УДК 681.3

А.Ю. Дорошенко, О.Г. Бекетов

МЕТОД ПАРАЛЕЛІЗАЦІЇ ЦИКЛІВ СІТКОВИХ ОБЧИСЛЮВАЛЬНИХ ЗАДАЧ ДЛЯ ГРАФІЧНИХ ПРИСКОРЮВАЧІВ

Розроблено формальне перетворення гнізда обчислювального циклу, що дозволяє здійснити перехід від послідовного алгоритму до паралельного, орієнтованого на виконання на пристрої з SIMD архітектурою, зокрема, на графічному прискорювачі із використанням технології CUDA та на гетерогенних кластерах.

Ключові слова: методи паралелізації, оптимізація циклів, обчислення загального призначення на графічних процесорах.

Вступ

Сучасні обчислювальні комплекси мають гетерогенну архітектуру і включають як багатоядерні процесори, так і графічні прискорювачі. Проте велику кількість існуючих програмних засобів створено для послідовного виконання на одноядерних машинах. Крім того, створення нових програмних засобів здебільшого проходить через етап послідовної програми. Таким чином, задача автоматичної паралелізації має велику актуальність. В більшості обчислювальних задач значну частину апаратних ресурсів витрачають обчислення, що здійснюються всередині циклів, тому використання автоматичної паралелізації на рівні потоків найбільш ефективне саме для них. Так, наприклад, більшість задач математичної фізики розв'язуються чисельними методами з використанням сіткових обчислень. До виникнення циклів призводять різницеві схеми, метод скінченних елементів, загалом, задачі, що вимагають виконання операцій над матрицями даних. Типовий для сіткової задачі обчислювальний цикл має вигляд:

$$
\begin{aligned}
&for \quad i_0 \in I_0 \\
&for \quad i_1 \in I_1 \\
&\quad \ldots \\
&\quad for \quad i_N \in I_N \\
&\quad\quad F(data, i_0, \ldots, i_N); \qquad (1)
\end{aligned}
$$

де $data$ – множина даних, що обробляються, $I_0, I_1, \ldots, I_N$ – множини значень індексів $i_0, i_1, \ldots, i_N$, $N+1$ – глибина вкладеності циклу, $data$ – множина даних, $F : (data, I_0, \ldots, I_N) \mapsto data$ – відображення, що здійснює перетворення даних. Цикли такого роду у випадку відсутності залежності між вхідними та вихідними даними природним чином розкладаються на паралельні потоки обчислень, оскільки на різних ітераціях цикл повторює однотипні операції над різними даними. Par4All [1] – програмний засіб, що дозволяє на рівні коду в автоматичному режимі здійснити перехід від послідовних програм, написаних мовами C та Fortran, до паралельних для різних апаратних платформ, в тому числі графічного прискорювача із використанням CUDA API. Par4All використовує механізм сіток для зниження глибини вкладеності циклів, проте не надає можливості обробляти обсяги даних, що перевищують обсяг пам'яті прискорювача, а також використовувати гетерогенні обчислювальні кластери.

1. Зміна індексації ітераторів циклу

Розглянемо складений цикл глибиною $N+1$, що має вигляд

$$
for \quad i_0 = 0..\#I_0
$$

$$
\begin{aligned}
&for \quad i_1 = 0..\#I_1 \\
&\quad ... \\
&\quad\quad for \quad i_N = 0..\#I_N \\
&\quad\quad\quad F(in, out, i_0, ..., i_N),
\end{aligned} \qquad (2)
$$

де in – множина вхідних даних, out – множина вихідних даних, причому перетворення $F : (in, I_0, ..., I_N) \mapsto out$ таке, що для кожного набору індексів $(i_0, ..., i_N)$ із гіперкуба $I_0 \times ... \times I_N$ виконується умова $in \cap out \subset \varnothing$ (*), а символом октоторпа позначено потужність відповідної множини. Таким чином, для проведення обчислень цикл (2) здійснює $C = \#(I_0 \times ... \times I_N)$ викликів F. Не обмежуючи загальності, цикл вигляду (2) можна вважати рівносильним циклу (1).

Пронумеруємо елементи гіперкуба $I_0 \times ... \times I_N$ наступним чином. Нехай елемент, що відповідає набору індексів $(i_0, i_1, ..., i_N)$, має номер $i_0 + \sum_{k=1}^{N} i_k \prod_{m=0}^{k-1} \#I_m$.

Введемо відображення $f(k)$, $0 \le k \le \#(I_0 \times ... \times I_N)$:

$$
f(k) = \left(D - \sum_{j=0}^{k-1} i_j \prod_{l=j+1}^{N} \#I_l \right) \Bigg/ \prod_{j=k+1}^{N} \#I_j, \quad k \ne 0,
$$

$$
f(0) = \left(D - \sum_{j=0}^{k-1} i_j \prod_{l=j+1}^{N} \#I_l \right) \bmod \prod_{j=k+1}^{N} \#I_j,
$$

$$
D = \prod_{k=0}^{N} I_k .
$$

Відображення $f(k)$ за номером елемента гіперкуба $I_0 \times ... \times I_N$ відновлює відповідний індекс i_k. Розглянемо цикл такого вигляду:

$$
\begin{aligned}
&for \quad k = 0..D \\
&\quad for \quad j = 0..N \\
&\quad\quad i_j' = f(j); \\
&\quad\quad F(in, out, i_0', ..., i_N').
\end{aligned} \qquad (3)
$$

При проходженні зовнішнього циклу в (3) індекси i_k пробігають ті самі значення, що і в (2), тому цикли (2) та (3) є рівносильними.

Зафіксуємо $0 < d < N$ і введемо відображення $g(k, id)$, $0 \le k \le N$, $0 \le id \le \prod_{j=0}^{d-1} \#I_j$:

$$
g(k, e) = \left(id - \sum_{j=0}^{k-1} i_j \prod_{l=j+1}^{d-1} \#I_l \right) \Bigg/ \prod_{j=k+1}^{d-1} \#I_j, \quad k \ne d-1,
$$

$$
g(d-1, e) = \left(id - \sum_{j=0}^{d-2} i_j \prod_{l=j+1}^{d-1} \#I_l \right) \Big/ \#I_{d-1} .
$$

Відображення $g(k, e)$ побудоване таким чином, що при проходженні наступного циклу індекси i_k пробігають ті самі значення, що і в (2):

$$
\begin{aligned}
&for \quad e = 0..\prod_{j=0}^{d-1} \#I_j \\
&\quad for \quad id = 0..\prod_{m=d}^{N} \#I_m \\
&\quad\quad for \quad k = d..N \\
&\quad\quad\quad i_j' = f(k); \\
&\quad\quad for \quad k = 0..d-1 \\
&\quad\quad\quad i_k' = g(k, id); \\
&\quad\quad\quad F(in, out, i_0', ..., i_N'),
\end{aligned} \qquad (4)
$$

тому останній наведений цикл (4) також рівносильний циклу (2). При цьому ітератори циклів, що мають глибину меншу, ніж d, лишаються сталими впродовж однієї ітерації внутрішнього циклу.

2. Підготовка початкових даних

Оскільки на F діє обмеження (*), операції над вхідними даними є незалежними, і можуть бути виконані паралель-

ними потоками. Припустимо, що F таке, що не містить умовних переходів, тоді різні потоки будуть здійснювати однотипні операції над різними даними і для виконання обчислень доцільно використовувати апаратну SIMD платформу. Пронумеруємо потоки символом $id \in [0, D]$. У випадку (3) кожен із потоків виконує окрему гілку зовнішнього циклу. У випадку (4) зовнішній цикл здійснюється окремим керуючим потоком. Таким чином, за допомогою заміни індексів циклу здійснюється перехід до багатопоточного алгоритму, при цьому не змінюючи F.

Крім того, у випадку, коли в початковому алгоритмі програми використовуються багатовимірні масиви даних, враховуючи той факт, що читання із пам'яті відбувається швидше із лінійних ділянок пам'яті, для оптимізації обробки великих обсягів даних доцільно використовувати перетворення серіалізації, що справедливо навіть для SISD пристроїв, не враховуючи витрат, необхідних для серіалізації даних.

3. Загальна схема перетворення алгоритму

Перехід від послідовного до паралельного алгоритму програми здійснюється у кілька етапів. Загальна схема переходу виглядає так.

1. Підготовчий етап. Полягає у зведенні вихідного алгоритму до циклу вигляду (2). При цьому слід позбавитись міжітераційних залежностей. На цьому етапі можуть використовуватись розбиття циклів, їх перестановка, афінні перетворення, попередній розрахунок даних тощо.

2. Визначення обсягу даних, що обробляються та кількості незалежних операцій, що виконуються всередині окремих циклів гнізда. Ці параметри можливо визначити за умови відсутності умовних переходів в обчисленнях.

3. Визначення параметра d згідно апаратних можливостей виконуючого пристрою. Параметр d обирається таким чином, щоб, з одного боку, обсяг даних, що обробляється циклом глибини d, не перевищував обсяг пам'яті виконуючого пристрою, і з іншого, щоб найбільш повно задіяти наявні апаратні потоки пристрою.

4. Заміна ітераторів згідно правил, описаних в розділі 1. При цьому програма залишається послідовною, змінюється лише спосіб індексації даних. Після цього перетворення залишаються лише два цикли – зовнішній (керуючий) та внутрішній.

5. Створення модуля серіалізації вхідних даних, який підготовляє порцію даних, що буде оброблятися поточною ітерацією керуючого циклу.

6. Створення модуля десеріалізації вихідних даних. Модуль приймає буфер вихідних даних, отриманих після виконання обчислень на поточній ітерації, та передає дані для подальшого використання.

7. Перехід до буферної схеми перенесення даних. На цьому етапі залучаються серіалізатор та десеріалізатор, що викликаються до початку та після виконання внутрішнього циклу відповідно. Змінні внутрішнього циклу переадресовуються на відповідні ділянки вхідного та вихідного буферів даних.

8. Створення ядра, тобто частини програми, що буде безпосередньо перенесена на виконуючий пристрій. Ядро є функцією, що приймає буфер вхідних даних та номер ітерації зовнішнього циклу, та повертає буфер вихідних даних. Функціонально ядро має виконувати ті самі операції, що і внутрішній цикл. Ітератор id внутрішнього циклу замінюється номером потоку виконуючого пристрою.

9. Залучення ядра. Внутрішній цикл замінюється на директиву виклику ядра. Також додаються операції перенесення вхідного буферу даних до виконуючого пристрою та повернення вихідного буферу даних назад в центральний обчислювальний пристрій.

10. Винесення роботи із буферними даними та з графічним прискорювачем в окремі потоки керуючого пристрою з метою уникнення простою виконуючого пристрою.

Проміжні етапи 4, 7, а також фінальні 9, 10 потребують тестових випробувань, які проводяться за допомогою засобів перевірки коректності результатів виконання обчислень. Наприклад, це може бути обробка тестового набору даних і порівняння із наперед відомими результатами. Етап 5 може бути здійснений автоматизовано за допомогою техніки переписуваних правил [2], що дозволить суттєво спростити здійснення перетворень. Для здійснення перетворень на етапі 9 доцільно задіяти Інтегрований інструментарій Проектування та Синтезу програм (ІПС) [3], що зокрема був налаштований на роботу із CUDA API [4, 5].

Роль параметра d циклу (4) полягає в тому, що він визначає глибину, починаючи з якої цикли виконуються безпосередньо виконуючим пристроєм. Таким чином, d встановлює кількість викликів та розподіл навантаження обчислювального ядра внутрішнього циклу. Така техніка надає перевагу в тому разі, якщо об'єм даних, що обробляються, перевищує ємність обчислювального пристрою. При значеннях $0 < d < N$ цикл (2) розбивається на $P = \prod_{i=0}^{d-1} N_i$ частин, кожна з яких оброблюється окремо. Відповідно, керуючий пристрій через змінну $0 \le e \le P$ передає виконуючому пристою номер ітерації, та необхідну на даній ітерації порцію вхідних даних. Порції вхідних даних підготовлюються окремим потоком. В свою чергу, обчислюючий пристрій за допомогою відображень f, g із e відновлює індекси елементів поданої порції даних, що відповідають ітерації зовнішнього керуючого циклу. Таким чином, при значенні $d = 0$ всі операції у циклі (2) будуть здійснюватись виконуючим пристроєм за один виклик ядра, і всі оброблювані дані мають бути передані виконуючому пристрою одночасно, і тоді випадок (4) приймає вигляд (3).

Проілюструємо наведений ланцюжок перетворень на прикладі. Розглянемо частковий випадок циклу вигляду (2) глибини 5:

$$
\begin{array}{l}
for \quad l = 0..L \\
\quad for \quad m = 0..M \\
\quad\quad for \quad n = 0..N \\
\quad\quad\quad for \quad k = 0..K \\
\quad\quad\quad\quad for \quad j = 0..J \\
\quad\quad\quad\quad\quad F(i_{lmnkj}, o_{lmnkj});
\end{array}
$$

де L, M, N, K, J – натуральні числа, i_{lmnkj}, o_{lmnkj} – елементи множин вхідних та вихідних даних ***I, O*** відповідно.

Обсяг вхідних даних рівний обсягу вихідних даних для кожного із циклів і становить $L \cdot M \cdot N \cdot K \cdot J \cdot s$, $M \cdot N \cdot K \cdot J \cdot s$, $N \cdot K \cdot J \cdot s$, $K \cdot J \cdot s$, $J \cdot s$ починаючи із зовнішнього циклу відповідно; s позначає розмір елемента даних. Оберемо параметр d рівним двом та виконаємо переіндексацію даних. Тоді останній цикл набуде наступного вигляду:

$$
\begin{array}{l}
for \quad e = 0..L \cdot M \\
\quad for \quad id = 0..N \cdot K \cdot J \\
\quad\quad l = e / T; \\
\quad\quad m = e \vdots T; \\
\quad\quad n = (id - l \cdot M \cdot N \cdot K \cdot J - \\
\quad\quad\quad - m \cdot N \cdot K \cdot J) / (K \cdot J); \\
\quad\quad k = (id - l \cdot M \cdot N \cdot K \cdot J - \\
\quad\quad\quad - m \cdot N \cdot K \cdot J - n \cdot K \cdot J) / J; \\
\quad\quad j = (id - l \cdot M \cdot N \cdot K \cdot J - \\
\quad\quad\quad - m \cdot N \cdot K \cdot J \\
\quad\quad\quad - n \cdot K \cdot J - k \cdot J) \vdots J; \\
\quad\quad F(i_{lmnkj}, o_{lmnkj}).
\end{array}
$$

Відображення-серіалізатор $Serialize\ (I,\ e,\ inbuf)$ будується наступним чином:

$$
\begin{array}{l}
l = e / T; \\
m = e \vdots T; \\
for \quad n = 0..N \\
\quad for \quad k = 0..K
\end{array}
$$

$$for \quad j = 0..J$$
$$index = n \cdot K \cdot J + k \cdot J + j;$$
$$inbuf_{index} = i_{lmnkj};$$

де $inbuf$ – буфер вхідних даних. Дані вміщуються у єдиний буфер, що згодом дасть зручну можливість переносити дані в виконуючий пристрій однією операцією перенесення.

Десеріалізатор

$$Deserialize\ (e,\ outbuf,\ O)$$

будується аналогічно серіалізатору:

$$l = e / T;$$
$$m = e \vdots T;$$
$$for \quad n = 0..N$$
$$for \quad k = 0..K$$
$$for \quad j = 0..J$$
$$index = n \cdot K \cdot J + k \cdot J + j;$$
$$o_{lmnkj} = outbuf_{index};$$

де $outbuf$ – буфер вихідних даних.

Після впровадження серіалізатора та десеріалізатора цикл набуває наступного вигляду:

$$for \quad e = 0..L \cdot M$$
$$Serialize\ (I,\ e,\ inbuf);$$
$$for \quad id = 0..N \cdot K \cdot J$$
$$l = e / T;$$
$$m = e \vdots T;$$
$$n = (id - l \cdot M \cdot N \cdot K \cdot J - m \cdot N \cdot K \cdot J) / (K \cdot J);$$
$$k = (id - l \cdot M \cdot N \cdot K \cdot J - m \cdot N \cdot K \cdot J - n \cdot K \cdot J) / J;$$
$$j = (id - l \cdot M \cdot N \cdot K \cdot J - m \cdot N \cdot K \cdot J - n \cdot K \cdot J - k \cdot J) \vdots J;$$
$$F(inbuf_{nkj}, outbuf_{nkj});$$
$$Deserialize\ (e,\ outbuf,\ O).$$

Ядро $Kernel(e,\ inbuf,\ outbuf)$ виконує ті самі операції, що і внутрішній цикл:

$$id = threadId();$$
$$if \quad id < N \cdot K \cdot J$$
$$l = e / T;$$
$$m = e \vdots T;$$
$$n = (id - l \cdot M \cdot N \cdot K \cdot J - m \cdot N \cdot K \cdot J) / (K \cdot J);$$
$$k = (id - l \cdot M \cdot N \cdot K \cdot J - m \cdot N \cdot K \cdot J - n \cdot K \cdot J) / J;$$
$$j = (id - l \cdot M \cdot N \cdot K \cdot J - m \cdot N \cdot K \cdot J - n \cdot K \cdot J - k \cdot J) \vdots J;$$
$$F(inbuf_{nkj}, outbuf_{nkj}),$$

де процедура $threadId()$ повертає номер потоку. Таким чином, після всіх перетворень вихідний цикл набуває остаточного вигляду:

$$for \quad e = 0..L \cdot M$$
$$Serialize\ (I,\ e,\ inbuf);$$
$$CopyInput\ (inbuf,\ inbuf');$$
$$Kernel(e,\ inbuf',\ outbuf');$$
$$CopyOutput\ (outbuf',\ outbuf);$$
$$Deserialize\ (e,\ outbuf,\ O);$$

де процедури *CopyInput* та *CopyOutput* переносять буфер вхідних та буфер вихідних даних від керуючого до обчислювального пристрою та в зворотному порядку відповідно. Процедуру *Serialize* доцільно винести в окремий потік керуючого пристрою. Для цього створюється окремий додатковий буфер, і робота *Serialize* виконується почергово:

$$Serialize\ (I,\ e,\ inbuf_0);$$
$$for \quad e = 0..L \cdot M$$
$$thread_0:$$

Serialize (*I*, *e*, $inbuf_{(e+1):2}$);

$thread_1$:

CopyInput ($inbuf_{e:2}$, $inbuf'$);

Kernel(*e*, $inbuf'$, $outbuf'$);

CopyOutput ($outbuf'$, $outbuf_{e:2}$);

$thread_3$:

Deserialize (*e*, $outbuf_{(e+1):2}$, *O*).

Крім того, для проведення обчислень можливо залучити гетерогенний кластер, до складу якого входять кілька SIMD пристроїв. В такому випадку окремі ітерації викликів *Kernel*() керуватимуться окремими потоками керуючого пристрою.

4. GPU та технологія CUDA

Як обчислювальний пристрій будемо використовувати графічний прискорювач. В такому випадку керуючий потік виконується на CPU, а решта потоків переносяться на GPU та виконуються в межах ядра. Для роботи з графічним прискорювачем будемо використовувати програмно-апаратну технологію CUDA [6], що дозволяє використовувати GPU як обчислювальний пристрій. CUDA відрізняється від інших GPGPU технологій тим, що надає безпосередній доступ до пристрою через програмний інтерфейс, що дозволяє гнучко керувати окремими потоками та всіма рівнями пам'яті графічного прискорювача. Крім того, саме для архітектури CUDA створюються спеціалізовані графічні прискорювачі для наукових та технічних обчислень загального призначення [7]. CUDA накладає обмеження на розмірність масивів даних, що не має перевищувати трьох. Це обмеження оминається на етапі серіалізації даних. Проблема, що виникає у тому випадку, якщо необхідна для проведення обчислень кількість потоків більша, ніж кількість апаратних потоків графічного прискорювача, автоматично оминається механізмом сіток CUDA. Таким чином, при використанні графічних прискорювачів, що дозволяють використання технології CUDA, основним обмеженням виступає обсяг пам'яті графічного прискорювача.

5. Експеримент

Проведено експеримент із циклом, що був взятий у задачі прогнозування погоди [8, 9]. Гніздо циклу складене із чотирьох вкладених циклів і обробляє чотиривимірний масив даних числових значень типу float. Навантаження масштабується шляхом зміни розмірності задачі. Було здійснено перехід від послідовної реалізації циклу до паралельної із застосуванням описаної методології та порівняно часові показники для різного масштабу навантаження та різних значень параметра *d*.

Випробування проводились із використанням процесора i5-3570 (у 64-бітному режимі) та графічного прискорювача GeForce GTX 650 Ti, що має наступні характеристики:

- 768 stream-процесори (CUDA-ядра), базова частота 928 MHz;
- обсяг глобальної пам'яті 1024 Mb;
- базова частота пам'яті 5400 MHz.

Обсяг пам'яті ОЗП становив 8 Гб.

При випробуваннях досліджено поведінку послідовної та паралельної програм при фіксованому значенні параметра d, та зміні розмірності задачі, що відповідає зміні кількості викликів ядра при сталому навантаженні на нього, та при зміні розмірності задачі, що відповідає зміні навантаження на один виклик ядра.

Графік на рисунку показує залежність часу виконання обчислень від обсягу даних для $d = 1$, при сталому навантаженні на ядро. При цьому відносне прискорення є сталим та становить близько 4,2.

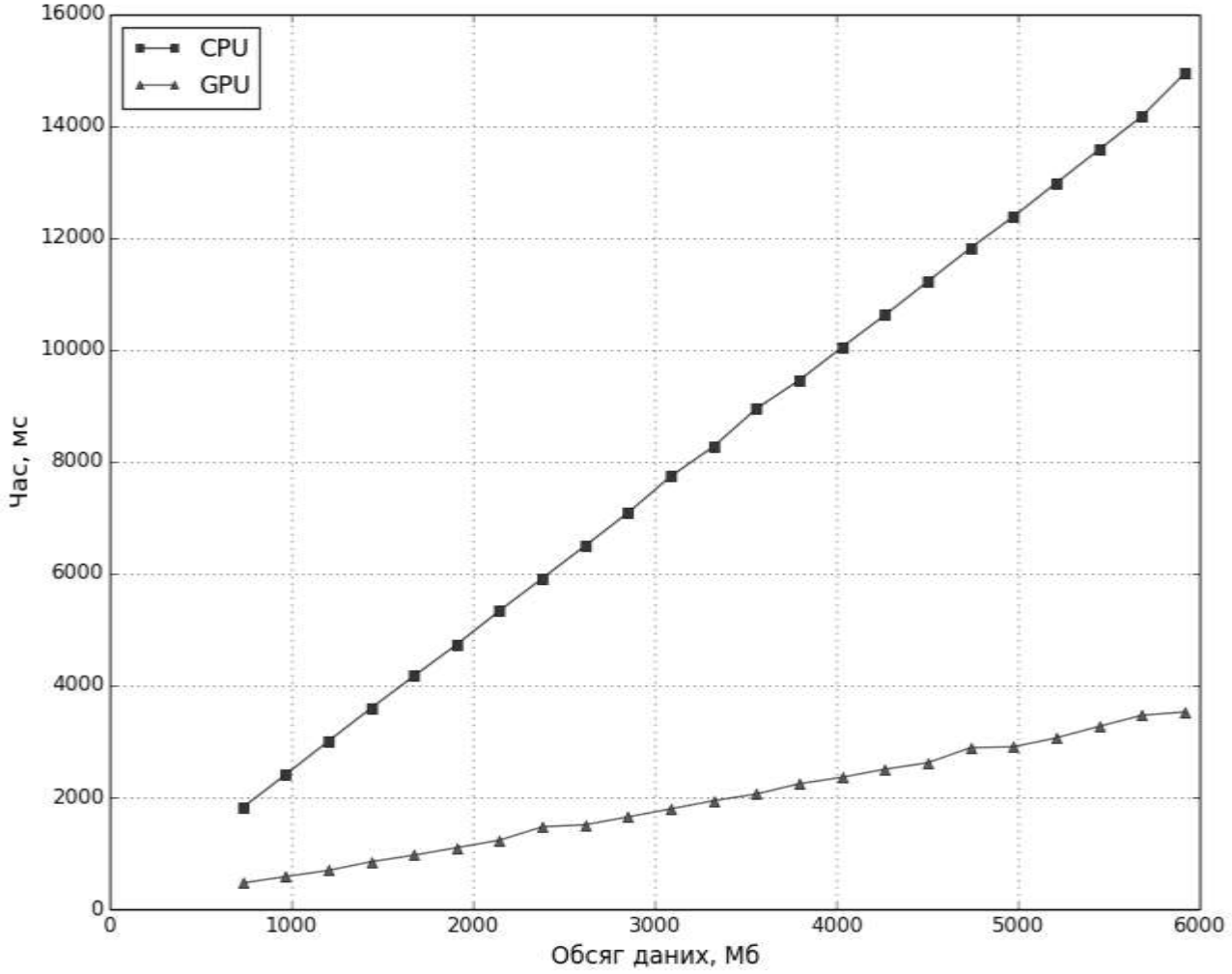

Рисунок. Графік залежності часу виконання послідовної та паралельної програм від обсягу даних, що обробляються

Висновки

Побудовано перетворення алгоритму виконання складеного циклу, що дозволяють здійснити перехід від послідовного до паралельного алгоритму проведення обчислень, що дозволяє залучення гетерогенної обчислювальної платформи. Перевагою застосування методу є те, що він дозволяє здійснювати перетворення над даними, обсяг яких перевищує обсяг пам'яті виконуючого пристрою, а також є автоматизовним. Проведено експеримент, що підтверджує доцільність запропонованого підходу. Таким чином, розроблено основу для подальшої практичної реалізації автоматизованої системи паралелізації вкладених циклів.

1. *Дорошенко А.Е., Шевченко Р.С.* Система символьных вычислений для программирования динамических приложений. Проблеми програмування. 2005. № 4. С. 718–727.
2. *Яценко Е.А.* Интеграция инструментальных средств алгебры алгоритмов и переписывания термов для разработки эффективных параллельных программ. Проблеми програмування. 2013. № 2. С. 62–70.
3. *Дорошенко А.Ю., Бекетов О.Г., Прусов В.А., Тирчак Ю.М., Яценко О.А.* Формалізоване проектування та генерація паралельної програми чисельного прогнозування погоди. Проблеми програмування. 2014. № 2–3. С. 72–81.
4. *Дорошенко А.Ю., Бекетов О.Г., Іванів Р.Б., Іовчев В.О., Мироненко І.О., Яценко О.А.* Автоматизована генерація паралельних програм для графічних прискорювачів на основі схем алгоритмів. Проблеми програмування. 2015. № 1. С. 19–28.

5. CUDA [Електронний ресурс] – Режим доступу до ресурсу: http://www.nvidia.com/object/cuda_home_new.html.
6. TESLA [Електронний ресурс] – Режим доступу до ресурсу: http://www.nvidia.com/object/tesla-servers.html.
7. PIPS: Automatic Parallelizer and Code Transformation Framework [Електронний ресурс] – Режим доступу до ресурсу: http://pips4u.org/.
8. *Прусов В.А., Дорошенко А.Ю.* Моделювання природних і техногенних процесів в атмосфері. Київ: Наукова думка. 2006. 542 с.
9. *Прусов В.А., Сніжко С.І.* Математичне моделювання атмосферних процесів. Київ: Ніка-Центр. 2005. 496 с.

References

1. Doroshenko, A.Yu. & Shevchenko R.S. (2005) Symbolic computation system for dynamical application programming. Problems in programming. (4). P. 718–727. (in Russian)
2. Yatsenko, O.A. (2013) Integration of Software Tools of Algebra of Algorithms and Rewriting Terms for Development of Effective Parallel programs. Problems in programming. (2). P. 62–70. (in Russian).
3. Doroshenko, A.Yu., Beketov, O.G , Prusov, V.A., Tyrchak, Yu.M. & Yatsenko, O.A. (2014) Formalized design and generation of parallel programs for numerical weather forecast. Problems in programming. (2-3). P. 72–81. (in Ukrainian).
4. Doroshenko, A.Yu., Beketov, O.G., Ivaniv, R.B., Iovchev, V.O., Mironenko, I.O. & Yatsenko, O.A. (2015) Automated generation of parallel programs for graphics processing units based on algorithm schemes. Problems in programming. (1). P. 19–28. (in Ukrainian).
5. CUDA [Online] – Available from: http://www.nvidia.com/object/cuda_home_new.html.
6. TESLA [Online] – Available from: http://www.nvidia.com/object/tesla-servers.html.
7. PIPS: Automatic Parallelizer and Code Transformation Framework [Online] – Available from: http://pips4u.org/.
8. Prusov V.A. & Doroshenko A.Yu. (2006) Simulation of natural and anthropogenic processes in the atmosphere. Kyiv: Naukova Dumka. (in Ukrainian).
9. Prusov, V.A. & Snizhko, S.I. (2005) Mathematical modeling of atmospheric processes. Kyiv: NikaTsentr. (in Ukrainian).

Одержано 01.02.2017

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень,
професор кафедри автоматики
та управління в технічних системах
НТУ України "КПІ".
Кількість наукових публікацій в
українських виданнях – понад 200.
Кількість наукових публікацій в
зарубіжних виданнях – понад 50.
Індекс Хірша – 5.
http://orcid.org/0000-0002-8435-1451,

Бекетов Олексій Геннадійович,
молодший науковий співробітник.
Кількість наукових публікацій в
українських виданнях – 10.
Кількість наукових публікацій в
зарубіжних виданнях – 1.
http://orcid.org/0000-0003-4715-5053.

Місце роботи авторів:

Інститут програмних систем
НАН України.
проспект Академіка Глушкова, 40.
03187, Київ,
НТУ України «КПІ»
Тел.: (044) 526 3559.
E-mail: dor@isofts.kiev.ua,
beketov.oleksii@gmail.com