Оптимизация вычислений

Розглядається новий гібридний алгоритм розв'язування систем лінійних алгебраїчних рівнянь із стрічковими симетричними додатно визначеними матрицями на комп'ютерах з графічними прискорювачами. Подано результати апробації алгоритму на багатоядерному комп'ютері з графічними прискорювачами Інпарком.

УДК 519.6

О.М. ХІМІЧ, А.Ю. БАРАНОВ

ГІБРИДНИЙ АЛГОРИТМ РОЗВ'ЯЗУВАННЯ ЛІНІЙНИХ СИСТЕМ ІЗ СТРІЧКОВИМИ МАТРИЦЯМИ ПРЯМИМИ МЕТОДАМИ

Вступ. При чисельному розв'язанні задач у багатьох випадках виникає необхідність розв'язувати задачу (або декілька підзадач) лінійної алгебри – систему лінійних алгебраїчних рівнянь (СЛАР). Наприклад, задачі лінійної алгебри виникають при дискрети-зації крайових задач чи задач на власні значення проекційно-різницевим методом (скінченних елементів, скінченних різниць). Також, при використанні ітераційних методів розв'язання нелінійних задач часто на кожній ітерації розв'язується лінеаризована задача – СЛАР.

Важливою особливістю задач лінійної алгебри, які виникають при дискретизації, являється те, що кількість ненульових елементів матриць таких задач складає kn, де $k << n$, а n – порядок матриці, тобто матриці є розрідженими [1]. Структура розрідженої матриці визначається нумерацією невідомих задач і часто є стрічковою, блочно-діагональною з обрамленням, профільною і т. п. В даній роботі розглядаються симетричні додатно визначені матриці стрічкової структури.

Іншою важливою особливістю є великий порядок матриця СЛАР – до десятків мільйонів. Це зумовлюється бажанням використовувати більш точні дискретні моделі, що дає можливість отримувати наближені розв'язки більш близькі до розв'язків вихідних задач, враховувати локальні особливості даного процесу чи явища.

Розв'язання таких задач потребує значних обчислювальних ресурсів, які забезпечуються використанням багатоядерних комп'ютерів. Однак, уже недостатньо продуктивності багатоядерних архітектур. Розв'язання проблеми нарощування обчислювальних ресурсів лежить у площині використання гібридних (MIMD, SIMD) архітектур. Пропонується алгоритм розв'язання СЛАР із стрічковими симетричними додатно визначеними матрицями, який використовує обчислювальні можливості CPU та GPU.

1. Постановка задачі

Розглянемо систему лінійних алгебраїчних рівнянь:

$$Ax = b, \tag{1}$$

де матриця A – стрічкова симетрична додатно визначена, n – порядок матриці A, m – напівширина стрічки матриці.

Найбільш ефективним прямим методом розв'язання такої задачі є, як відомо, метод Холецького [2]. Розв'язання системи (1) полягає у розв'язанні підзадач: трикутне розвинення матриці системи, розв'язання двох СЛАР з трикутними матрицями:

$$A = L * L^T, \tag{2}$$

$$Ly = b, \tag{3}$$

$$L^T x = y. \tag{4}$$

2. Гібридний алгоритм факторизації Холецького

При розробці програм для гібридних систем важливими етапами реалізації алгоритму є вибір ефективного способу задання даних та врахування особливостей архітектури гібридної системи.

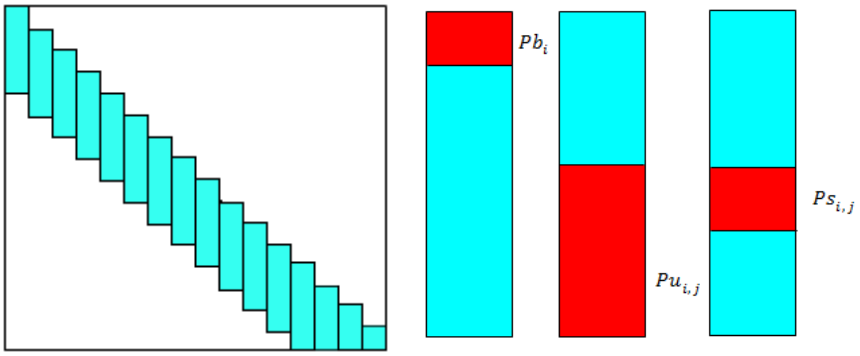
Для систем з багатьма графічними прискорювачами необхідними є операції обміну даними та синхронізації. Ці операції знижують ефективність алгоритму у випадку багатьох прискорювачів. Тому, доцільним є створення різних версій алгоритму для систем з одним та багатьма графічними прискорювачами.

2.1. Декомпозиція даних. При розробці алгоритмів розв'язання задач з розрідженими матрицями особливе значення має вибір способів задання та збереження ненульових елементів. Ці способи визначаються структурою (розміщення ненульових елементів) розрідженої матриці та потребами алгоритму розв'язання задачі.

Розіб'ємо стрічкову матрицю на вертикальні прямокутні плитки [3]. Оскільки матриця симетрична, то доцільно зберігати тільки нижню частину матриці системи, що дає можливість оптимізувати обсяг пам'яті необхідний для реалізації алгоритму. Також таке розбиття даних дає можливість більш ефективно використовувати кеші CPU та GPU, з'являється можливість використовувати високоефективні функції cuBLAS. Іншою важливою особливістю такого задання даних є те, що на кожному кроці блочного алгоритму Холецького потрібно буде мати в пам'яті тільки m/w плиток, де w – ширина плитки, які будуть розташовані послідовно за плиткою з ведучим блоком. Таким чином, така декомпозиція даних дає можливість на кожному кроці алгоритму копіювати в пам'ять GPU лише одну додаткову плитку.

Також введемо деякі позначення, які будуть потрібні для опису алгоритму трикутного розвинення (схематично показано на рис. 1, б):

- Pb_i – діагональний блок плитки з номером i;
- $Pu_{i,j}$ – підматриця плитки i, що починається з рядка $j*w$;
- $Ps_{i,j}$ – квадратна підматриця плитки i, що починається з рядка $j*w$.



а б

РИС. 1. Декомпозиція даних

2.2. Плитковий алгоритм факторизації для гібридних систем з одним GPU. При реалізації алгоритму Холецького використовуються три основні операції:

- POTRF – факторизація Холецького щільної матриці. В даному алгоритмі ця операція буде використовуватися для факторизації діагонального блоку плитки Pb_i;
- TRSM – розв'язування СЛАР з трикутною матрицею. Використовується для розв'язування рівняння $X * Pb_i = Pu_{i,1}$;
- GEMM – матрично-матрична операція. Використовується для оновлення плиток матриці системи, які розташовані за плиткою з ведучим блоком:

$$Pu_{i+j,0} \leftarrow Pu_{i+j,0} - Pu_{i,j} * (Ps_{i,j})^T. \quad (5)$$

Для виконання перших двох операцій, на i-му кроці алгоритму, потрібно використовувати лише дані, які знаходяться в i-й плитці. Для операції GEMM, окрім i-ї плитки, також використовуються плитки, які розташовуються послідовно за i-ю плиткою. Отже, кількість плиток, які використовуються на i-му кроці алгоритму дорівнює m/w. На кожному наступному кроці в пам'ять GPU потрібно копіювати одну додаткову плитку.

Доцільним є виконувати копіювання плитки, яка буде потрібна на наступному кроці, одночасно з виконанням операції GEMM. Таким чином, можливо досягти більшої ефективності алгоритму. Для того, щоб на кожному кроці не виділяти додаткову пам'ять для наступної плитки, потрібно на початку роботи

алгоритму виділити пам'ять для зберігання $m/w+1$ плитки. На кожному кроці m/w плиток будуть використовуватися для обчислень, а наступна плитка буде копіюватися на вільний фрагмент пам'яті. Також, значно зменшено використання пам'яті GPU у порівняння з алгоритмами, коли вся матриця зберігається на GPU.

Розіб'ємо реалізацію алгоритму на два етапи: на першому – виконується початкова ініціалізація, на другому – обчислюється трикутне розвинення матриці системи.

Перший етап:

- ініціалізація cuBLAS;
- створення двох екземплярів cudaStream_t: cudaCopyStream, cudaCalculateStream;
- потік cudaCalculateStream використовується як основний потік cuBLAS;
- виділення пам'яті для $m/w+1$ плитки на GPU. Вказівник на цю пам'ять gpuTilesMemory;
- копіювання перших m/w в плиток пам'ять GPU.

Варто зазначити, що доцільним є виділення одного фрагменту пам'яті для збереження $m/w+1$ плиток в GPU, що дає можливість швидше виділити всю необхідну пам'ять.

На другому етапі, для кожного $i=1...,n/w$ виконуємо наступні кроки алгоритму:

- факторизація діагонального блоку i-ї плитки Pb_i. Результат записується в Pb_i;
- перетворення на GPU нижньої частини плитки $Pu_{i,1}$ за формулою $Pu_{i,1} \leftarrow Pu_{i,1} * (Pb_i)^{-1}$. Використовується функція cublasDtrsm;
- асинхронне копіювання в потоці cudaCopyStream i-ї плитки в пам'ять CPU;
- асинхронне копіювання плитки, яка потрібна на наступному кроці в gpuTilesMemory, використовуючи cudaCopyStream;
- виконати цикл по j від 1 доки виконується нерівність $j * w < len[i]$; на кожній ітерації виконати перетворення (5), використовуючи функцію cublasDgemm;
- синхронізація потоку cudaCopyStream та збереження i-ї плитки на CPU;
- синхронізація основного потоку cuBLAS – cudaCalculateStream;

Після виконання всіх кроків алгоритму отримаємо нижню трикутну матрицю, для якої справджується формула (2).

2.3. Плитковий алгоритм факторизації для гібридних систем з декількома GPU. Наведемо алгоритм факторизації матриці СЛАР A на комп'ютерах, де встановлено декілька GPU. Нехай кількість доступних GPU дорівнює $numGPU$. У цьому випадку i-ту плитку матриці буде обробляти GPU

з номером $i \bmod numGPU$ (нумерація GPU з 0). При цьому, нові версії CUDA дають можливість використовувати один CPU для роботи зі всіма доступними GPU. Оскільки основна частина обчислень виконується на GPU, то будемо використовувати один CPU потік.

Як і в попередньому випадку розіб'ємо реалізацію алгоритму на два етапи. На першому етапі виконуються такі операції:

- ініціалізація cuBLAS для кожного GPU;
- створення екземплярів cudaStream_t для кожного GPU, та їх запис у масиви: cudaCopyStreamArr[], cudaCalculateStreamArr[];
- встановлення основних потоків для доступних GPU;
- виділення пам'яті для $\frac{n}{w * numGPU} + 1$ плиток у кожному GPU;
- копіювання перших m / w плиток у пам'ять відповідних GPU.

На другому етапі, для кожного $i = 1, ..., n / w$ виконуємо наступні кроки алгоритму:

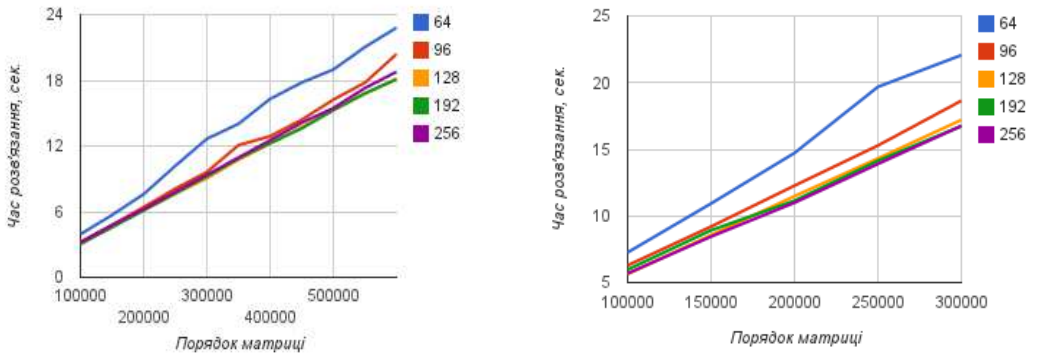
- виклик функції cudaSetDevice($i \bmod numGPU$). Таким чином поточним GPU стає той, в якому зберігається i-та плитка;
- перетворення на GPU нижньої частини плитки $Pu_{i,1}$ за формулою $Pu_{i,1} \leftarrow Pu_{i,1} * (Pb_i)^{-1}$. Використовується функція cublasDtrsm;
- асинхронне копіювання в потоці cudaCopyStreamArr[$i \bmod numGPU$] i-ї плитки в пам'ять CPU;
- встановлення поточним того GPU, в якому має зберігатися плитка, яка потрібна на наступному кроці. Копіювання цієї плитки в пам'ять відповідного GPU, використовуючи cudaCopyStreamArr;
- копіювання i-ї плитки в пам'ять всіх інших GPU;
- виконати цикл по j від 1 доки виконується нерівність $j * w < len[i]$; на кожній ітерації виконати перетворення (5), використовуючи функцію cublasDgemm. Перед викликом cublasDgemm потрібно встановити GPU з номером $(i + j) \bmod numGPU$ як поточний;
- синхронізація потоків cudaCopyStreamArr та збереження i-ї плитки на CPU;
- синхронізація основних потоків cuBLAS – cudaCalculateStreamArr.

Після виконання всіх кроків алгоритму отримаємо нижню трикутну матрицю, для якої справджується формула (2).

2.4. Розв'язання трикутних систем. Кількість операцій при розв'язанні трикутних систем (3, 4) значно менше, чим при трикутному розвиненні матриці системи. Тому ефективний алгоритм їх розв'язання може бути реалізований на CPU. Для розв'язання цих систем доцільно використовувати бібліотеку Intel MKL [4], в якій ефективно реалізовані функції для розв'язання таких систем.

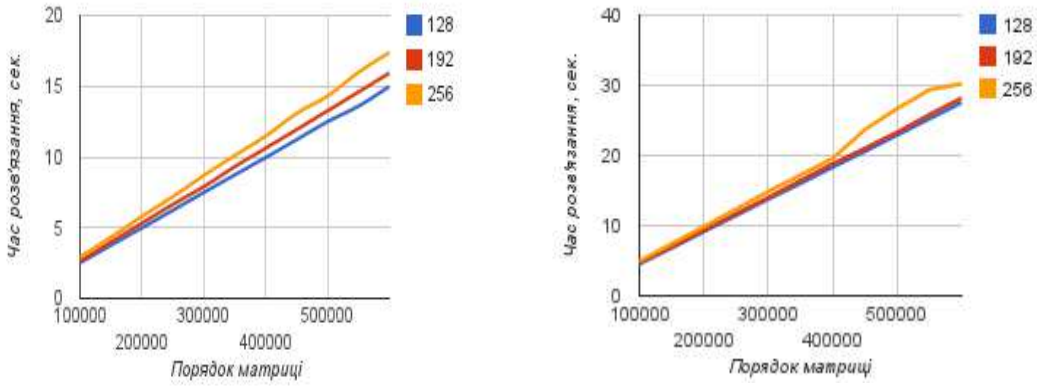
2.5. Експерементальне дослідження ефективності гібридного алгоритму. На основі запропонованого паралельного алгоритму розроблена програма для комп'ютера з гібридною (MIMD, SIMD) архітектурою. Апробація програми проведена на паралельному комп'ютері з графічними прискорювачами Інпарком [5].

На рис. 2, а та б показано залежність часу роботи програми розв'язання СЛАР від ширини плитки для алгоритму з одним GPU. На рис. 3, а та б показано залежність часу роботи програми розв'язання СЛАР від ширини плитки для алгоритму з декількома GPU. З графіків видно, що оптимальний розмір напівширин стрічки відрізняється для алгоритмів з одним та декількома GPU.



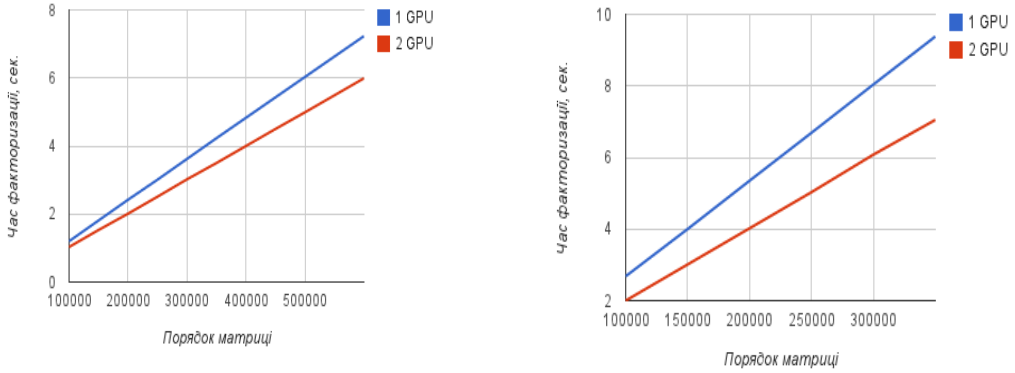
а б

РИС. 2. Залежність часу розв'язання СЛАР від ширини плитки з різною напівшириною стрічки (1000, 2000) для алгоритму з одним GPU



а б

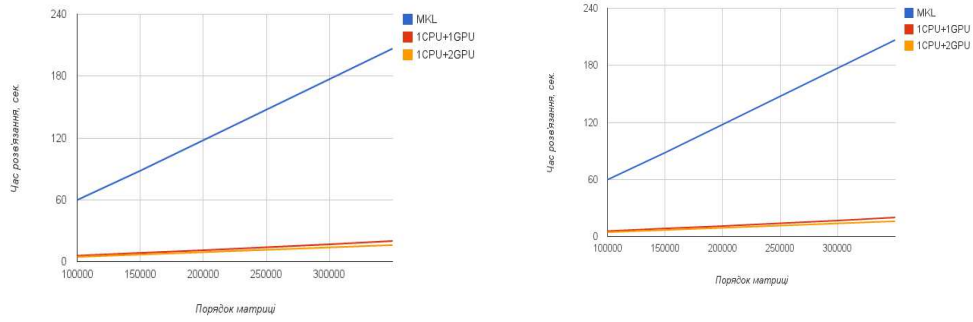
РИС. 3. Залежність часу розв'язання СЛАР від ширини плитки з різною напівшириною стрічки (1000, 2000) для алгоритму з двома GPU



а б

РИС. 4. Порівняння часу факторизації матриці СЛАР для різної напівширини стрічки (1000, 2000)

На рис. 4 показано порівняння часу факторизації матриці СЛАР алгоритмами з одним та декількома GPU. На рис. 5 – порівняння часу розв'язання СЛАР використовуючи Intel MKL [4] та описані алгоритми з одним та декількома GPU.



а б

РИС. 5. Порівняння часу розв'язання СЛАР з використанням MKL та алгоритмів для одного та декількох GPU

Висновки. Для СЛАР із стрічковими симетричними додатно визначеними матрицями запропоновано алгоритм розв'язання, який забезпечує високу ефективність розпаралелювання на GPU, враховує структуру стрічкової матриці, оптимізує використання пам'яті GPU. Також проведено дослідження вибору оптимального розміру ширини плиток, на які розбивається вихідна матриця.

Подальші дослідження доцільно направити на розробку алгоритмів з використанням декількох CPU і декількох GPU.

А.Н. Химич, А.Ю. Баранов

ГИБРИДНЫЙ АЛГОРИТМ РЕШЕНИЯ ЛИНЕЙНЫХ СИСТЕМ
С ЛЕНТОЧНЫМИ МАТРИЦАМИ ПРЯМЫМ МЕТОДОМ

Рассматривается новый гибридный алгоритм решения систем линейных алгебраических уравнений с ленточными симметричными положительно определенными матрицами на компьютерах с графическими ускорителями. Представлены результаты апробации алгоритма на многоядерном компьютере с графическими ускорителями Инпарком.

O.M. Khimich, A.Y. Baranov

HYBRID ALGORITHM FOR SOLVING LINEAR SYSTEMS WITH TAPE
MATRIX DIRECT METHODS

A new hybrid algorithm for solving systems of linear algebraic equations with band symmetric positive definite matrix on computers with GPU is considered. The results of testing of the algorithm on multicore computer Inparcom are presented.

1. *Химич А.Н., Попов А.В., Полянко В.В.* Алгоритмы параллельных вычислений для задач линейной алгебры с матрицами нерегулярной структуры // Кибернетика и системный анализ. – 2011. – 47, № 6. – С. 159–174.
2. *Уилкинсон Дж.Х., Райнш К.* Справочник алгоритмов на языке Алгол. Линейная алгебра. – М.: Машиностроение, 1976. – 389 с.
3. *Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra.* A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures.Parallel Computing. –2009. – Vol. 35, N 1. – P. 38–53.
4. http://software.intel.com/en-us/intel-mkl
5. *Химич А.Н., Молчанов И.Н., Мова В.И. и др.* Численное программное обеспечение MIMD-компьютера Инпарком. – Киев: Наукова думка, 2007. – 222 с.

Одержано 15.10.2013

Про авторів:

Хіміч Олександр Миколайович,
доктор фізико-математичних наук,
завідуючий відділом Інституту кібернетики імені В.М. Глушкова НАН України,

Баранов Андрій Юрійович,
аспірант Інституту кібернетики імені В.М. Глушкова НАН України.