УДК 519.688

І.П. Струбицька
Тернопільський національний економічний університет,

ОСОБЛИВОСТІ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ РОЗПАРАЛЕЛЕННЯ ПРОЦЕСУ ПОБУДОВИ ДИСКРЕТНИХ ДИНАМІЧНИХ МОДЕЛЕЙ

Проаналізовано особливості програмної реалізації розпаралелення процесу побудови дискретних динамічних моделей. Для задачі розпаралелення використано SIMD-архітектуру. Для програмної реалізації застосовано технологію CUDA та графічний процесор NVIDIA.

Ключові слова: графічний процесор, CUDA, паралельні обчислення, розпаралелення, дискретні динамічні моделі.

Analysis of characteristics of software implementation of parallelization of process of constructing of discrete dynamical models was conducted in this paper. SIMD-architecture was used for the task of parallelization. Technology CUDA and GPU NVIDIA was used for this software implementation.

Key words: GPU, CUDA, parallel computing, parallelization, discrete dynamic models.

Вступ. Постановка проблеми

Одним із перспективних підходів до моделювання є використання динамічних моделей на основі дискретних рівнянь стану. Головною перевагою даного підходу є його універсальність як стосовно класу об'єктів, що моделюються, так і математичної форми представлення результату. Також цей підхід легко піддається комп’ютеризації. Тому він є актуальним в умовах інтенсивного застосування обчислювальних методів, які орієнтовані на сучасні комп’ютерні засоби.

У працях [1–2] динамічні моделі будуються за оптимізаційним підходом. Такий підхід дає змогу зробити побудову моделей універсальною. Проте така універсалізація призводить до появи складних оптимізаційних задач, які важко розв’язати за прийнятний час навіть з використанням сучасних засобів обчислювальної техніки. У деяких випадках математичні моделі повинні забезпечити моделювання у реальному масштабі часу. Також для врахування всіх чинників моделі її обчислювальна складність зростатиме експоненційно, а отже, необхідні значні обчислювальні затрати. Все це зумовлює високі вимоги до швидкодії та необхідної оперативної пам’яті обчислювальних засобів.

Тому актуальним є створення таких методів побудови моделей, які б забезпечували необхідну швидкодію і були реалізовані на доступних засобах комп’ютерної техніки. Саме такими методами є розпаралелення обчислювального процесу, яке відповідає зазначеним вище вимогам.

Однак більшість сучасних технічних засобів розпаралелення сьогодні є високовартісними. Тому для розпаралелення обчислювальних процедур доцільно використовувати загальнодоступні апаратні засоби, що підтримують розпаралелення обчислень. Найпоширеніші з них – це кластерні, багатопроцесорні, багатоядерні системи і графічні обчислювальні системи. Кластерні та багатопроцесорні системи мають високу вартість, багатоядерні не забезпечують відповідної продуктивності. Сучасні графічні процесори, навпаки, поширені в настільних комп'ютерах та ноутбуках і орієнтовані на виконання паралельних обчислювальних операцій з масивами даних у реальному масштабі часу. Саме тому актуальною є адаптація класичних алгоритмів моделювання для реалізації на графічних процесорах.

У статті розглянуто особливості програмної реалізації розпаралелення побудови дискретних динамічних моделей на GPU.

Аналіз останніх досліджень

Методи побудови дискретних динамічних моделей є достатньо розроблені та широко використовуються. Задачі параметричної ідентифікації дискретних динамічних моделей достатньою мірою описано в працях Л. Заде та Ч. Дезоера [3], В. Стрейца [4] Л. Люнга [5], В. М. Кунцевича [6]. Переваги методу дискретних рівнянь стану описано у працях [7–9]. У працях П. Г. Стахіва та Ю. Я. Козака [1–2] запропоновано використовувати оптимізацію для ідентифікації параметрів дискретних динамічних моделей. Однак такий підхід має високу обчислювальну складність, відповідно великими будуть і затрати машинного часу на побудову моделі. Використання графічних процесорів для неграфічних обчислень є новим напрямом досліджень. Особливості застосування GPU і технології CUDA описано у працях [10–12].

Виклад основного матеріалу

Розглянемо узагальнену математичну модель у формі дискретних рівнянь стану (1):

$$\begin{cases} \mathbf{x}^{(k+1)} = F\mathbf{x}^{(k)} + G\mathbf{v}^{(k)} + \mathbf{\Phi}(\mathbf{x}^{(k)}, \mathbf{v}^{(k)}), \\ \mathbf{y}^{(k+1)} = C\mathbf{x}^{(k+1)} + D\mathbf{v}^{(k+1)}, \end{cases} \quad (1)$$

де $\mathbf{x}^{(k)}$ – вектор змінних стану; $\mathbf{v}^{(k)}$ – вектор вхідних значень; $\mathbf{y}^{(k+1)}$ – вектор вихідних значень; F, G, C, D – матриці з невідомими коефіцієнтами, які необхідно знайти при побудові моделі; $\mathbf{\Phi}$ – деяка нелінійна вектор-функція багатьох змінних, форму і коефіцієнти якої також потрібно знайти.

Така форма моделі (1) характеризується деяким вектором невідомих параметрів $\mathbf{l}$. Для моделі (1) цей вектор складається з елементів матриць F, G, C, D та коефіцієнтів вектор-функції $\mathbf{\Phi}(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})$.

Критерієм точності моделі є $Q(\mathbf{l}) > 0$. Цей критерій позначає відхилення поведінки моделі від поведінки модельованого об’єкта. Функція мети $Q(\mathbf{l})$ розраховується як середньоквадратична похибка значень моделі від значень модельованої системи:

$$Q(\mathbf{l}) = \sum (\mathbf{y} - \mathbf{y}^*(\mathbf{l}))^2,$$

де $\mathbf{y}$ – відомі характеристики модельованого об'єкта; $\mathbf{y}^*(\mathbf{l})$ – перехідні характеристики, розраховані за допомогою моделі.

Тому побудову такої моделі можна звести до знаходження значень вектора $\mathbf{l}$, при яких функція мети буде мінімальною. Ця задача розв’язується за допомогою алгоритму оптимізації.

Задача знаходження мінімальної точки нелінійної функції $Q(\mathbf{l})$ багатьох змінних є складним завданням. При побудові дискретних динамічних моделей функція мети має чітко виражений ядровий характер з великою кількістю локальних мінімумів. Для розв’язання таких задач найкращими характеристиками володіє метод напрямного конуса Растригіна [13]. За допомогою цього підходу можна провести цілеспрямований перебір локальних мінімумів, що прискорює знаходження глобального мінімуму цільової функції [14].

Характерною особливістю оптимізованої функції є складність її розрахунку в сенсі затрат процесорного часу. Оскільки функція мети – це сума відхилень поведінки моделі від поведінки об’єкта, що моделюється, для певної множини часових відліків, то час, що необхідний для її обчислення, лінійно залежатиме від кількості дискрет. Відповідно обчислювальна складність задачі буде пропорційна кількості вхідних даних, а не порядку створюваної моделі. Для побудови якісної моделі потрібно залучити значну кількість даних, що використовуються для обчислення функції мети. Отже, значними будуть затрати машинного часу на обчислення значень відповідних функцій.

Для такої процедури побудови дискретних динамічних моделей пропонується використати розпаралелення за двома рівнями паралелізму: крупнозернистий та дрібнозернистий [14].

У праці [14] запропоновано розпаралелення цієї задачі з використанням SIMD-архітектури, яка дозволяє виконувати один і той самий потік команд для багатьох потоків даних. Блок-схему такого розпаралелення наведено на рис. 1.

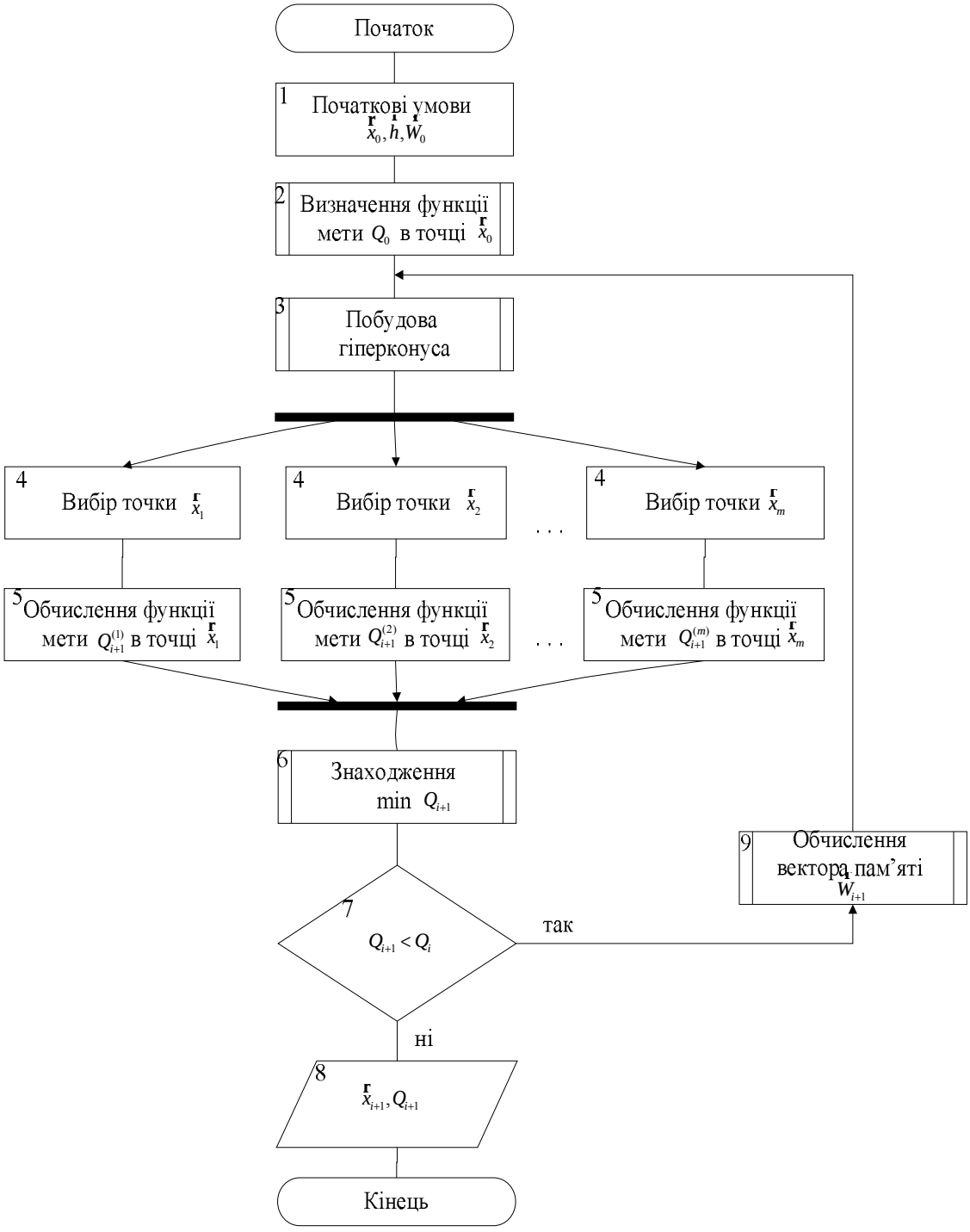

Рис. 1. Блок-схема розпаралелення оптимізаційної процедури побудови моделей [14]

З практично доступних сьогодні пристроїв з SIMD-архітектурою та за критерієм ціна/продуктивність є графічні процесори [10, 12].

GPU є пристроєм, який реалізує потокову обчислювальну модель. Тобто є потоки вхідних і вихідних даних, які складаються з однотипних елементів, що можуть бути оброблені незалежно один від одного. Обробка елементів здійснюється ядром (kernel). Модель потокових обчислень наведено на рис. 2.

Рис. 2. Модель потокових обчислень

З програмного погляду реалізація запропонованого розпаралелення не вимагає великих затрат завдяки програмній архітектурі графічного процесора. Для всіх потоків пересилається ідентичний програмний код. Вхідними даними для кожного потоку є параметри моделі, які для всіх потоків однакові. На виході кожного потоку отримується значення функції мети, яке обчислене у випадковій точці гіперконуса. Для кожного потоку це значення своє. Усі вихідні дані зберігаються у спільній пам'яті потоків для подальшого обчислення мінімуму.

Для програмної реалізації розпаралеленого алгоритму використано технологію CUDA (Compute Unified Device Architecture). Сьогодні обчислення на графічних процесорах за технологією CUDA – це інноваційне поєднання обчислювальних особливостей нового покоління графічних процесорів NVIDIA, що обробляють відразу тисячі потоків з високим рівнем інформаційного завантаження [12].

Ця технологія грунується на розширенні мови С, яка дає можливість організації доступу до набору інструкцій графічного прискорювача та управління його пам'яттю при організації паралельних обчислень. Технологія CUDA допомагає реалізовувати алгоритми, що виконуються на відеоприскорювачах GeForce восьмого покоління і старших (серії GeForce 8, GeForce 9, GeForce 200), а також Quadro і Tesla. Основними перевагами цієї технології є її безкоштовність, гнучкість і простота.

Основними характеристиками технології CUDA є [12]:

– уніфікований програмно-апаратний розв'язок для паралельних обчислень на відеочипах NVIDIA;
– великий набір підтримуваних рішень, – від мобільних до мультичипових;
– стандартна мова програмування С;
– стандартні бібліотеки числового аналізу FFT (швидке перетворення Фур'є) і BLAS (лінійна алгебра);
– оптимізований обмін даними між CPU і GPU;
– взаємодія з графічними API OpenGL і DirectX;
– підтримка 32– і 64-бітних операційних систем: Windows XP, Windows Vista, Linux і MacOS X;
– можливість розроблення на низькому рівні.

Проте технологія CUDA має певні обмеження [12]:

– відсутність підтримки рекурсії для функцій, що виконуються;
– мінімальна ширина блоку в 32 потоки.

Модель паралельних обчислень за допомогою технології CUDA дає змогу програмісту спочатку розбити задачу на незалежні блоки, які можуть виконуватись паралельно. Тоді кожен блок розбивається на множину потоків (thread), які теж виконуються паралельно та можуть залежати один від одного. Модель CUDA забезпечує засоби розширення мови С для паралельного запуску множини потоків, що виконують одну й ту саму функцію (ядро, kernel). Потоки об'єднуються у блоки, а блоки об'єднуються в мережу (решітка, grid). Фактично блок відповідає підзадачі, що виконується незалежно. Потоки всередині блока можуть взаємодіяти, тобто спільно розв'язувати підзадачу через спільну пам'ять та функцію синхронізації всіх потоків блока [12].

Швидкодія програми суттєво залежить від швидкості робіт з пам'яттю. Саме тому в традиційних CPU велику частину кристала займають різні кеші, що призначені для пришвидшення роботи з пам'яттю. Тоді, як для GPU, основну частину кристала займають ALU (арифметико-логічні пристрої).

На основі цього можна зробити висновок, що технологія CUDA добре підходить для розв'язання широкого кола задач з високим паралелізмом. Ця технологія працює на великій

кількості відеопроцесорів NVIDIA, покращує модель програмування GPU, значно спрощує її та додає велику кількість можливостей.

Програми, написані з використанням технології CUDA мають розширення .cu, а для їх компіляції використовують утиліту nvcc, яка входить до складу CUDA Toolkit. Ця утиліта розділяє код для CPU і код для GPU. Для компіляції коду на центральному процесорі використовується зовнішній компілятор.

Для написання програми за допомогою технології CUDA потрібно здійснити такі кроки:

1. Виділення пам'яті на GPU;
2. Копіювання даних з пам'яті CPU у виділену пам'ять GPU;
3. Запуск ядра або послідовний запуск кількох ядер;
4. Копіювання результатів обчислень назад у пам'ять CPU;
5. Звільнення виділеної пам'яті GPU.

Тобто розпаралелення виконується автоматично під час запуску ядра. На центральному процесорі виконується тільки послідовна частина, що передбачає підготовку та копіювання даних на пристрій, задання параметрів для ядра та його запуск. Паралельні частини оформляються в ядра, які виконуються на великій кількості потоків на GPU.

Розглянемо детально програмну реалізацію розпаралелення процесу побудови дискретних динамічних моделей. На першому етапі потрібно виділити пам'ять на графічному процесорі для даних. При розпаралеленні обчислень функції мети використовують такі дані: вектор вхідних значень $\vec{v}$, вектор вихідних значень $\vec{y}$ і матриця l, яка містить F, G, C, D та $\vec{x}_0$. Найпростішим способом виділення та звільнення лінійної пам'яті в GPU є використання функцій cudaMalloc та cudaFree відповідно [12]. Тоді потрібно скопіювати дані з пам'яті центрального процесора в пам'ять графічного. Для цього використовують функцію cudaMemcpy.

Після того, як провели копіювання даних із пам'яті CPU у виділену пам'ять GPU, здійснюють запуск ядра або послідовний запуск кількох ядер. Ядро – це функція (послідовність команд графічного процесора) з паралелізмом даних, що виконується над великою кількістю потоків. Для запуску ядра на GPU використовують конструкцію kernelName<<<аргументи>>>.

Ядро паралельної програми для оптимізаційної задачі побудови дискретних динамічних моделей наведено на рис. 3.

Після виклику ядра для обчислень необхідно скопіювати результати назад в оперативну пам'ять. Цей процес здійснюється за допомогою функції cudaMemcpy. Проте функція ядра має особливість – асинхронне виконання. У цьому випадку потрібно використати засоби синхронізації, а саме події (event). Тому перед копіюванням результатів у CPU виконуємо синхронізацію потоків графічного процесора за допомогою подій.

Для створення подій використано функцію cudaEventCreate. Запис події виконано за допомогою функції cudaEventRecord. За допомогою функції cudaEventSynchronize проведено синхронізацію подій. Ця функція чекає закінчення роботи всіх потоків графічного процесора і проходження заданої події та тільки після цього віддає управління програмі. На рис. 4 представлено схему, яка відображає процес синхронізації на графічному процесорі [12].

Після виконання ядра результати обчислень копіюються з пам'яті GPU в пам'ять CPU. Після цього потрібно звільнити пам'ять графічного процесора.

Узагальнюючи викладене, побудову дискретних динамічних моделей з використанням багатоядерного графічного процесора можна подати такими кроками:

1. Ініціалізація;
2. Ітерація підготовки;
3. Виконання обчислень на GPU;
4. Ітерація завершення.

На рис. 5 наведено модифікацію цього алгоритму.

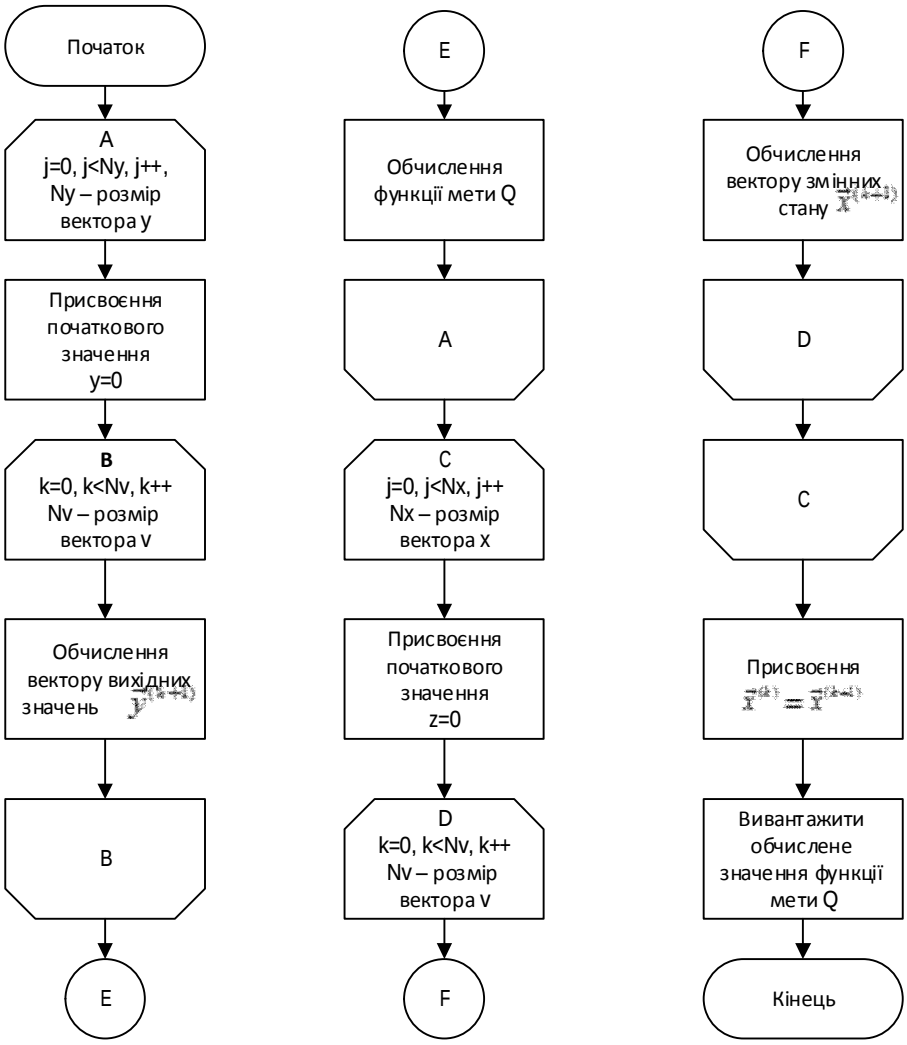

Рис. 3. Блок-схема ядра паралельної програми побудови дискретних динамічних моделей

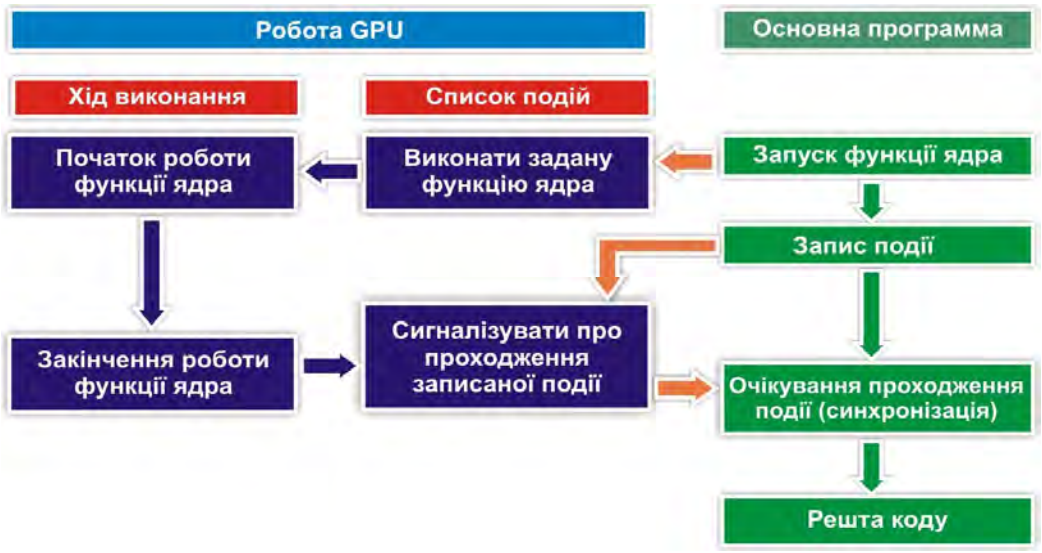

Рис. 4. Синхронізація роботи основної програми і GPU

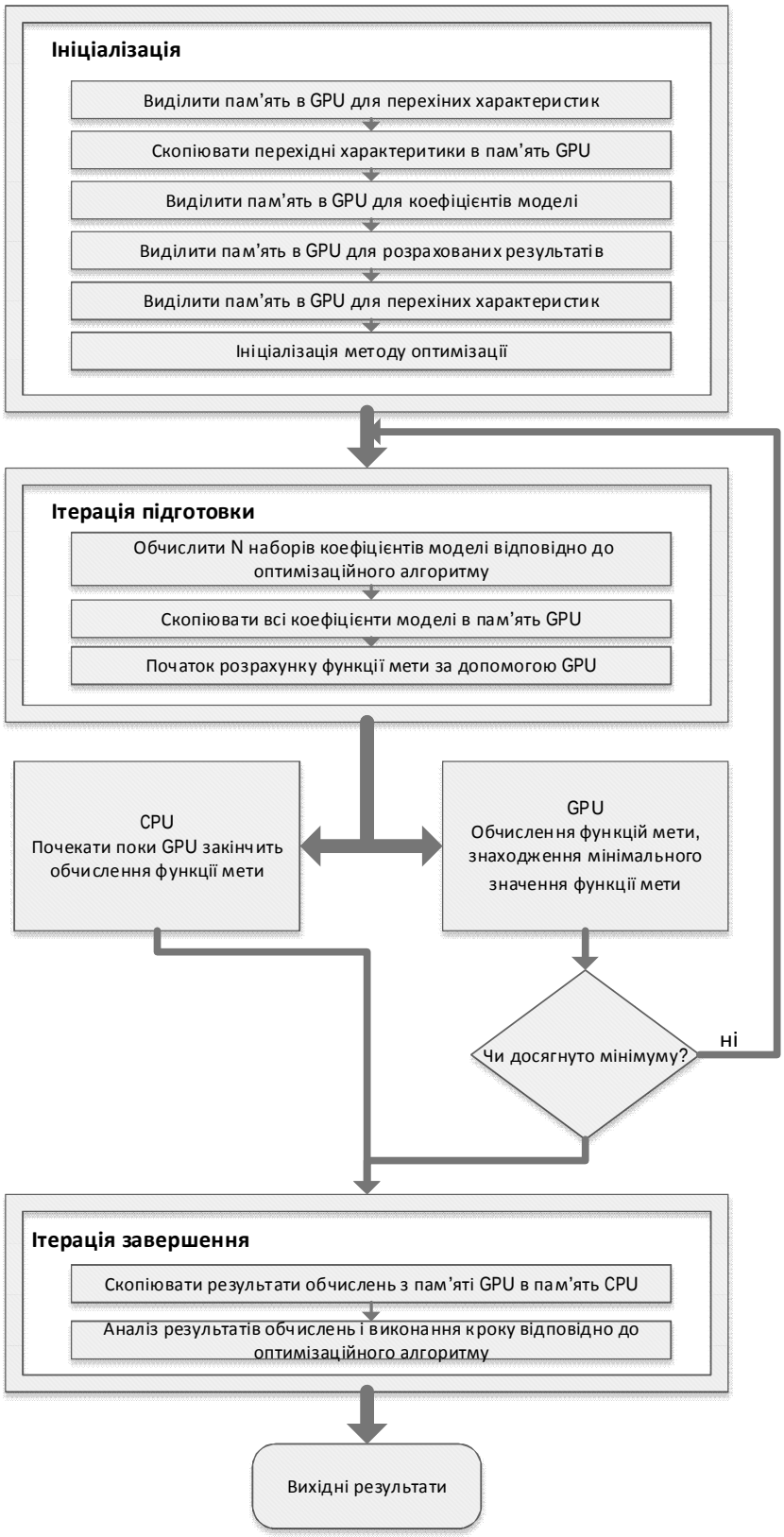

Рис. 5. Модифікація алгоритму побудови дискретних динамічних моделей з використанням GPU

Як видно з рис. 5, на першому етапі проводиться ініціалізація, тобто виділяється пам'ять на графічному процесорі для перехідних характеристик, для коефіцієнтів моделі та для розрахованих результатів. Тоді перехідні характеристики копіюються в пам'ять GPU та проводиться ініціалізація методу оптимізації. Наступним етапом є ітерація підготовки, коли обчислюють N наборів коефіцієнтів моделі. Тоді всі коефіцієнти моделі копіюються в пам'ять графічного процесора та

розраховують функцію мети за допомогою GPU. Центральний процесор очікує, поки графічний процесор закінчить всі обчислення, а GPU проводить обчислення функцій мети і знаходження мінімального значення цільової функції. На наступному етапі відбувається перевірка – чи досягнуто мінімуму. При негативній відповіді повертаємось до ітерації підготовки, а при досягненні мінімального значення функції мети переходимо до ітерації завершення. На цьому кроці результати обчислень з пам'яті графічного процесора копіюються в пам'ять центрального процесора та аналізуються результати обчислення.

Реалізоване програмне забезпечення є кросплатформеним, тобто може виконуватись і у середовищі Linux-подібних систем, і у середовищі Windows. Також ця програма є універсальною для всіх графічних процесорів, які підтримують технологію CUDA.

Висновки

У результаті цього дослідження проведено програмну реалізацію розпаралелення процесу побудови дискретних динамічних моделей. Для цієї процедури використано нову архітектуру графічних процесорів компанії NVIDIA та технологію CUDA. Паралельна програма побудови моделей використовує SIMD-архітектуру, яка дає змогу виконувати один і той самий потік інструкцій для багатьох наборів даних, що характерне для таких задач. Нове програмне забезпечення для побудови дискретних динамічних моделей на основі використання графічних процесорів, на відміну від послідовного, характеризується нижчою часовою складністю.

1. Стахів П.Г, Побудова макромоделей електромеханічних компонент із використанням оптимізації / П.Г. Стахів, Ю.Я. Козак // Технічна електродинаміка. – 2001. – №4. – С. 33–36. 2. Стахів П.Г. Побудова математичної макромоделі електромеханічного перетворювача вентильного двигуна з використанням оптимізації / П.Г. Стахів, Ю.Я. Козак, В.Г. Гайдук // Електроенергетичні та електромеханічні системи, вісник національного університету "Львівська політехніка". – 2001. – №418. – С. 159–164. 3. Заде Л. Теория линейных систем. Метод пространства состояний / Л. Заде, Ч. Дезоер – М.: Наука, 1970. – 704 с. 4. Стрейц В. Метод пространства состояний в теории дискретних линейных систем управления/ В. Стрейц [Пер. с англ. под ред. Я.З. Цыпкина] – М.: наука. Главная редакция физико-математической литературы, 1985. – 296 с. 5. Люнг Л. Идентификация систем. Теория для пользователя / Л. Люнг; [пер. с англ. под ред. Я.З. Цыпкина]. – М.: Наука. Гл. ред. физ.-мат. лит., 1991. – 432 с. 6. Кунцевич Р.М. О редуцированных моделях дискретных динамических объектов и их гарантированных оценках в задачах управления / Р.М. Кунцевич // Проблемы управления и информатики. – 2001. – № 1. – С. 42- 50. 7. Стахив П.Г. Анализ динамических режимов в электронных схемах с многополюсниками [Текст] / П.Г. Стахив. – Львов: Высш. школа, 1988. – 154 с. 8. Hinamoto T. Approximation of polynomial state-affine discrete-time systems / T. Hinamoto, S. Mackava – IEEE Trans. Circ. and Syst. – 1984. – Vol. 33, № 8. – P. 713–721. 9. Isidori A. Direct Construction of minimal bilinear realization from nonlinear input-output maps / A. Isidori – IEEE. – 1973. – vol. AC-18, № 6. – P. 626-631. 10. CUDA Zone – The resource for CUDA developer. С [Електронний ресурс]. – Режим доступу: http://www.nvidia.com/object/cuda_home_new.html .11. NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, Version 2.0. – 2008. –107 p. 12. Боресков А.В. Параллельные вычисления на GPU. Архитектура и программная модель CUDA/ А.В. Боресков, А.А. Харламов и др.. – М.: Издательство Московского университет, 2012. – 336 с. 13. Растригин Л.А. Адаптация сложных систем / Л.А. Растригин – Рига: Зинатне, 1981. – 375 с. 14. Козак Ю.Я. Розпаралелення алгоритму оптимізації параметрів дискретних динамічних моделей на масивно-паралельних процесорах / Ю.Я. Козак, П.Г. Стахів, І.П. Струбицька // Відбір і обробка інформації. - 2010. – Вип. 32 (108). – С. 126–130.