УДК 519.688

Ю. Я. Козак, П. Г. Стахів, І. П. Струбицька

РОЗПАРАЛЕЛЕННЯ АЛГОРИТМУ ОПТИМІЗАЦІЇ ПАРАМЕТРІВ ДИСКРЕТНИХ ДИНАМІЧНИХ МОДЕЛЕЙ НА МАСИВНО-ПАРАЛЕЛЬНИХ ПРОЦЕСОРАХ

Method of paralleling task of parameters optimization of discrete dynamic models on massively parallel processors is proposed in this paper. Construct the block diagram of the algorithm, which allows a parallel implementation. Two methods for different areas of parallel algorithm are proposed.

Key words: *parallel computation, massively parallel processors, discrete dynamic models, optimization method.*

Запропоновано метод розпаралелення задачі оптимізації параметрів дискретної динамічної моделі на масивно-паралельних процесорах. Побудовано блок-схему алгоритму, яка дає змогу провести паралельну реалізацію. Запропоновано два методи розпаралелення для різних ділянок алгоритму.

Ключові слова: *паралельні обчислення, масивно-паралельні процесори, дискретні динамічні моделі, метод оптимізації.*

Важливим етапом у дослідженні складних динамічних систем є ідентифікація параметрів дискретної моделі. Цей етап відіграє ключову роль під час оцінення адекватності моделі у відношенні до реальної системи.

Побудова математичної моделі завжди у нашому підході [2, 4] передбачає оптимізацію в сенсі мінімізації відхилення поведінки моделі від поведінки моделюваного об'єкта.

Задача знаходження мінімуму нелінійної функції багатьох змінних є доволі складним завданням у загальному випадку. Особливо це стосується функції, які не мають аналітичного опису. Для знаходження екстремуму застосовують різні методи, які враховують характерні особливості конкретної задачі. При побудові дискретних динамічних моделей функція мети має чітко виражений яровий характер з великою кількістю локальних мінімумів. З погляду придатності до вирішення таких задач найкращими характеристиками вирізняється метод напрямного конуса Растригіна [3, 5].

Алгоритм напрямного конуса для побудови дискретних динамічних моделей можна подати такими кроками:

1. Будується гіперсфера з центром у точці $\vec{x}_0$ (параметри моделі) і радіусом h. На поверхні гіперсфери вибирається m випадкових точок із рівномірним законом розподілу. Серед цих точок вибирається точка $\vec{x}_1$, в якій значення функції мети є найменшим. Визначається вектор пам'яті:

$$\vec{W}_0 = \frac{\vec{x}_1 - \vec{x}_0}{h}. \qquad (1)$$

2. Далі на кожному кроці будується гіперконус із вершиною в точці $\vec{x}_i$, кутом розкриття ψ і напрямком $\vec{W}_i$.

3. Цей гіперконус відсікає від гіперсфери з центром у точці $\vec{x}_i$ і радіусом h деякий сегмент. На ньому вибирається m випадкових, рівномірно розподілених точок, і за напрям $\vec{x}_{i+1}$ вибирається точка, в якій значення функції мети має найменше значення. Перераховують новий вектор пам'яті:

$$\vec{W}_{i+1} = \alpha \cdot \vec{W}_i + \beta \cdot \frac{\vec{x}_{i+1} - \vec{x}_i}{h}, \qquad (2)$$

де $0 \le \alpha < 1,\ 0 < \beta \le 1.$

4. Пошук продовжується доти, поки зменшується значення функції мети. У випадку потреби, мінімум уточнюють за допомогою методу локальної оптимізації.

Функцією мети є середньоквадратичне відхилення поведінки моделі від поведінки модельованого об'єкта:

$$Q(\vec{\lambda}) = \sum_i \left| \vec{y}_i - \vec{y}_i^{\,*} \right|^2, \qquad (3)$$

де $\vec{y}_i$ – відгук моделі; $\vec{y}_i^{\,*}$ – реакція реального об'єкта; $\vec{\lambda}$ – вектор невідомих параметрів.

За допомогою запропонованого методу можна провести цілеспрямований перебір локальних мінімумів, що прискорює знаходження глобального мінімуму функції мети.

Блок-схему алгоритму напрямного конусу Растригіна зображено на рис. 1.

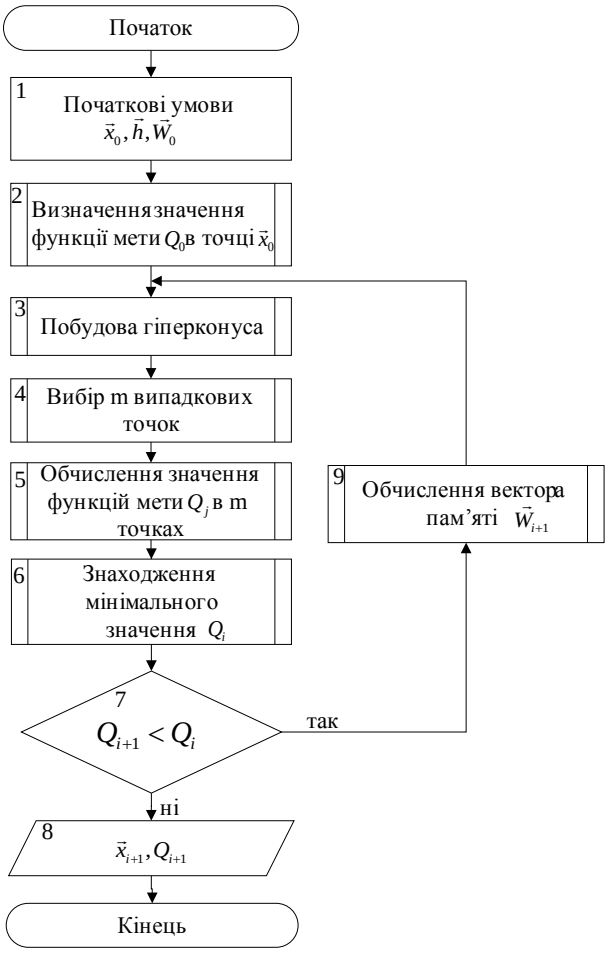

Рис. 1. Блок-схема алгоритму напрямного конуса Растригіна.

Часова складність представленого алгоритму лінійно залежатиме від кількості відомих дискрет. Відповідно, обчислювальна складність задачі буде пропорційна кількості вхідних даних. Для побудови якісної моделі потрібно вико-

ристати значну кількість даних для обчислення значення функції мети. Отже, значними будуть і затрати машинного часу на обчислення значень відповідних функцій.

Відповідно, для зменшення обчислювальних затрат доцільним є розгляд питання щодо розпаралелення обчислювального процесу.

Зважаючи на велику кількість операцій над векторами, для практичної реалізації алгоритму розглядаємо архітектуру процесорів SIMD, яка дає змогу виконати один і той самий потік команд для багатьох потоків даних, що характерне для цієї задачі. Системи цього типу будуються таким чином, що процесорні елементи, які входять у систему, ідентичні, і всі вони керуються одною і тією ж послідовністю команд. Проте кожний процесор обробляє свій потік даних. Під таку схему добре підходять задачі обробки матриць або векторів, задачі розв'язку систем лінійних та нелінійних, алгебраїчних і диференціальних рівнянь, задачі теорії поля та інші [1, 6].

В описаному вище алгоритмі пропонується використати розпаралелення за двома рівнями паралелізму: дрібнозернистий та крупнозернистий. При крупнозернистому паралелізмі кожне паралельне обчислення є незалежне від інших, причому вимагає лише епізодичного обміну інформацією між окремими процесорами. Компонентами такого розпаралелення є досить великі, однак незалежні програми, що можуть налічувати тисячі команд. На відміну від крупнозернистого, при дрібнозернистому кожне паралельне обчислення мале та елементарне і реалізується десятками команд. При ньому компонентами розпаралелення є елементи виразів або окремі ітерації циклу, які мають незначну кореляцію за даними [6].

З практично доступних сьогодні пристроїв з SIMD-архітектурою та за критерієм ціна/продуктивність є GPU (Graphics Proccesing Unit) [7-11]. Тому пропонований алгоритм пошуку екстремуму потрібно адаптувати саме для цих акселераторів обчислень.

Враховуючи специфіку архітектури GPU (рис. 2), яка передбачає використання в багатопотокових і багатопроцесорних обчислювальних системах, алгоритм на рис. 1 можна привести до алгоритму на рис. 3, який використовує розпаралелення.

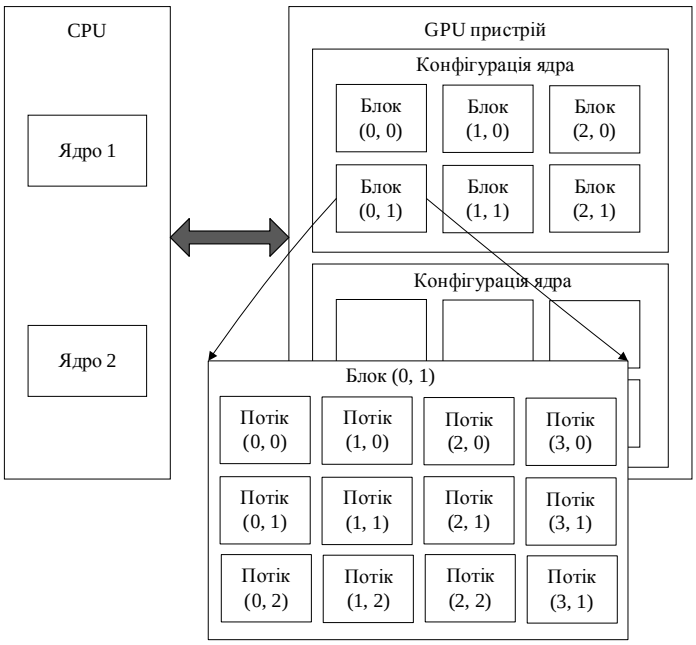

Рис. 2. Ієрархія елементів конфігурації GPU пристрою.

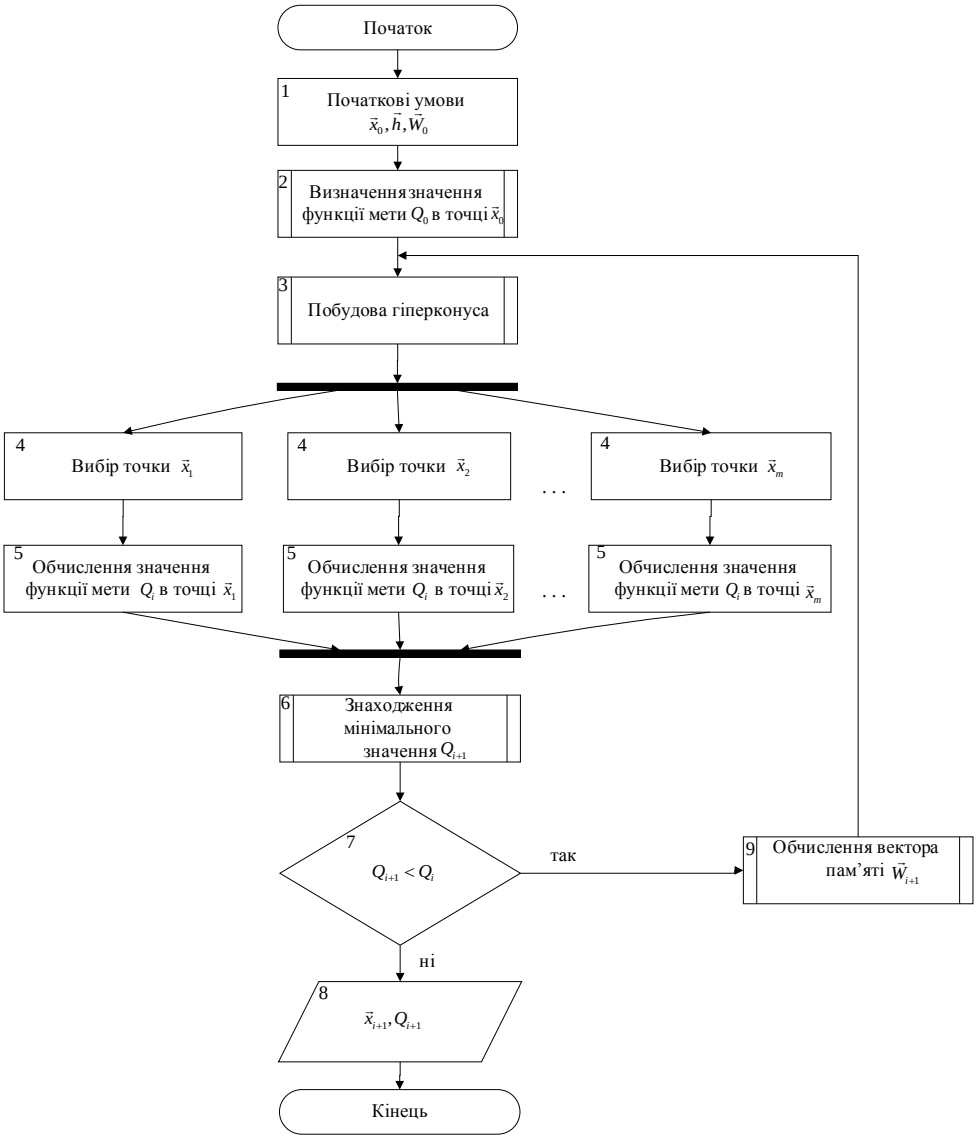

Рис. 3. Блок-схема розпаралеленого алгоритму напрямного конуса Растригіна.

У цьому алгоритмі для блоків 4, 5 паралельного алгоритму використаємо крупнозернисте розпаралелення. А саме у блоці 4 (вибір точки) кожна точка буде в окремому потоці, які виконуватимуться паралельно. У блоці 5 (обчислення значення функції мети в m точках) розрахунок значення функції мети $Q_{i+1}^{(1)},...,Q_{i+1}^{(m)}$ проводиться також на окремих потоках. Враховуючи запропоновану архітектуру, для оптимальності обчислень бажано, щоб кількість точок була кратна 128. Для цього блоку використаємо крупнозернисте і дрібнозернисте розпаралелення одночасно.

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

Основна складність у разі застосування такого підходу – в генерації випадкових точок на гіперконусі. Враховуючи особливості роботи програмних генераторів псевдовипадкових чисел, можна сподіватися, що під час виконання на паралельних процесорах з однаковими початковими даними результат генерації буде однаковим. Тобто в результаті буде отримано n абсолютно однакових точок, що є не припустимим. Для цього, щоб уникнути такої ситуації, пропонується генератори випадкових чисел ініціалізувати числом, яке є номером поточного потоку обчислення. Таке число не потребує затрат на своє отримання і це дасть гарантію того, що генератори випадкових чисел на різних потоках будуть генерувати дійсно різні випадкові послідовності випадкових чисел. Імовірність того, що в деяких потоках точки гіперконусу будуть дублюватися, є дуже мала і у випадку великої кількості потоків суттєво не вплине на обчислювальні затрати.

Для максимального використання паралельності архітектури пропонується для блоків алгоритму 6, 9 (рис. 3) застосувати дрібнозернисте розпаралелення. Зокрема, у блоці 6 необхідно знайти мінімальне значення функції мети Q. Для цього краще використати метод редукції розмірності даних, який досить легко реалізовується на запропонованій архітектурі й не вимагає значних обчислювальних затрат. Для знаходження вектора пам'яті $\vec{W}$, що обчислюється покомпонентно, у блоці 8 знову застосовується дрібнозернисте розпаралелення.

Задача редукції полягає у знаходженні мінімального значення серед елементів масиву $Q_{i+1}^{(1)}, Q_{i+1}^{(2)}, ..., Q_{i+1}^{(m)}$. При паралельній редукції лімітованим фактором є доступ до пам'яті. Для цього використовується shared-пам'ять. Класичний метод пошуку мінімуму простий, проте паралельний ефективний. Кожного разу буде звільняти фізичне місце, яке можна використати з користю.

Задачу запускають на декількох ядрах, і виникають блоки (рис. 2). Кожен блок отримує свою частину великого масиву, який копіює в shared-пам'ять. У цій же пам'яті ієрархічно обчислюється мінімальне значення. Наприклад, блок бере 128 значень, а видає одне. Тобто в результаті отримується новий масив, який в 128 разів менший за попередній. Такі операції проводять доти, поки на виході не залишиться один елемент, який і буде мінімальним. Результат зберігається у глобальній пам'яті.

Схема ієрархічного знаходження мінімального елемента подана на рис. 4. Такий метод дає змогу проводити обчислення мінімуму функції мети паралельно, використовуючи багато потоків (ниток).

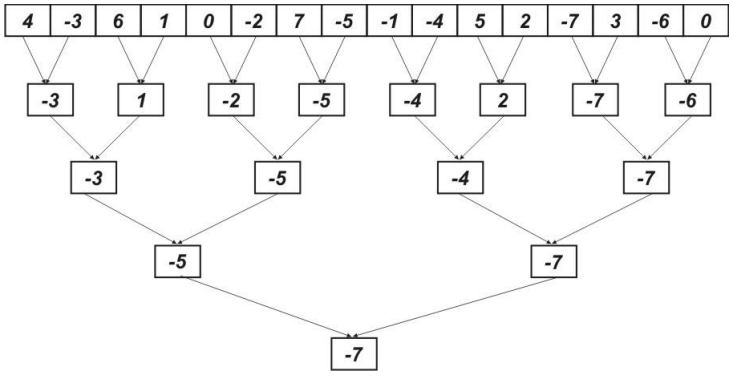

Рис. 4. Схема ієрархічного знаходження мінімального елемента.

Програмно реалізувати редукцію на архітектурі SIMD можна декількома способами, що дає змогу практично повністю позбавитись від розгалуження і від конфліктів. Можна також змінити порядок порівняння двох елементів: порівню-

ються сусідні елементи і відстань збільшують вдвічі; порівнюють найбільш віддалені елементи і відстань зменшують вдвічі на кожній ітерації циклу.

Такий підхід до розпаралелення, який передбачає одночасне використання і крупнозернистого і дрібнозернистого розпаралелення, дає змогу максимально використати обчислювальні ресурси масивно-паралельної процесорної архітектури. Коефіцієнт розпаралелення для цього алгоритму можна оцінити виходячи з того, що затрати на обчислення значення функції мети значно перевищують обчислювальні затрати для прийняття рішень згідно з оптимізаційним алгоритмом. Оскільки в паралельній реалізації обчислення значення функції мети в різних точках виконується незалежно, то коефіцієнт розпаралелення близький до кількості процесорів (за умови кратності кількості розрахунків функції мети на один крок оптимізаційної процедури).

У результаті таких модифікацій алгоритму напрямного конуса Растригіна для архітектури SIMD зменшується його обчислювальна складність. У цьому разі застосовують нові архітектурні рішення, які безпосередньо не були розроблені для задач такого класу.

ВИСНОВКИ

У результаті цього дослідження було:

1. Модифіковано метод напрямного конуса Растригіна;
2. Запропоновано спосіб розпаралелення алгоритму напрямного конуса;
3. Вибрано архітектуру SIMD як таку, що найкраще підходить до розв'язання цієї задачі;
4. Запропоновано використати поєднання двох методів розпаралелення (крупнозернистого та дрібнозернистого) для різних ділянок алгоритму;
5. Побудовано блок-схему алгоритму, яка враховує його подальшу реалізацію на архітектурі SIMD.

1. *Гергель В. П., Стронгин Р. Г.* Основы параллельных вычислений для многопроцессорных вычислительных систем: Учеб. пособие. – Нижний Новгород: Изд-во ННГУ им. Н. И. Лобачевского, 2003. – 184 с.
2. *Люнг Л.* Идентификация систем. Теория для пользователя / Пер. с англ.; под ред. Я. З. Цыпкина. – М.: Наука. Гл. ред. физ.-мат. лит., 1991. – 432 с.
3. *Растригин Л. А.* Адаптация сложных систем. – Рига: Зинатне, 1981. – 375 с.
4. *Семенов А. Д., Артамонов Д. В., Брюхачев А. В.* Идентификация объектов управления: Учебн. пособие. – Пенза: Изд-во Пенз. гос. ун-та, 2003.– 211 с.
5. *Стахів П. Г., Козак Ю. П.* Побудова макромоделей електромеханічних компонент з використанням оптимізації // Технічна електродинаміка. – 2001. – № 4. – С. 33–36.
6. *Применение* технологии MPI в Грид / Г. И. Шпаковский, В. И. Стецюренко, А. Е. Верхотуров, Н. В. Серикова. – Минск: БГУ, 2008. – 137 с.
7. *Göddeke M., Strzodka R., Turek S.* Accelerating Double Precision FEM Simulations with GPUs // Proc. of ASIM 2005 18th Symposium on Simulation Technique. – 2005. – P. 139–144.
8. *Using* the graphic processor as a highperformance computational engine for solving system of hyperbolic conservation low / T. R. Hagen, M. O. Henriksen, J. M. Hjelmervik, K.-A. Lie // Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF – Springer, 2007. – P. 211–264.
9. *Visual* simulation of shallow-water waves / T. R. Hagen, J. M. Hjelmervik, K.-A. Lie, et al. // Simulation Practice and Theory. Special Issue on Programmable Graphics Hardware. – 2005. – 13(9). – P. 716–726.
10. *Hagen T. R., Lie K.-A., Natvig J. R.* Solving the Euler equation on graphical processing units // Computational Science – ICCS 2006: 6th Int. Conf., Reading, UK, May 28–31, 2006, Proc., Part IV, Vol. 3994 of Lecture Notes in Computational Science (LNCS). – Springer Verlag, 2006. – P. 220–227.
11. *Photon* mapping on programmable graphic hardware / T. J. Purcell, C. Donner, M. Commarano, et al. // Proc. of the ACM SIGGRAPH/EUROGRAPHICS Conf. on Graphics Hardware. – Eurographics Association, 2003. – P. 41–50.

Національний університет "Львівська політехніка";
Тернопільський національний економічний університет

Одержано
03.03.2010