ISSN2224-087X. Електроніка та інформаційні технології. 2015. Випуск 5. С. 185–191
Electronics and information technologies. 2015. Issue 5. P. 185–191

УДК 519.622.2

ПАРАЛЕЛЬНІ МЕТОДИ РОЗВ'ЯЗАННЯ СЛАР З ВИКОРИСТАННЯМ АРХІТЕКТУРИ CUDA

А. Харів, І. Хвищун

Львівський національний університет імені Івана Франка,
вул. Ген. Тарнавського, 107, 79017, Львів, Україна
andriy.khariv@gmail.com

Задачі математичного моделювання, з формального погляду, не викликають труднощів, оскільки процес розв'язування систем рівнянь математичних моделей часто зводиться до розв'язування систем лінійних алгебричних рівнянь, які розв'язують методом Гауса чи його модифікаціями. Однак під час розв'язування систем лінійних алгебричних рівнянь високих порядків обчислювальна складність цих методів значна, тому актуальним є дослідження способів застосування тут паралельних алгоритмів з використанням можливостей сучасних графічних процесорів.

Ключові слова: CUDA, NVIDIA, паралельні обчислення, СЛАР, графічні процесори, математичне моделювання.

З огляду на актуальні потреби математичного моделювання фізико-технічних систем та нові можливості сучасної обчислювальної техніки, а саме – процесорів компанії NVIDIA з підтримкою паралельної програмно-апаратної архітектури CUDA [1], доцільно пристосувати алгоритми відомих послідовних методів розв'язування систем лінійних алгебричних рівнянь (СЛАР) [2] до їхньої роботи на графічних процесорах способом паралельного виконання цими процесорами невластивих їм обчислень загального призначення.

Для оптимальної оцінки ефективності роботи подібних паралельних алгоритмів використаємо СЛАР на основі матриці Гільберта, яка є не виродженою, але погано зумовленою.

Запишемо СЛАР у матричній формі: $\boldsymbol{Ax} = \boldsymbol{b}$, де $\boldsymbol{A} = (a_{ij})$ – матриця Гільберта розміру $n \times n$, елементи якої задають як $a_{ij} = \dfrac{a}{i + j - 1}$, b – вектор розміру n.

Прямий хід алгоритму методу Гауса полягає у послідовному виключенні невідомих у системі рівнянь, його обчислюють так:

$$a_{ki} = a_{kj} - \mu_{ki} \cdot a_{ij}; \quad b_k = b_k - \mu_{ki} \cdot b_i,$$

$$i \le j \le n-1; \; i < k \le n-1; \quad 0 \le i < n-1,$$

де $\mu_{ki} = (a_{ki} / a_{ii})$ – множники Гауса.

Зворотний хід алгоритму полягає в знаходженні вектора невідомих СЛАР із верхньою трикутною матрицею і в загальному вигляді його можна записати так:

$$x_i = (c_i - \sum_{j=i+i}^{n} u_{ij}x_j) / u_{ii}, i = n,...,1.$$

Розклад LU – це зображення матриці A у вигляді $A = L*U$, елементи яких можна обчислити як $l_{11}=1$, $u_{11} = a_{11}$,

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}, i \le j, i = \overline{1,\ j-1},\ j = \overline{2,n},$$

$$l_{ij} = (a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}) / u_{jj}, i > j,\ j = \overline{1,\ n-1},\ i = \overline{2,\ n}.$$

Відомо, що обчислювальна складність прямого ходу LU-розкладу становить $\frac{2}{3}n^3 + O(n^2)$, а його зворотного ходу – $O(n^2)$ операцій. Отже, кількість операцій, необхідних для розв'язування СЛАР методами Гауса та LU-розкладу, є однакова, проте коли необхідно розв'язати декілька СЛАР з відмінними лише правими частинами, то використання LU-розкладу матиме однозначну перевагу, оскільки розкладення матриці СЛАР потрібно зробити лише один раз, що зменшує кількість обчислень.

Не важко помітити, що на кожному ітераційному кроці виконується декілька однотипних обчислень над рядками матриці коефіцієнтів системи. За основу реалізації паралельного алгоритму методу Гауса та методу LU-розкладу можна взяти принцип паралелізації даних [4].

Алгоритм розкладу матриці A на верхню трикутну на нижню трикутну матриці можна записати у компактній схемі Гауса.

```
for (int i = 1; i <= N; i++)
 {
 for (int j = i; j <= N; j++) // формування верхньої трикутної матриці
  {
   for (int k = 1; k <= i - 1; k++)
    a[i][j] -= a[i][k] * a[k][j];
  }
 for (int j = i + 1; j <= N; j++) // формування нижньої трикутної матриці
  {
   for (int k = 1; k <= i - 1; k++)
    a[j][i] -= a[j][k] * a[k][i];
    a[j][i] /= a[i][i];
  }
 }
```

Легко побачити, що тут є два паралельні цикли, які формують матриці L та U. Для методу Гауса можливе паралельне виконання таких процедур:

- пошук ведучого /провідного?? рядка;
- обчислення ведучого рядка з усіх рядків, які опрацьовують на i-й ітерації;
- виконання зворотного ходу.

Ефективність алгоритму багато в чому залежатиме від доступу до пам'яті, тому нам треба мінімізувати обмін даними між глобальною (Global memory) та локальною пам'яттю (Local memory). Сам процес ітераційного оновлення нижньої трикутної матриці можна виконати трьома паралельними методами доступу до елементів (рис. 1).

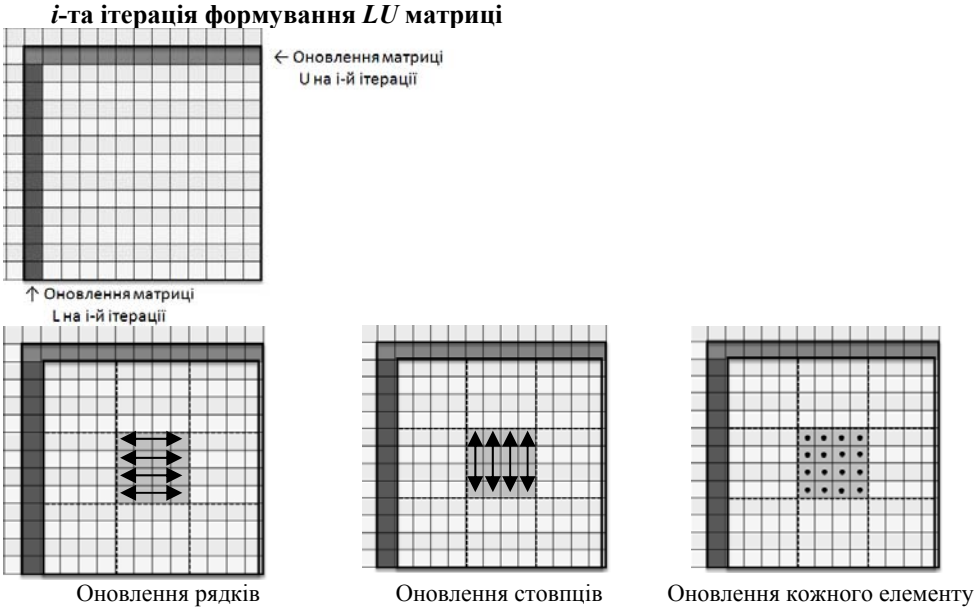

Рис. 1. Методи доступу до елементів.

Метод оновлення рядків реалізують обчисленням кожного елемента рядка окремим потоком графічного процесора, причому можна реалізувати оновлення матриці безпосередньо в глобальній пам'яті, що дасть змогу потокам працювати незалежно та відкине необхідність копіювати дані до та після обчислення. Глобальна пам'ять є найбільшою пам'яттю відеокарти і призначена для зберігання даних від моменту початку роботи графічного процесора до моменту передачі управління центральному процесору (рис. 2). Однак вона є найповільнішою пам'яттю, тому виконувати обчислення, зберігаючи проміжні дані в ній, недоцільно.

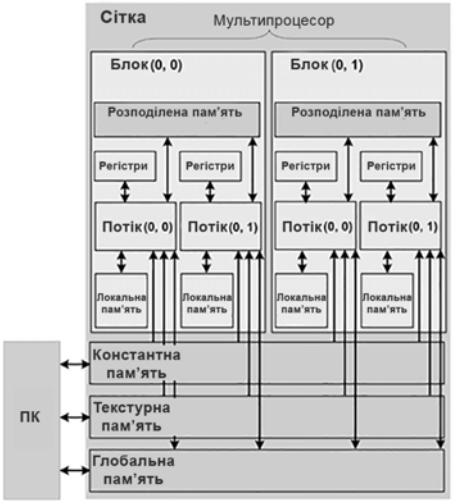

Рис. 2. Ієрархія пам'яті CUDA.

Іншим методом є копіювання даних у розподілену пам'ять під час ініціалізації та подальша робота з нею. Часові затримки копіювання даних лише під час ініціалізації є меншими, ніж затримки, постійного оновлення даних у глобальній пам'яті, однак нам доведеться контролювати використання розподіленої пам'яті потоками для уникнення конфлікту даних, оскільки для збільшення її ефективності розподілену пам'ять використовують усі потоки в межах одного блоку.

Аналогічно можна реалізувати оновлення стовпців, коли кожен потік обчислюватиме один стовпець матриці. Він також реалізується або в глобальній пам'яті, або в розподіленій локальній пам'яті.

Оновлення кожного елемента блоку подібне до попередніх двох способів, однак його доцільно реалізувати лише в глобальній пам'яті, оскільки кожен потік виконує обчислення лише одного коефіцієнта, а не локальної групи, тому копіювання даних до локальної розподіленої пам'яті не буде ефективним і лише зменшить швидкодію методу.

Обчислення виконували на комп'ютері з процесором Intel Core i5 2430m 2.4 GHz та графічному процесорі NVIDIA GeForce GT555m (рис. 3).

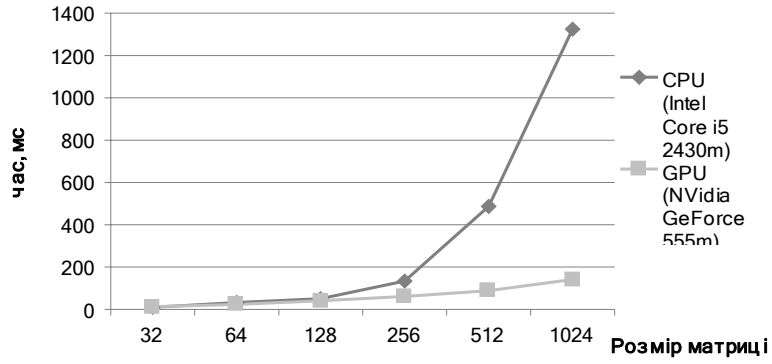

Рис. 3. Результати обчислення методом оновлення кожного елемента.

Метод Холецького [5] застосовують для розв'язування систем з симетричною додатною матрицею, він ґрунтується на розкладенні матриці A в добуток $A = L*L^T$, де L – нижня трикутна матриця з додатними елементами на головній діагоналі.

Після розкладу матриці розв'язок системи зводиться до розв'язування двох систем з трикутними матрицями $L*y = b$, $L^T*x=y$. Елементи матриці L можна обчислити як

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}, \; i = \overline{1, n},$$
$$l_{ji} = \frac{1}{l_{ii}} \left(a_{ji} - \sum_{k=1}^{i-1} l_{ik} l_{jk} \right), \; j = \overline{i+1, n}.$$

Обчислювальна складність для розкладу Холецького наведеним вище методом становить $\frac{2}{3}n^3 + O(n^2)$

При великих n кількість операцій методу Холецького буде приблизно в два рази менша, ніж кількість операцій методу Гауса, що зумовлено симетричністю матриці A.

Алгоритм перетворення матриці A в нижню трикутну матрицю зі строго додатними елементами на діагоналі:

```
for (int i = 0; i < n; i++)
 {
  double temp;
  for (int j = 0; j < i; j++)
   {
    for (int k = 0; k < j; k++)
     {
      a[i][j] -= a[i][k] * a[j][k]
     }
     a[i][j] /= a[j][j];
   }
  for (int k = 0; k < i; k++)
   {
    a[i][i] -= a[i][k] * a[i][k];
   }
  a[i][i] = Math.Sqrt(temp);
 }
```

Основою паралельної модифікації є паралелізація даних, яку можна проводити аналогічно до наведеної вище паралелізації методу Гауса, оновленням рядків, оновленням стовпців та блочним оновленням даних. Для методу Холецького можливе паралельне виконання таких процедур (рис. 4):

- обчислення діагонального елемента l_{ii};
- обчислення елементів i-го рядка;
- виконання оберненого ходу.

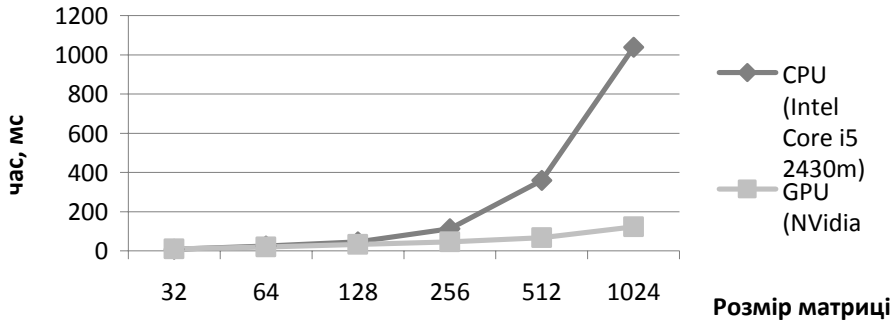

Рис. 4. Результати виконання паралельного методу Холецького.

Отже, досліджено реалізацію паралельних методів розв'язування СЛАР за допомогою технології CUDA. З'ясовано, що для ефективної роботи паралельних алгоритмів можна використовувати особливості архітектури відеокарти, а саме – її локальну розподілену пам'ять. У цьому разі необхідно мінімізувати кількість операцій читання і записування даних під час роботи з пам'яттю [6]. Використання гетерогенної структури з поєднання центрального (CPU) та графічного (GPU) процесорів дає змогу модифікувати

лише ту частину коду, яка є критичною з погляду швидкодії алгоритму, і обчислювати її на графічному процесорі, причому послідовні частини алгоритму, а також операції з попередньої підготовки даних чи їхнього введення–виведення залишити за центральним процесором. Однак зазначимо, що масово-паралельна архітектура CUDA ефективно виявляється лише у випадку, коли кількість паралельних потоків опрацювання даних дуже велика.

Можливим напрямом подальших досліджень є використання оновленої версії архітектури з об'єднаною пам'яттю, дослідження бібліотек для розв'язування задач лінійної алгебри та проведення порівняльного аналізу швидкодії нових власних методів та бібліотечних, зокрема бібліотек, розроблених компанією NVIDIA.

Список використаної літератури

1. *Sanders J.* CUDA by Example: An Introduction to General-Purpose GPU Programming / J. Sanders, E. Kandrot // Addison-Wesley - 2011.
2. *Хвищун І. О.* Програмування і математичне моделювання / І. О. Хвищун. – К. : ІнЮре, 2007.
3. *Гергель В. П.* Теория и практика параллельных вычислений / В. П. Гергель. – М., 2007.
4. *Farber R.* CUDA Application Design and Development / R. Farber // Elsevier - 2011.
5. *Ортега Дж.* Введение в параллельные и векторные методы решения линейных систем Дж. Ортега. – М. : Мир, 1991.
6. *Воеводин В. В.* Параллельные вычисления / В. В. Воеводин, Вл. В. Воеводин. – СПб. : БХВ-Петербург, 2002.

Стаття: надійшла до редакції 05.05.2015
доопрацьована 12.05.2015
прийнята до друку 13.05.2015

PARALLEL METHODS FOR SOLVING LINEAR ALGEBRAIC EQUATIONS USING CUDA

A. Khariv, I. Khvyshchyn

Ivan Franko National University of Lviv,
107 Tarnavsky St., UA–79017, Lviv, Ukraine
andriy.khariv@gmail.com

Mathematical modeling problems from a formal point of view is not a problem because the differential, integral or non-linear systems are often reduced to solving systems of linear algebraic equations that can be solved by Gauss or its modifications. However, when solving systems of linear algebraic equations high order methods for computational complexity is rather high and, along with this is the need to solve these systems in real time.

Key words: CUDA, NVIDIA, parallel computing, graphics processors, mathematical modeling.

ПАРАЛЛЕЛЬНЫЕ МЕТОДЫ РЕШЕНИЯ СЛАУ С ИСПОЛЬЗОВАНИЕМ АРХИТЕКТУРЫ CUDA

А. Харив, И. Хвищун

Львовский национальный университет имени Ивана Франка,
ул. Ген. Тарнавского, 107, 79017 Львов, Украина
andriy.khariv@gmail.com

Задачи математического моделирования, с формальной точки зрения, не вызывают трудностей, поскольку процесс решения систем уравнений математических моделей часто сводится к решению систем линейных алгебраических уравнений, которые решают методом Гаусса или его модификациями. Однако при решении систем линейных алгебраических уравнений высоких порядков вычислительная сложность этих методов значительная, поэтому актуальным является исследование способов применения здесь параллельных алгоритмов с использованием возможностей современных графических процессоров.

Ключевые слова: CUDA, NVIDIA, параллельные вычисления, СЛАУ, графические процессоры, математическое моделирование.