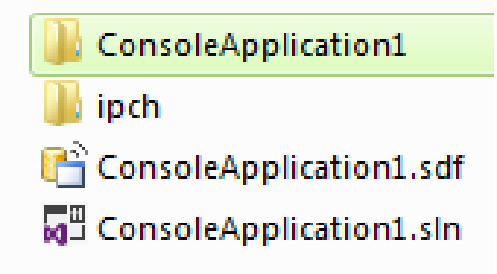

Рис. Б.5. Папка для файлів CUDA

У браузері рішень натиснути правою кнопкою на «Header Files» – «Add» – «Existing Item» (рис. Б.6).

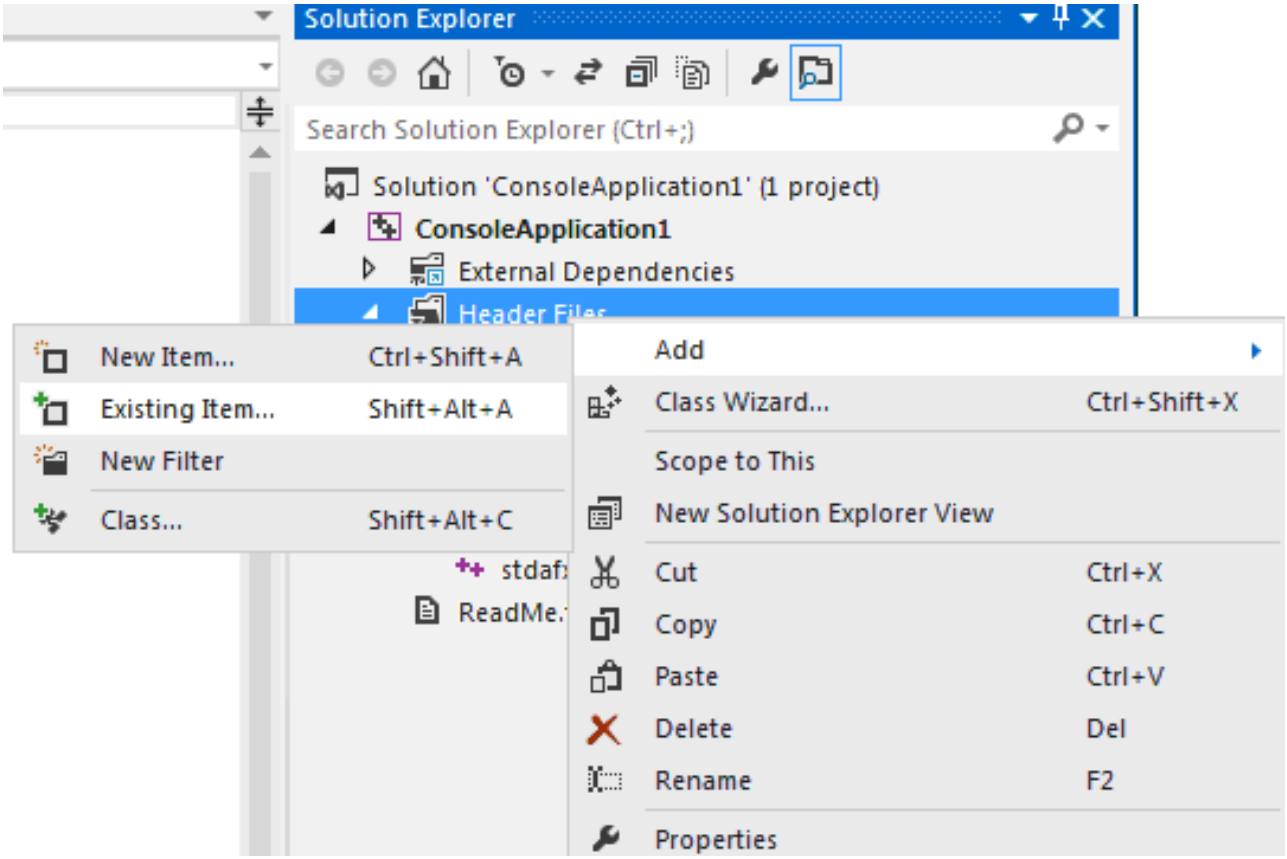

Рис. Б.6. Меню для додавання файлів CUDA

У вікні додати файли (рис. Б.7).

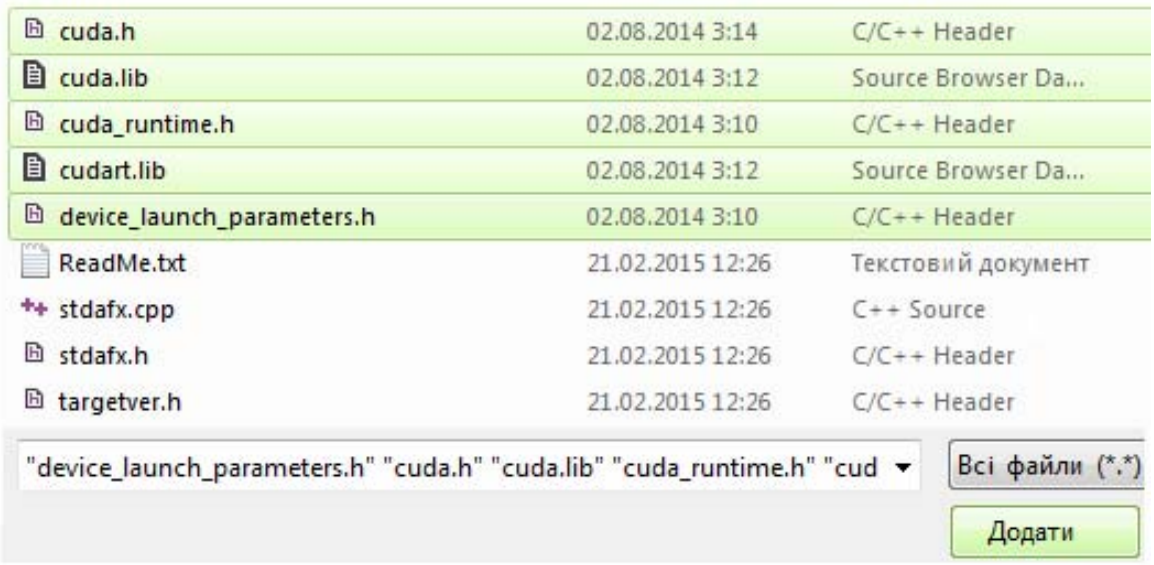

Рис. Б.7. Додавання файлів CUDA

5. Натиснути правою кнопкою на проект і в меню натиснути «Properties». У вікні вибрати «Manifest Tool» – «Input And Output» і додати текст cudart.lib;cuda.lib; (рис. Б.8).

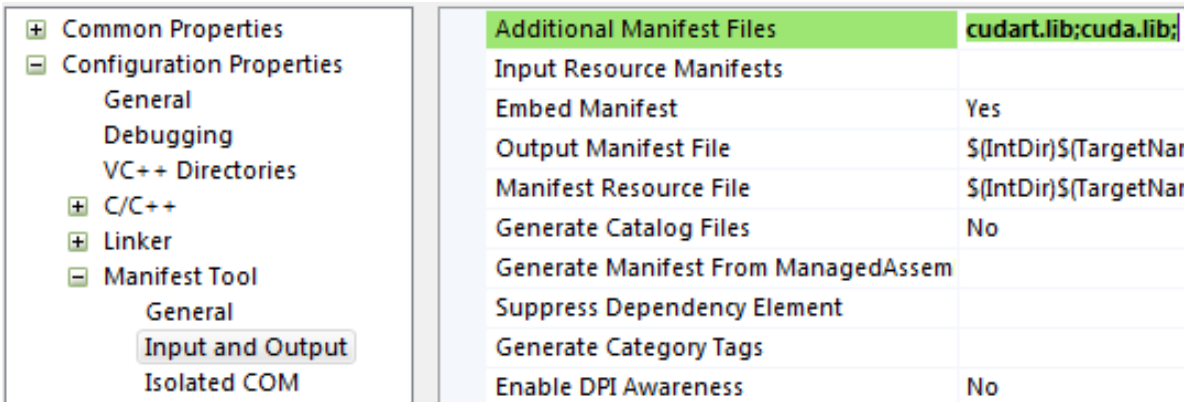

Рис. Б.8. Додавання бібліотек CUDA

6. Знайти в установленому CUDA Toolkit наступні файли (рис. Б.9) і скопіювати їх на системний диск за адресою Program Files / MSBuild / 4.0.

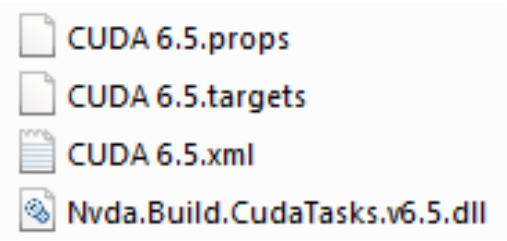

Рис. Б.9. Файли властивостей CUDA

7. Правою кнопкою натиснути на проекті і в меню натиснути «Build Customization» (рис. Б.10). У вікні зробити позначку і натиснути ОК (рис. Б.11).

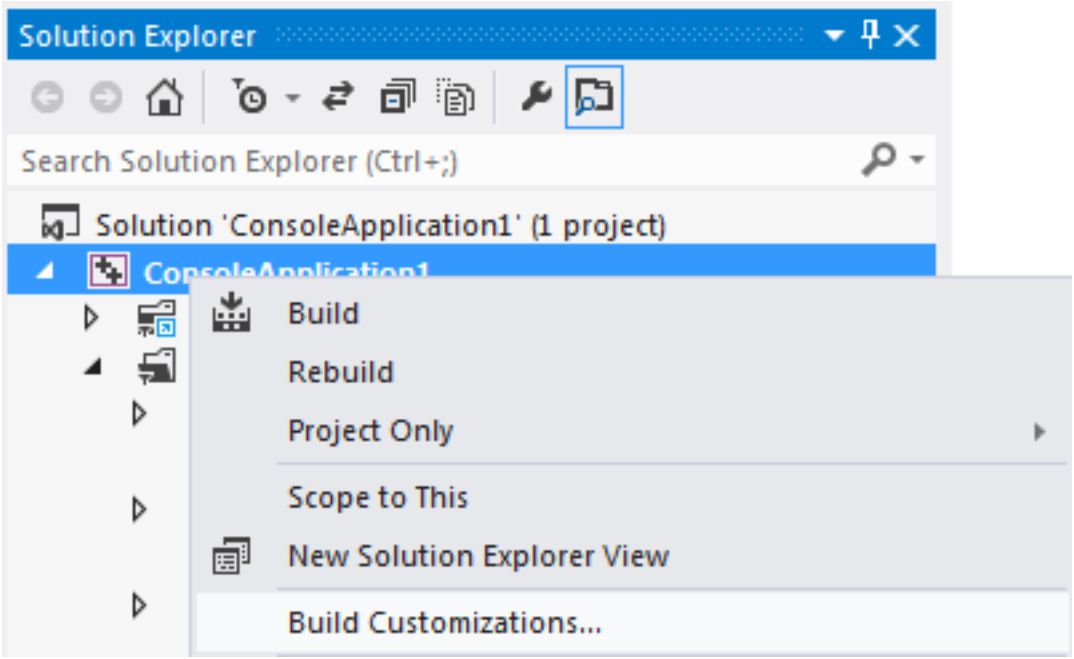

Рис. Б.10. Налаштування побудови

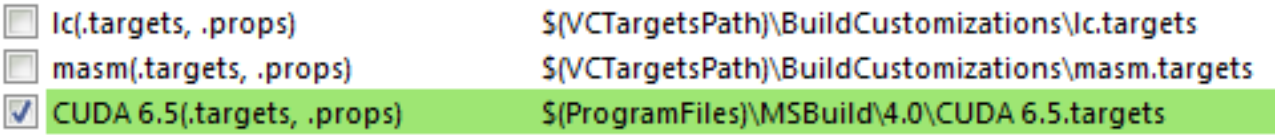

Рис. Б.11. Додавання CUDA в налаштуваннях побудови

8. Правою кнопкою натиснути на файлі ConsoleApplication1.cu і в меню натиснути «Properties». Відкрити «General» – «Item Type» і встановити «CUDA C/C++» (рис. Б.12). Натиснути «ОК».

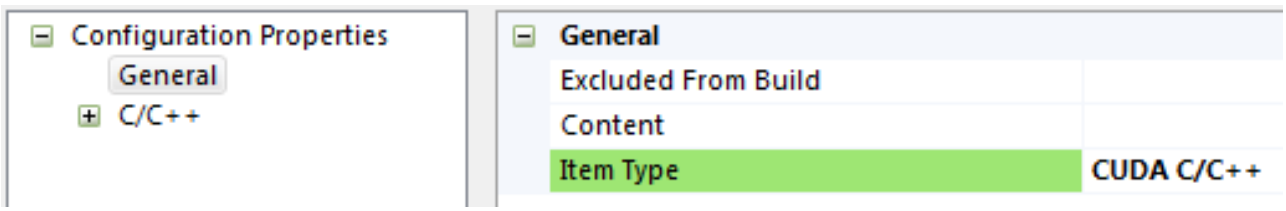

Рис. Б.12. Зміна властивостей конфігурації файлу

9. Вставити в файл ConsoleApplication1.cu наступний код (простий приклад, який обчислює на CUDA синус для елементів масиву) замість того коду, який був у файлі.

```
#include "stdafx.h"
#include "cuda_runtime.h"
#include <cuda.h>
#include "device_launch_parameters.h"
#include <stdio.h>
#include <time.h>
#include <iostream>
using namespace std;

__global__ void func(double *out, double *in){ // функція, яка виконується на GPU
int i = threadIdx.x+blockIdx.x*blockDim.x; // індекс потоку
out[i] = sin(in[i]); // обчислення синуса
}

int main(){

int n=100000; // розмір масиву
cout << "array length = ";
printf("%u", n);
cout << "\n";

double *in = new double[n]; // масив для початкових значень
double *out = new double[n]; // масив для значень синуса

for(int i=0;i<n;i++){
        in[i]=i;//заповнення масиву початковими значеннями
}

 size_t size = n * sizeof(double); // обсяг пам’яті для зберігання масиву

clock_t startTime = clock(); // вимір часу початку
double * dIn = (double*)malloc(size); // створення покажчика на пам’ять GPU
double * dOut = (double*)malloc(size); // створення покажчика на пам’ять GPU
```

```
cudaMalloc((void**)&dIn, size); // виділення пам’яті на GPU
cudaMalloc((void**)&dOut, size); // виділення пам’яті на GPU
cudaMemcpy(dIn, in, size, cudaMemcpyHostToDevice); // переміщення масиву в пам’ять GPU

func<<<100,1000>>>(dOut,dIn); // запуск обчислень на GPU

cudaMemcpy(out, dOut, size, cudaMemcpyDeviceToHost); // отримання результатів з GPU
cudaFree(dIn); // звільнення пам’яті GPU
cudaFree(dOut); // звільнення пам’яті GPU
clock_t endTime = clock(); // вимір часу закінчення
cout << "CUDA execution time = ";
cout << endTime – startTime; // час обчислення на GPU і час операцій із пам’яттю
cout << " ms\n";

for(int i=0;i<20;i++){ // відображення перших 20 результатів
        cout << "sin[";
        printf("%u",i);
        cout << "]=";
        printf("%g", out[i]);
        cout << "\n";
}

cout << "\nEnd.\n";
}
```

10. Запустити проект без налаштування «DEBUG» – «Start Without Debugging».

Додаток В

ТЕХНОЛОГІЯ JCUDA

В.1. ВІДПРАЦЮВАННЯ НАЛАШТУВАННЯ JCUDA ДЛЯ СЕРЕДОВИЩА РОЗРОБКИ ECLIPSE У WINDOWS X32

1. Для розробки JCUDA-програм необхідно встановити в зазначеній послідовності:

- CUDA Toolkit https://developer.nvidia.com/cuda-toolkit;
- Java Development Kit (JDK) http://www.oracle.com/technetwork/java/javase/downloads/index.html;
- середовище розробки для Java, в цьому випадку Eclipse (рекомендується Eclipse IDE for Java EE Developers) http://www.eclipse.org/downloads;
- файли JCUDA http://jcuda.org/downloads/downloads.html;
- JCUDA-утиліта «jcudaUtils» http://jcuda.org/utilities/utilities.html;
- компілятор мови C «cl.exe» (входить до Microsoft Visual Studio 9.0 (2008) і інші середовища розробки) для компіляції .cu-файлів.

Для запуску готових JCUDA-програм досить CUDA Toolkit і Java (JDK або JRE).

2. Додати змінні середовища, якщо вони не були додані автоматично:

- PATH з текстом C:\Program Files\Nvidia Corporation\PhysX\Common;C:\Program Files\Nvidia GPU Computing Toolkit\CUDA\v6.5\bin; C:\Program Files\Nvidia GPU Computing Toolkit\CUDA\v6.5\libnvvp; C:\Program Files\Microsoft Visual Studio 11.0\VC\bin;C:\Program Files\Java\jre7\bin;
- CUDA_PATH з текстом C:\Program Files\Nvidia GPU Computing Toolkit\CUDA\v6.5;
- CUDA_PATH_V6_5 з текстом C:\Program Files\Nvidia GPU Computing Toolkit\CUDA\v6.5;

3. Потім в Eclipse треба створити новий проект Java через меню «File» – «New» – «Java Project» (рис. В.1).

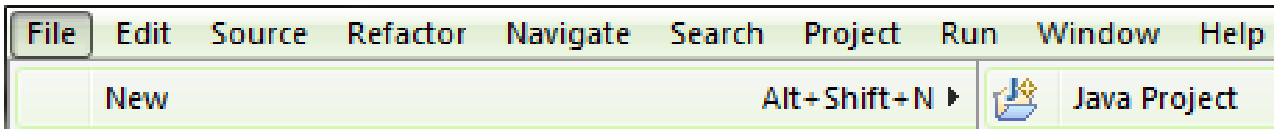

Рис. В.1. Меню для створення проекту

У вікні необхідно ввести ім’я проекту і шлях до нього (рис. В.2), а потім внизу натиснути кнопку «Finish».

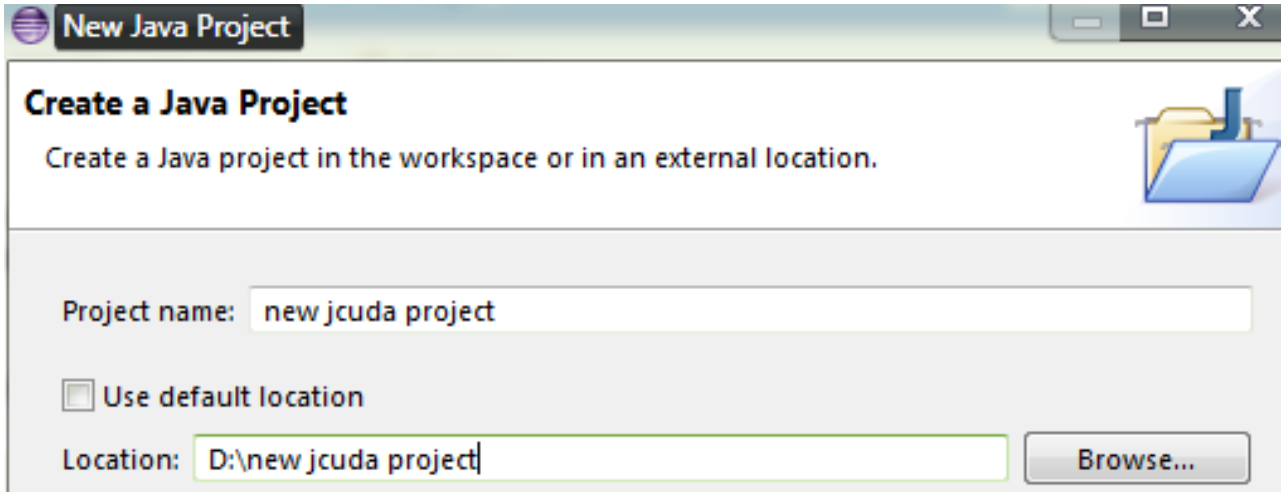

Рис. В.2. Вказівка імені проекту і шляхи до нього

4. Після скачування файлів JCUDA (рис. В.3) і файлу «jcudaUtils-0.0.4.jar» необхідно помістити файли в кореневу папку проекту Java (рис. В.4) і підключити їх до проекту в Eclipse. Обов’язкові для копіювання в проект .jar-файли «jcuda» і «jcudaUtils» і .dll-файли «JCudaDriver» і «JCudaRuntime». Решту – .jar-файли бібліотек і відповідні їм .dll-файли можна не включати в проект, якщо їх не використовувати в програмах.

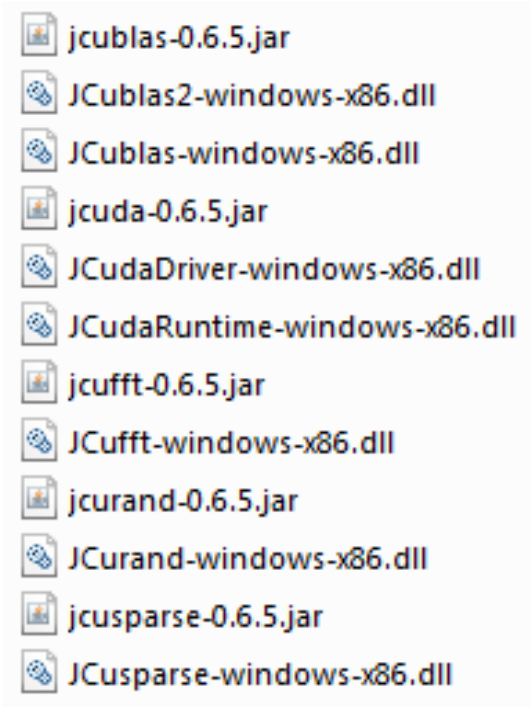

Рис. В.3. Файли JCUDA

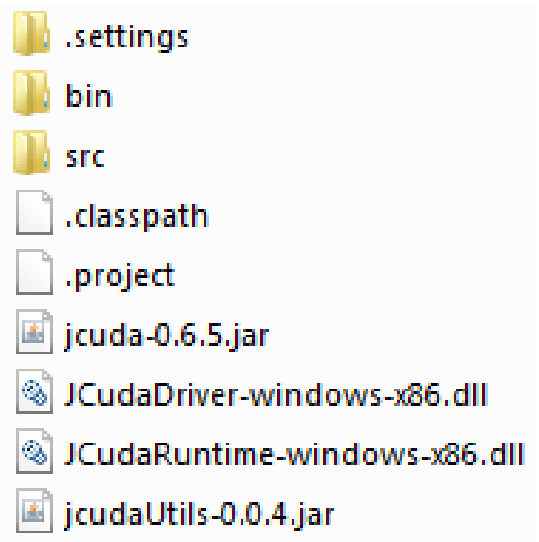

Рис. В.4. Файли JCUDA в проекті Java

Потім в Eclipse в навігаторі проекту (Package Explorer) після натискання правої кнопки миші на імені проекту і потім у випадаючому меню після натискання «Refresh» (рис. В.5) додані файли JCUDA з'являться в навігаторі проекту.

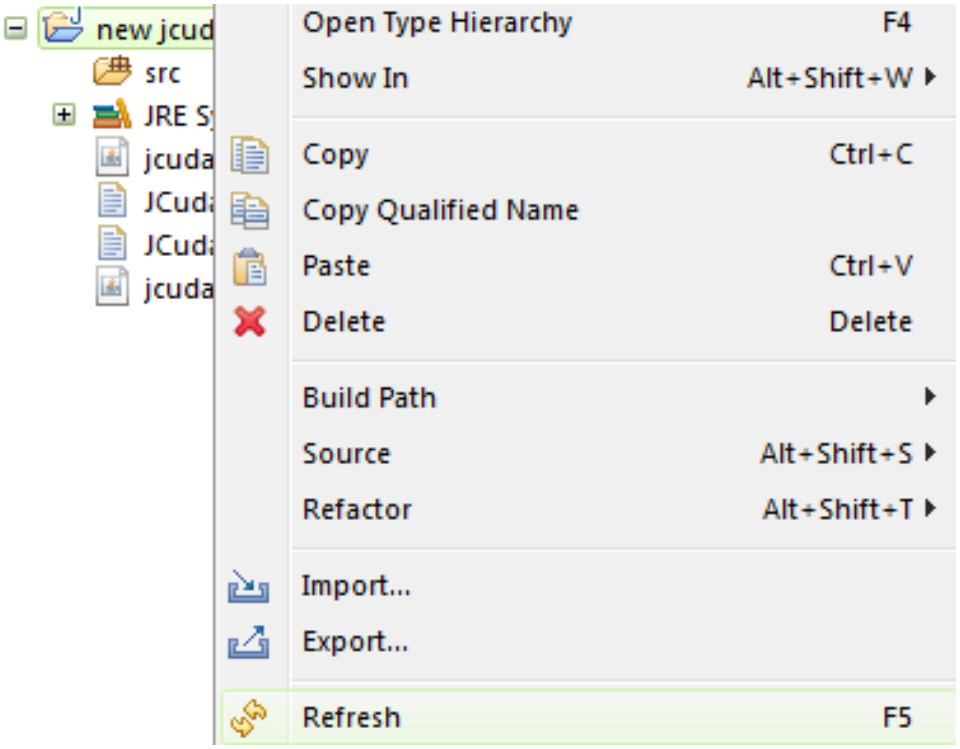

Рис. В.5. Оновлення проекту Java

5. У навігаторі проекту після натискання правої кнопки миші на імені проекту у випадаючому меню необхідно вибрати пункт «Properties». У вікні в списку ліворуч вибрати «Java Build Path». У тому ж вікні праворуч натиснути кнопку «Add JARs...» (рис. В.6).

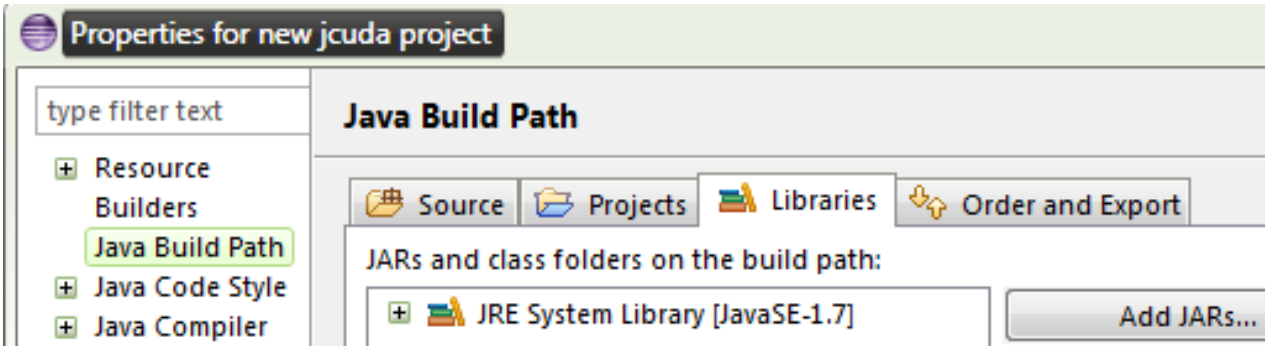

Рис. В.6. Вікно «Java Build Path»

6. З’явиться вікно, в якому необхідно вказати .jar-файл з бібліотеки JCUDA і натиснути «ОК» (рис. В.7). Таким чином додати всі необхідні .jar-файли.

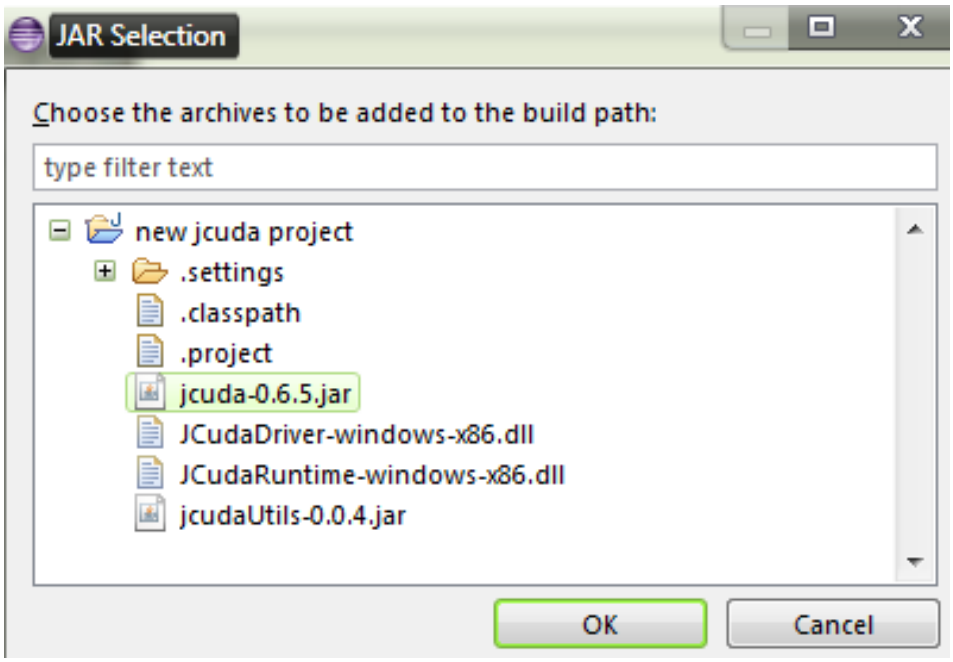

Рис. В.7. Додавання .jar-файлів

7. Потім у вікні «Java Build Path» необхідно розкрити «JRE System Library» і натиснути «Native library location: (None)». Потім натиснути кнопку праворуч «Edit» (рис. В.8) і у вікні натиснути «Workspace ...» (рис. В.9). Потім у вікні потрібно вибрати поточний проект і натиснути «OK». Так буде завершене налаштування JCUDA.

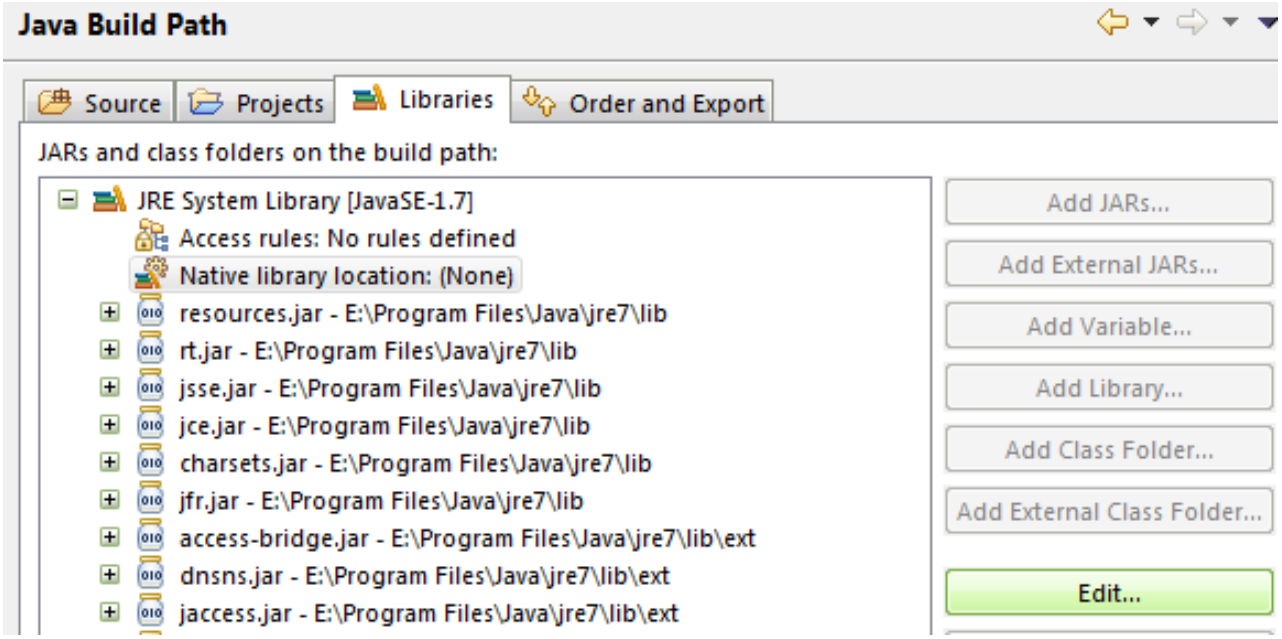

Рис. В.8. Зміна «Native library location»

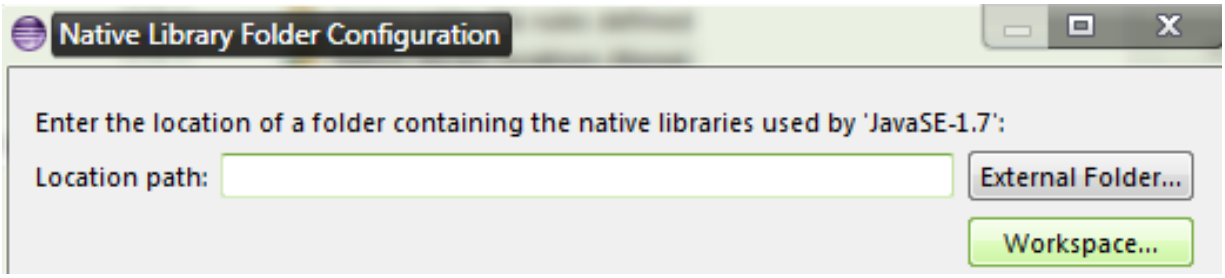

Рис. В.9. Вікно вибору «Native library location»

8. Далі під час створення програми основна її частина буде знаходиться в класах Java, а CUDA-модулі потрібно помістити в .cu-файли.

В.2. ВСТУП ДО JCUDA

Існує кілька неофіційних проектів, що забезпечують зв'язок мови програмування Java та технології CUDA. Найбільш розвиненим на сьогодні є проект JCUDA (jcuda.org). Застосування JCUDA подібно застосуванню CUDA:

- взаємодія з бібліотеками на основі CUDA API (для JCUDA додається буква «J» до імені CUDA-бібліотеки) – JcuBLAS, JcuFFT, JcuDPP, JcuRAND, JcuSPARSE та ін .;

- написання власних функцій у вигляді CUDA-модулів (.cu) мовою програмування Сі та виконання їх на GPU викликом із Java-програми.

Деякі можливості CUDA не підтримуються в JCUDA, наприклад:

- деякі шаблони покажчиків («pointers»);
- зворотні виклики («stream callbacks»);
- розрахунок завантаження GPU («occupancy calculation»);
- уніфікована віртуальна пам’ять («unified memory»);
- динамічний паралелізм («dynamic parallelism»);
- частина можливостей бібліотек CUDA.

Таблиця В.1

Порівняння CUDA та JCUDA

	CUDA	JCUDA
поділ CPU- і GPU-частини програми на різні файли	ні	так
мова програмування CPU-частини	C/C++	Java
мова програмування GPU-частини	Сі	Сі
код для компіляції функцій CUDA	не потрібен	потрібен
підключення функцій CUDA	не потрібно	KernelLauncher kernelLauncher = KernelLauncher.create (folder,"func","");
обсяг пам’яті для масиву	size_t size = N * sizeof(double);	int size = N * Sizeof.DOUBLE;
створення покажчика на масив	double * dIn = (double*)malloc(size);	CUdeviceptr dIn = new CUdeviceptr();
виділення пам’яті на GPU	cudaMalloc((void**)&dIn, size);	cuMemAlloc(dIn, size);
копіювання в пам’ять GPU	cudaMemcpy(dIn, arrIn, size, cudaMemcpyHostToDevice);	cuMemcpyHtoD (dIn, Pointer.to(arrIn), size);
виклик функції CUDA	func<<< nBlocks, nThreads >>>(dIn, dOut);	kernelLauncher.setup(nBlocks, nThreads).call (dIn, dOut);
синтаксис функції CUDA	__global__ void <ім’я функції>…	extern "C" __global__ void <ім’я функції>…
копіювання з пам’яті GPU	cudaMemcpy(arrOut, dOut, size, cudaMemcpyDeviceToHost);	cuMemcpyDtoH(Pointer.to (arrOut),dOut, size);
звільнення пам’яті GPU	cudaFree(dIn);	cuMemFree(dIn);

В.3. ВИЗНАЧЕННЯ ПАРАМЕТРІВ GPU У JAVA-ПРОГРАМІ

Під час запуску програми корисно робити перевірку на наявність CUDA-пристрою:

```
if(cudaGetDeviceCount(new int[1])!=cudaSuccess){
        System.out.println("Пристрій з підтримкою CUDA відсутній!");
}
```

Отримання кількості CUDA-пристроїв:

```
int deviceCountArray[] = new int[1];
cudaGetDeviceCount(deviceCountArray);
String cudaDeviceCount="CUDA пристроїв підключено:" +deviceCountArray[0];
```

Для отримання параметрів CUDA-пристрою необхідно створити об'єкт класу «cudaDeviceProp» і отримати параметри методом «cudaGetDeviceProperties», другим параметром якого вказується порядковий номер CUDA-пристрою (за наявності одного CUDA-пристрою його порядковий номер дорівнює 0, для наступного – 1 і т. д.):

```
cudaDeviceProp deviceProp = new cudaDeviceProp();
cudaGetDeviceProperties(deviceProp, 0);
```

Потім за допомогою «deviceProp» можна отримати параметри CUDA-пристрою:

```
"Відеокарта: "+new String(deviceProp.name).substring(0, new String(deviceProp.name).indexOf(0));
"Кількість ядер: "+deviceProp.multiProcessorCount*8+" (може бути неправильною)";
"Глобальна пам'ять: "+new BigDecimal((float)(deviceProp.totalGlobal Mem)/1048576).setScale(0,RoundingMode.UP).floatValue()+" Мбайт";
"Максимальна кількість потоків на блок: "+deviceProp.maxThreads PerBlock;
"Максимальний розмір блоку: "+deviceProp.maxThreadsDim[0]+"x"+ deviceProp.maxThreadsDim[1]+"x"+deviceProp.maxThreadsDim[2];
"Максимальний розмір сітки: "+deviceProp.maxGridSize[0]+ "x"+deviceProp. maxGridSize[1]+"x"+deviceProp.maxGridSize[2];
"Частота: "+new BigDecimal((float)(deviceProp.clockRate)/1000000). setScale (2, Rounding Mode.UP).floatValue()+" ГГц";
"Ліміт часу виконання: "+(deviceProp.kernelExecTimeoutEnabled!= 0?"Yes":"No");
```

Отримання «CUDA Driver Version» та «CUDA Runtime Version»:

```
int driverVersionArray[] = new int[1];
cudaDriverGetVersion(driverVersionArray);
int runtimeVersionArray[] = new int[1];
cudaRuntimeGetVersion(runtimeVersionArray);
"CUDA Driver Version: "+(driverVersionArray[0]/1000)+"."+
(driverVersionArray[0]%100);
"CUDA Runtime Version: "+(runtimeVersionArray[0]/1000)+"."+
(runtimeVersionArray[0]%100);
```

Параметри ОС, які можуть знадобитися для забезпечення багатоплатформності, можна визначити за допомогою методу «System.get Property»:

```
"Операційна система: "+System.getProperty("os.name");
"Архітектура ОС: "+System.getProperty("os.arch");
"Версія ОС: "+System.getProperty("os.version");
```

В.4. ПІДКЛЮЧЕННЯ CUDA-МОДУЛІВ І ЇХ ВИКОНАННЯ

CUDA-модуль являє собою файл з розширенням .cu. Для створення файлу можна відкрити файл .txt «Блокнотом» і зберегти його як файл .cu, додавши після імені файлу .cu і вказати в типі файлу «Всі файли» замість «Текстові документи (.txt)». Файл .cu редагується в середовищі розробки для мови Java або текстовим редактором, наприклад програмою «Блокнот».

В .cu-файлі пишеться код мовою програмування Сі з розширеннями для CUDA у вигляді функцій, які викликаються або з Java-програми, або з інших функцій .cu-файлу. Перед оголошенням функцій вказується директива «extern "C"». В іншому код функції подібний до того, як це було розглянуто для мови Сі.

Перед підключенням CUDA-модуля файл .cu компілюється в файл .ptx:

```
String ptxFileName = cuFileName.substring(0,
cuFileName.lastIndexOf('.')+1)+"ptx";
File ptxFile = new File(ptxFileName);
ptxFile.delete();
File cuFile = new File(cuFileName);
String command="nvcc -
m"+System.getProperty("sun.arch.data.model")+" -ptx -arch sm_35
"+cuFile.getPath()+" -o "+ptxFileName;
System.out.println("Executing\n"+command);
```

```
Process process=null;

try{
        process = Runtime.getRuntime().exec(command);
}catch(IOException e){
        e.printStackTrace();
}

String errorMessage = new String(process.getErrorStream().toString());
String outputMessage = new String(process.getErrorStream().toString());
int exitValue = 0;

try{
        exitValue = process.waitFor();
}catch (InterruptedException e){
        Thread.currentThread().interrupt();
        System.out.println("Interrupted while waiting for nvcc output, "+e);
}

if (exitValue != 0){
        System.out.println("nvcc process exitValue "+exitValue);
        System.out.println("errorMessage:\n"+errorMessage);
        System.out.println("outputMessage:\n"+outputMessage);
        System.out.println("Could not create .ptx file: "+errorMessage);
}

System.out.println("PTX file created");
```

де «cuFileName» – ім’я .cu-файлу з додаванням приставки «.cu» і відносного або абсолютного шляху до нього;

«ptxFileName» – ім’я .ptx-файлу збігається з ім’ям .cu-файлу, але має власне розширення файлу;

«ptxFile» – скомпільований .cu-файл в .ptx-файл;

«command» – команда компіляції файлу;

«process» – процес, який компілює файл;

«sm_35» – компіляція файлу за версією обчислювальних можливостей 3.5. Параметр повинен дорівнювати або бути меншим за версію обчислювальних можливостей GPU, на якому відбуваються обчислення.

Підключення CUDA-модуля являє собою підключення функції з .cu-файлу, наприклад, підключення функції «func» з файлу «func.cu»:

KernelLauncher kernelLauncher = KernelLauncher.create ("func.cu","func","");

Особливості розпаралелювання програми, створення покажчиків на масиви і копіювання масивів відбуваються подібно до того, як це було описано для мови Сі:

```
int size = N * Sizeof.DOUBLE;
int BLOCKSIZE = 50;
dim3 nThreads = new dim3(BLOCKSIZE, 1, 1);
dim3 nBlocks = new dim3(N/BLOCKSIZE, 1, 1);
CUdeviceptr ptrOutput = new CUdeviceptr();
CUdeviceptr ptrInput = new CUdeviceptr();
cuMemAlloc(ptrOutput, size);
cuMemAlloc(ptrInput, size);
cudaMemcpy(ptrInput, Pointer.to(arrInput), size, cudaMemcpyHost ToDevice);
```

Виконання функції «func» з файлу «func.cu» на GPU:

```
kernelLauncher.setup(nBlocks, nThreads).call(N, ptrInput, ptrOutput);
```

Після обчислень на GPU результат у вигляді масиву повертається («cudaMemcpyDeviceToHost») в основну частину програми і записується в масив «arrOutput», пам’ять на GPU звільняється:

```
cudaMemcpy(Pointer.to(arrOutput), ptrOutput, size, cudaMemcpy DeviceToHost);
cuMemFree(ptrInput);
cuMemFree(ptrOutput);
```

І. В. Гущин – старший викладач кафедри штучного інтелекту та програмного забезпечення факультету комп'ютерних наук Харківського національного університету імені В. Н. Каразіна з 2013 року, викладає курс «Розробка систем штучного інтелекту», займається нейрокомп'ютерами, алгоритмами Machine Learning, питаннями Data Mining. У 2008 році закінчив ХНУ імені В. Н. Каразіна за спеціальністю «Інформаційні управляючі системи та технології». Аспірант факультету комп'ютерних наук ХНУ імені В. Н. Каразіна за спеціальністю «Математичне моделювання та обчислювальні методи» у 2009–2013 роках. Автор 9 наукових статей.

В. М. Куклін – професор, доктор фізико-математичних наук, завідувач кафедри штучного інтелекту та програмного забезпечення Харківського національного університету імені В. Н. Каразіна. Автор фундаментальних праць із нелінійної теорії розповсюдження і стійкості хвиль, з теорії формування й еволюції просторових структур у нерівноважних нелінійних середовищах. Автор 2 монографій, 9 оглядів, понад 150 наукових статей у галузі природознавства, низки книг з інформатики та інших галузей науки.

О. В. Мішин – старший викладач кафедри штучного інтелекту та програмного забезпечення факультету комп'ютерних наук Харківського національного університету імені В. Н. Каразіна з 2014 року, викладає курси «Розробка систем штучного інтелекту» і «Теорія експертних систем». У 2009 році закінчив ХНУ імені В. Н. Каразіна за спеціальністю інформаційні управляючі системи та технології. Аспірант факультету комп'ютерних наук ХНУ імені В. Н. Каразіна за спеціальністю «Математичне моделювання та обчислювальні методи» у 2009–2013 роках. З 2009 року займається комп'ютерним моделюванням фізичних процесів, паралельним програмуванням на графічних процесорах і ін. З 2012 року професійно займається тестуванням Web-додатків, автоматизацією тестування. Автор 6 наукових статей.

О. В. Приймак – викладач кафедри штучного інтелекту та програмного забезпечення факультету комп'ютерних наук Харківського національного університету імені В. Н. Каразіна. У 2009 році закінчив Харківський національний університет радіоелектроніки за спеціальністю «Інформаційні управляючі системи та технології». Аспірант факультету комп'ютерних наук ХНУ імені В. Н. Каразіна за спеціальністю «Математичне моделювання та обчислювальні методи» у 2012–2015 роках, викладач із 2016 року. З 2010 року займається комп'ютерним моделюванням фізичних процесів, задач штучного інтелекту, паралельним програмуванням, зокрема на графічних процесорах. З 2011 року проводить практичні заняття у студентів із технології CUDA. Автор 12 наукових статей.

Наукове видання

Гущин Іван Валерійович
Куклін Володимир Михайлович
Мішин Олександр Вікторович
Приймак Олексій Вікторович

МОДЕЛЮВАННЯ ФІЗИЧНИХ ПРОЦЕСІВ ІЗ ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ CUDA

Монографія

Коректор *А. І. Самсонова*
Комп'ютерне верстання *Н. О. Ваніна*
Макет обкладинки *Ю. М. Рижова*

Формат 60x84/16. Ум. друк. арк. 7,6. Наклад 300 пр. Зам. № 42/17.

Видавець виготовлювач
Харківський національний університет імені В. Н. Каразіна,
61022, м. Харків, майдан Свободи, 4.
Свідоцтво суб'єкта видавничої справи ДК № 3367 від 13.01.2009

Видавництво ХНУ імені В. Н. Каразіна
Тел. 705-24-32