Кібернетика та комп'ютерні технології

УДК 519.6 DOI:10.34229/2707-451X.25.4.6

О.М. ХІМІЧ, В.А. СИДОРУК, А.В. ПАВЛЮК

MULTI-GPU ДВОВИМІРНИЙ БЛОЧНО-ЦИКЛІЧНИЙ АЛГОРИТМ LU-ФАКТОРИЗАЦІЇ ЩІЛЬНИХ МАТРИЦЬ

Представлено паралельну реалізацію LU-факторизації великих щільних матриць, розроблену для високопродуктивних обчислень у середовищі з кількома графічними процесорами. Алгоритм ґрунтується на двовимірному блочно-циклічному розподілі даних, що забезпечує рівномірне завантаження GPU, зменшення комунікаційних витрат та підвищення масштабованості. Реалізація використовує сучасні засоби GPU-прискорення (NVLink, cuBLAS, cuSolver, NCCL), а також асинхронні потоки та перекриття комунікацій з обчисленнями, що дозволяє підтримувати високий ступінь паралелізму. Експериментальні результати, отримані на системі з вісьмома NVIDIA RTX 2080 Ti, демонструють ефективність до 95 % для матриць розміром до N = 20 000. Розроблений підхід є придатним для широкого спектра наукових і інженерних застосувань, включаючи моделювання фізичних процесів, аналіз конструкцій та задачі машинного навчання.

Ключові слова: *LU факторизація, мульти-GPU система, блочно-циклічний розподіл даних, паралельні обчислення, CUDA, cuBLAS, cuSolver, NCCL.*

Вступ. Розв'язання великих систем лінійних алгебраїчних рівнянь (СЛАР) – одна з базових задач прикладної математики та обчислювальної лінійної алгебри, що широко використовується в математичному моделюванні, фізичних симуляціях, машинному навчанні та фінансовому аналізі. Із зростанням розмірності задачі зростає потреба у високопродуктивних методах розв'язання, здатних ефективно масштабуватись на сучасних паралельних обчислювальних системах.

У наш час значного поширення набули гібридні високопродуктивні обчислювальні архітектури, які поєднують багатоядерні процесори (CPU) з масивно паралельними графічними прискорювачами (GPU). Архітектура мульти-GPU систем (або мультиграфічних обчислювальних систем) – це підхід до побудови комп'ютерної системи, що дозволяє використовувати декілька графічних процесорів для паралельних обчислень без залучення CPU.

Один із ключових викликів під час розв'язання СЛАР на таких системах – це ефективна організація розподілу даних, балансування навантаження та мінімізація витрат на комунікацію. У цій статті представлено паралельну реалізацію LU-факторизації на основі двовимірного блочно-циклічного розподілу матриці, орієнтовану на мульти-GPU системи. Особливу увагу приділено аналізу стратегії розподілу блоків, алгоритмічним аспектам прямого ходу, а також теоретичним характеристикам прискорення та масштабованості.

Дослідження паралельної LU-факторизації значно еволюціонували з появою гетерогенних обчислювальних платформ. Класичні алгоритми для комп'ютерів MIMD архітектури, такі як ті, що реалізовані в ScaLAPACK [1] та [2, 3], заклали основи блочно-циклічного розподілу даних для забезпечення балансування навантаження між процесорами.

З появою можливості проведення обчислень на графічних процесорах прискорилось створення гібридних алгоритмів, що дозволяють поєднувати обчислення на CPU та GPU [4–8]. Це дозволило отримати значні прирости у продуктивності обчислень як для щільних, так і розріджених матриць.

Варто зазначити роботи пов’язані з розвитком таких бібліотек як MAGMA [9] і cuSolver та cuBlas [10]. Останні досягнення розширюють класичні паралельні підходи на асинхронні моделі на основі завдань та техніки передбачення, покращуючи паралельність конвеєра та обчислювальне перекриття між фазами комунікації та обчислення.

В даній роботі запропоновано двовимірний блочно-циклічний мульти-GPU алгоритм LU факторизації, що поєднує асинхронні потоки CUDA та оптимізовані механізмами комунікації. Вона містить комплексний аналіз алгоритмічної масштабованості, ефективності та особливостей реалізації на сучасних мульти-GPU системах.

Мульти-GPU системи. Ключові переваги впровадження мульти-GPU систем охоплюють прискорення обробки, покращену масштабованість, підвищену надійність, оптимізацію витрат, об'єднання пам'яті та спеціалізацію робочого навантаження. Задачі, що вимагають інтенсивних обчислень, зокрема при реалізації методів машинного навчання, генеративного штучного інтелекту, великих мовних моделей. Вирішення цих задач, у будь-якому разі, потребує інтенсивного розв’язання задач лінійної алгебри.

Мульти-GPU системи можуть бути реалізовані у декількох конфігураціях, кожна з яких має свої особливості та застосування:

1. Мульти-GPU в одній системі (Single-system multi-GPU): це найпоширеніша конфігурація, де декілька GPU встановлені в одній робочій станції та з'єднані через материнську плату.

2. Пряме з'єднання GPU-GPU: у цьому випадку GPU спілкуються безпосередньо один з одним через високошвидкісні інтерконекти, такі як NVIDIA NVLink або AMD Infinity Fabric. Ця архітектура мінімізує затримку та максимізує пропускну здатність, що є критично важливим для інтенсивних обчислень.

3. Мережеві GPU-кластери (Network-based GPU clusters): ця конфігурація включає декілька комп'ютерів, кожен з яких оснащений одним або декількома GPU, з'єднаних через високошвидкісні мережі. Такі кластери часто використовуються у центрах обробки даних та хмарних середовищах для великомасштабних розподілених обчислень.

Різні типи мульти-GPU конфігурацій диктують вибір конкретних алгоритмів та програмних моделей.

Блочно-циклічні обчислення. Одна із проблем для створення ефективних паралельних обчислень – збалансованість обчислень. Наприклад, застосування послідовної схеми розподілу даних для паралельного розв’язування систем лінійних алгебраїчних рівнянь $Ax = b$ призведе до нерівномірного обчислювального навантаження процесорів: у міру виключення (на прямому ході) невідомих у методах триангуляції для все більшої частини процесорів обчислення будуть завершені й процесори будуть простоювати (так званий ефект Гайдна). Вирішення проблеми балансування обчислень полягає у використанні циклічної схеми для розподілу даних між процесорами та їх обробки. Одновимірна блочно-циклічна схема вперше була запропонована в [11]. Надалі для балансування процесорів, найбільше застосування отримав двовимірний блочно-циклічний алгоритм, обробки даних, який забезпечує кешизацію обчислень. Популярності цієї схеми сприяло використання рекурентного представлення блочного алгоритму для методів на основі триангуляції матриць.

Нехай A – квадратна матриця порядку n. Розіб’ємо її на квадратні блоки розміру $s \times s$. Не втрачаючи загальності міркувань, вважатимемо, що n/s – ціле число.

$$A = \left(A_{ij}\right), i, j = \overline{1,2,3,\ldots l}, l = n/s. \tag{1}$$

На k-ому кроці алгоритму ($k = 1,2,\ldots$) підматрицю $A^{(k)}$ порядку $r = n - \left(k-1\right)s$, яка містить останні r рядків і r стовпчиків у правому нижньому куті матриці $A^{(k)}$, представимо у вигляді:

$$A^{(k)} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = P \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = P \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{pmatrix}, \quad (2)$$

де A_{11} – блок, який має розмір $s \times s$, A_{12} – блок, що має розмір $s \times (r-s)$, A_{21} – блок, розміром $(r-s) \times s$, прямокутна підматриця, що містить блоки, розташовані нижче провідного, а A_{22} – блок, який має розмір $(r-s) \times (r-s)$. P – матриця перестановок у випадку вибору головного елементу (нехай $P = E$).

Таким чином рекурентний алгоритм блочної факторизації на k-ому блочному кроці ($k = 1, 2, \ldots, r-1$) матиме наступний вигляд:

$$A_{11} = L_{11}U_{11}, \quad (3)$$

$$U_{12} = (L_{11})^{-1} A_{12}, \quad (4)$$

$$L_{21} = A_{21}(U_{11})^{-1}, \quad (5)$$

$$\tilde{A}_{22} = A_{22} - L_{21}U_{12}. \quad (6)$$

Розподіл даних. У реалізації паралельної LU-факторизації ключову роль відіграє стратегія розподілу вхідної матриці A розміром $n \times n$ між обчислювальними ресурсами. Для забезпечення балансування навантаження та ефективної організації комунікацій використовується двовимірна блочно-циклічна схема розподілу даних.

$GPU(i, j)$ це GPU, розташований в i-му рядку та j-му стовпчику двовимірної $p \times q$ решітки, де $i \in [0, p-1]$, $j \in [0, q-1]$. Суть блочно-циклічного розподілу полягає у тому, що блоки призначаються GPU згідно з їхніми координатами за модулем розміру мережі:

$$GPU(i, j) \leftarrow A_{kl} \text{ якщо } i = k \bmod p, \; j = l \bmod q.$$

Такий підхід дозволяє: забезпечити рівномірний розподіл обчислювального навантаження між усіма процесорами у ході факторизації матриці, незалежно від розмірності задачі; зменшити пікове навантаження на окремі GPU; мінімізувати глобальні комунікації та пов'язані з ними затримки; уникнути ситуацій, коли один GPU виконує більшість обчислень на останніх етапах алгоритму.

Паралельна форма алгоритму

Означення. GPU, в якому знаходиться блок A_{11} у контексті (2) (на кожному кроці він може бути різним) назвемо головним GPU.

Стовпчик GPU, в якому знаходиться головний GPU назвемо головним стовпчиком GPU.

Рядок GPU, в якому знаходиться головний процесор назвемо головним рядком GPU.

Тоді паралельна форма алгоритму матиме наступний вигляд:

для $k = 1, 2, \ldots, r-1$ виконуємо:

- в головному GPU факторизуємо A_{11}

$$A_{11} = L_{11}U_{11};$$

- L_{11} розсилаємо вправо і вліво по головному рядку GPU;
- U_{11} розсилаємо вверх і вниз по головному стовпчику GPU;
- одночасно у всіх GPU головного рядка GPU виконуємо

$$U_{12} = (L_{11})^{-1} A_{12};$$

- одночасно у всіх GPU головного стовпчика GPU виконуємо

$$L_{21} = A_{21}\left(U_{11}\right)^{-1};$$

- всі GPU головного рядка передають вверх і вниз свої частини (панелі) U_{12};
- всі GPU головного стовпчика передають вправо і вліво свої частини (панелі) L_{21};
- одночасно у всіх $p \times q$ GPU виконуємо

$$\tilde{A}_{22} = A_{22} - L_{21}U_{12}.$$

Матриця $\tilde{A}_{22}$ згідно (2) буде мати на одиницю менший блочний порядок.

Ефективність. Для оцінки ефективності паралельного алгоритму *LU*-факторизації з двовимірним блочно-циклічним розподілом доцільно розглянути теоретичну модель часу виконання. Нехай t_g – час виконання однієї арифметичної операції, t_o – час передачі одного слова між GPU (обмін), t_c – час однієї глобальної синхронізації. Розмірність матриці – $n \times n$, сітка GPU – $p \times q$, розмір блоку – $s \times s$. Прискорення і ефективність обчислюються за наступними співвідношеннями:

$$S_p = \frac{T_1}{T_p}, \quad E_p = \frac{S_p}{p}, \quad T_p = T + T_o + T_c,$$

де T_1 – час виконання обчислень на 1 GPU, T_p – час виконання обчислень на p GPU, $T = N_p t_g$ – час паралельних обчислень, $T_o = Mt_o$ – час обміну даними, $T_c = Qt_c$ – час налаштування зв'язку.

Для нашого алгоритму для $p = q$ маємо:

$$T_1 = \frac{2n^3 t_g}{3},$$

$$N_p \approx \frac{2s}{p^2}\sum_{k=1}^{l}\left(n - ks\right)^2 \approx \frac{2}{3p^2}n^3,$$

$$M \approx s\sum_{k=1}^{l}(n - sk) \approx \frac{n^2}{2},$$

$$Q \approx 2pl = \frac{2pn}{s},$$

$$T_p \approx \frac{2}{3p^2}n^3 t_g + \frac{n^2}{2}t_o + \frac{2pn}{s}t_c,$$

$$S_p \approx \frac{T_1}{T_p}\frac{p^2}{\left(1 + \frac{3}{4}\frac{p^2}{n}\tau_o + \frac{3}{s}\frac{p^3}{n^2}\tau_c\right)},$$

$$\tau_o = \frac{t_o}{t_g}, \quad \tau_c = \frac{t_c}{t_g}.$$

Ці формули дозволяють проаналізувати вплив порядку системи, розміру блоку та швидкодії обміну і синхронізації на масштабованість алгоритму.

Особливості програмної реалізації. Програмна реалізація алгоритму LU-факторизації з блочно-циклічним розподілом для мульти-GPU середовища передбачає врахування як архітектурних особливостей обчислювальної платформи, так і оптимізації ключових обчислювальних етапів. У цьому розділі наведено основні принципи реалізації та методи, застосовані для забезпечення високої продуктивності та масштабованості.

На кожному кроці алгоритму виконується LU-факторизація блоку A_{11}, що зберігається локально на одному з GPU. Для цього використовується функціонал бібліотеки cuSolver (cusolverDnDgetrf), який забезпечує обчислення з частковим pivoting. Отримані вектори перестановок синхронізуються з іншими GPU для коректного відображення структури факторизації.

Після завершення локальної факторизації панель (тобто блоки L_{21} та U_{12}) транслюється іншим GPU за допомогою колективних комунікацій. Для цього використовується бібліотека NCCL [12], що забезпечує ефективну реалізацію операцій broadcast з урахуванням топології NVLink. У разі наявності peer-to-peer доступу між GPU використовується механізм cudaMemcpyPeerAsync.

Для обчислення блоків нижче та правіше від поточної панелі виконується розв'язування трикутних систем (TRSM). Для цього залучається бібліотека cuBLAS (cublasDtrsm), яка виконує операції у спеціалізованих CUDA-потоках. Розв'язування виконується паралельно для всіх залежних блоків, локальних для відповідного GPU.

Оновлення елементів матриці A_{22}, реалізується через виклики cublasDgemm. Обчислення виконуються у відповідних асинхронних потоках, що дозволяє перекривати передачу панелі, TRSM та обчислення GEMM.

Для кожного GPU використовується кілька CUDA-потоків: окремо для факторизації панелі, TRSM та GEMM. Для синхронізації етапів використовуються виклики cudaEvent_t, які дозволяють реалізувати ефективне перекриття обчислень та комунікацій. Завдяки цьому досягається зменшення часу простоїв та підвищення ефективності використання ресурсів.

Усі робочі буфери (панелі, блоки L/U, проміжні результати GEMM) попередньо виділяються в глобальній пам'яті GPU. Завдяки повторному використанню буферів та уникненню частих cudaMalloc/cudaFree викликів, зменшуються накладні витрати на керування пам'яттю. У разі підтримки CUDA Memory Pool або cudaMallocAsync – використовується відповідна інфраструктура.

Продуктивність оцінюється через заміри часу обчислень (cudaEventElapsedTime) та підрахунок кількості виконаних FLOPs на кожному етапі. Теоретична складність LU-факторизації становить $\frac{2}{3}n^3$, а для кожного етапу додатково враховуються FLOPs для TRSM та GEMM. Реальна ефективність обчислюється у відсотках від пікового значення GPU.

Результати чисельних експериментів.

Апробація розробленого паралельного алгоритму, що базується на ідеології двовимірного блочно-циклічного розподілу даних, проводилася на задачі обробки квадратної матриці порядку $N = 20000$. Експерименти виконувалися на одновузловому кластері СКІТ [13] g4301, обладнаному 8 GPU Nvidia GeForce RTX 2080 Ti. Основна мета апробації – визначення ефективності масштабування та встановлення оптимальних параметрів розподілу.

Залежність прискорення від кількості графічних прискорювачів показано на рис. 1. Аналіз отриманих результатів свідчить, що застосування двовимірного блочно-циклічного розподілу даних забезпечує високий рівень паралелізму обчислень і дозволяє ефективно використовувати апаратні ресурси мульти-GPU системи. Спостерігається майже лінійне масштабування, що вказує на домінування локальних обчислень над комунікаційними витратами та підтверджує достатню збалансованість обчислювального навантаження між пристроями.

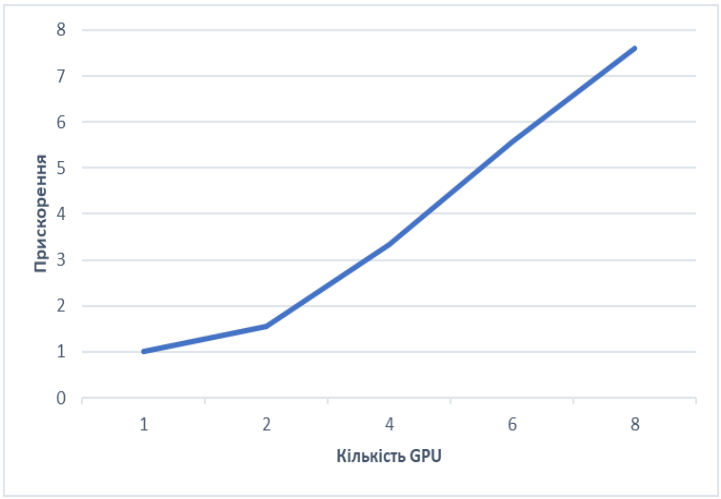

РИС. 1. Залежність прискорення від кількості GPU

На рис. 2 показано залежність ефективності паралельного виконання від кількості задіяних графічних прискорювачів. Отримані результати демонструють стабільне зростання ефективності при масштабуванні від одного до восьми GPU: ефективність підвищується з приблизно 74 % для одного прискорювача до понад 90 % при використанні шести та восьми GPU. Така поведінка свідчить про високу збалансованість запропонованої 2D блочно-циклічної схеми розподілу даних і про здатність алгоритму ефективно залучати додаткові апаратні ресурси без істотного збільшення комунікаційних витрат. Зростання ефективності зі збільшенням числа прискорювачів також підтверджує домінування локальних обчислень у структурі алгоритму та оптимальний характер організації між-GPU обмінів. Таким чином, результати експериментів засвідчують, що запропонований підхід забезпечує продуктивне масштабування навіть за умов значного збільшення кількості обчислювальних пристроїв.

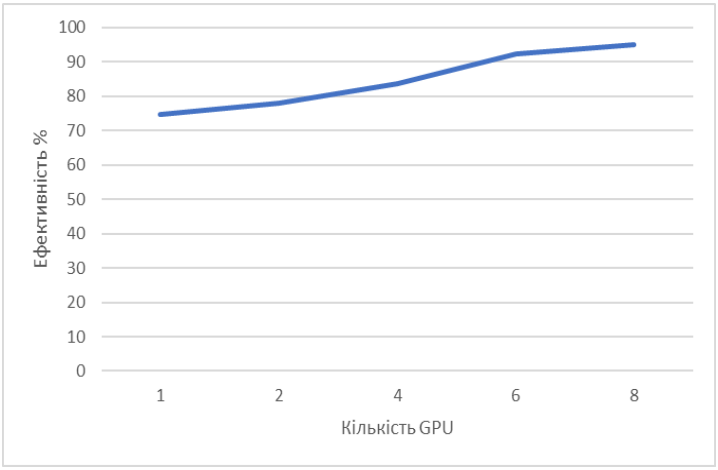

РИС. 2. Залежність ефективності від кількості GPU

Для підтвердження оптимальності обраної ідеології розподілу було проведено аналіз впливу розміру блоку s на загальний час виконання алгоритму при фіксованій кількості GPU $p=1$ (рис. 3).

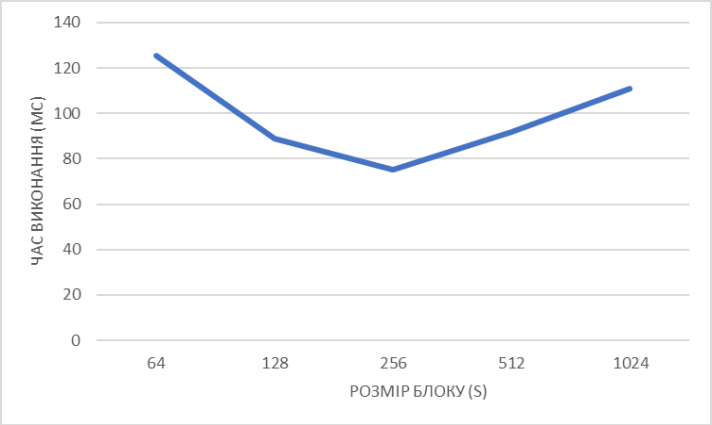

РИС. 3. Залежність часу виконання від розміру блоку *s*

Проведені чисельні експерименти повністю підтвердили ефективність розробленої ідеології двовимірного блочно-циклічного розподілу даних та її програмної реалізації. Було досягнуто виняткової ефективності (95.0 %) і визначено оптимальний розмір блоку $s = 256$, що дозволяє рекомендувати алгоритм для високопродуктивних обчислень на GPU-акселерованих системах.

Ефективність паралельного алгоритму LU-факторизації. У мульти-GPU середовищі ефективність залежить не лише від формальної структури алгоритму, а й від багатьох практичних факторів, пов'язаних з архітектурою апаратного забезпечення, організацією пам'яті, характеристиками комунікацій та ефективністю реалізації.

Обчислювальна складність LU-факторизації пропорційна $O\left(n^3\right)$, тоді як обсяг переданих даних у процесі комунікацій – лише $O\left(n^2\right)$. Таким чином, зі збільшенням розмірності задачі відносна частка комунікаційних витрат зменшується. Для малих матриць (порядку кількох тисяч) навпаки – алгоритм є **communication-bound**, тобто домінують витрати на обмін даними, синхронізацію та очікування, що обмежує реальне прискорення навіть при використанні великої кількості обчислювальних пристроїв.

Для практичних обчислень це означає, що ефективне масштабування LU-факторизації спостерігається лише при досить великих розмірностях ($n > 10^4$). Водночас збільшення n вимагає значного обсягу пам'яті та пропускної здатності міжмодульних каналів, що накладає апаратні обмеження.

Вибір розміру блоку $s \times s$ суттєво впливає на баланс між обчисленнями, комунікаціями та ефективністю кешування. Оптимальний розмір блоку повинен забезпечувати:

- максимальне завантаження обчислювальних ядер GPU, включно з тензорними блоками;
- мінімізацію промахів кешу при доступі до елементів матриці;
- зменшення частоти комунікацій між обчислювальними вузлами;
- збалансоване навантаження у межах 2D блочно-циклічного розподілу.

Надмірно великі блоки знижують гнучкість розподілу даних і призводять до дисбалансу навантаження (частина процесорів може простоювати). Навпаки, занадто дрібне блочне розбиття створює значні накладні витрати на синхронізацію і передачу малих порцій даних. На практиці оптимальне значення *s* вибирають емпірично або через автоматичний тюнінг, враховуючи тип GPU, обсяг локальної пам'яті та швидкодію NVLink. Для сучасних архітектур NVIDIA значення s зазвичай лежить у діапазоні 128–512.

Мережеві інтерконекти відіграють ключову роль у забезпеченні високої продуктивності при багатопроцесорній реалізації. NVLink забезпечує пропускну здатність до 600–900 ГБ/с між GPU, що в десятки разів перевищує можливості PCIe 4.0/5.0, а затримки – на порядок менші. Це дозволяє ефективно реалізовувати колективні операції типу broadcast, reduce, all-to-all, які є критичними для LU-факторизації з розподіленими блоками.

Якщо NVLink відсутній і обмін даними здійснюється через PCIe, продуктивність може знижуватись у 2–4 рази через необхідність копіювання через host-пам'ять. У масштабованих системах з декількох вузлів використовується InfiniBand з технологією GPUDirect RDMA, яка дає змогу прямого обміну між пам'ятями GPU різних вузлів без участі CPU, мінімізуючи затримки і покращуючи масштабування.

Для досягнення максимальної ефективності при Програмній реалізації доцільно враховувати такі фактори:

- cuBLAS/cuSolver використовуються для локальних обчислень (*TRSM, GEMM, GETRF*), забезпечуючи пікову продуктивність GPU;
- NCCL або MPI+CUDA-aware використовуються для колективних операцій обміну даними (*broadcast, reduce, gather*);
- асинхронні потоки (CUDA streams) дають змогу перекривати обчислення та передачу даних, зменшуючи простої;
- Look-ahead та double buffering дозволяють починати оновлення віддалених блоків ще до завершення поточної факторизації панелі, підвищуючи рівень конвеєризації.

Висновки. Представлено ефективну паралельну реалізацію LU-факторизації для великих щільних матриць на основі двовимірного блочно-циклічного розподілу даних у середовищі з виключно графічними процесорами (multi-GPU). Запропонований підхід спрямований на підвищення продуктивності обчислень під час розв'язання задач великої розмірності ($n > 10^4$) завдяки рівномірному розподілу навантаження між GPU, локалізації обчислень та мінімізації обсягів комунікацій.

Розроблена схема поєднує структурну регулярність алгоритму LU-факторизації з оптимальним використанням апаратних ресурсів сучасних GPU, зокрема, швидкісних інтерконектів NVLink і високопродуктивних бібліотек cuBLAS, cuSolver та NCCL. Це забезпечує ефективне масштабування на системах із кількома обчислювальними пристроями, дозволяючи досягати високого рівня паралелізму при збереженні чисельної стійкості.

Запропонована реалізація це поєднання класичної матричної LU-декомпозиції з сучасними принципами паралельної обробки даних у гетерогенних системах. Новизна роботи полягає у розробці двовимірного блочно-циклічного паралельного алгоритму з теоретичним обґрунтуванням прискорення та ефективності. Асинхронні потоки та перекриття комунікацій з обчисленнями у виключно у GPU середовищі, що забезпечує високу пропускну здатність і стале завантаження GPU протягом усіх етапів факторизації. Практична цінність результатів полягає у можливості застосування розробленого підходу для чисельного моделювання складних фізичних процесів, зокрема, у задачах механіки суцільних середовищ, обчислювальної гідродинаміки, аналізу стійкості конструкцій та великих систем диференціальних рівнянь, для задач машинного навчання та штучного інтелекту.

Подальший розвиток роботи передбачає.

1. Розширення алгоритму на розріджені матриці із використанням адаптивного розподілу ненульових блоків між GPU.
2. Використання змішаної розрядності обчислень (mixed precision) із автоматичним контролем похибки для підвищення енергоефективності та прискорення виконання без втрати точності.
3. Адаптацію методики до факторизацій інших типів (Cholesky, QR, LDL^t), що відкриває перспективи створення єдиної бібліотеки високопродуктивних блочних факторизацій для гетерогенних систем.

Отримані результати підтверджують, що двовимірна блочно-циклічна схема LU-факторизації це ефективний і масштабований інструмент для реалізації паралельних алгоритмів у середовищах з багатьма GPU. Використання сучасних бібліотек, асинхронних потоків і високошвидкісних комунікацій забезпечує наближення до теоретичних меж продуктивності, що робить запропонований підхід придатним для розв'язання задач великомасштабного чисельного моделювання на суперкомп'ютерних системах нового покоління.

Внесок авторів: Хіміч О.М. – розробка теоретичної основи (ідеології) двовимірного блочно-циклічного розподілу даних, а також визначення паралельної форми алгоритму та критеріїв ефективності; Сидорук В.А. – огляд літератури, розробка паралельного алгоритму, програмна реалізація і аналіз шляхів оптимізації; Павлюк А.В. – апробація алгоритму: програмна реалізація, проведення чисельних експериментів.

Список літератури

1. Dongarra J., Whaley R.C., Petitet A. A ScaLAPACK User's Guide: Solving Linear Algebra Problems on Distributed Memory Computers. 1998. SIAM, Philadelphia. https://www.netlib.org/scalapack/slug/scalapack_slug.html
2. Химич А.Н., Молчанов И.Н, Мова В.И. и др. Численное программное обеспечение MIMD-компьютера Инпарком. Киев: Наук. думка, 2007. 222 с.
3. Химич А.Н., Молчанов И.Н., Попов А.В., Чистякова Т.В., Яковлев М.Ф. Параллельные алгоритмы решения задач вычислительной математики. Киев: Наук. думка, 2008. 247 с.
4. Volkov V., Demmel J. LU, QR and Cholesky Factorizations Using Vector Capabilities of GPUs. University of California, Berkeley, 2008. https://www.netlib.org/lapack/lawnspdf/lawn202.pdf
5. Dongarra J., Tomov S., Haidar A. MAGMA: Matrix Algebra on GPU and Multicore Architectures. Innovative Computing Laboratory, University of Tennessee. 2012.
6. Хіміч О.М., Полянко В.В., Чистякова Т.В. Паралельні алгоритми розв'язування лінійних систем на гібридних комп'ютерах. *Cybernetics and Computer Technologies*. 2020. № 2. С. 3–66. https://doi.org/10.34229/2707-451X.20.2.6
7. Khimich O.M., Popov O.V., Chistyakov, O.V. et al. A Parallel Algorithm for Solving a Partial Eigenvalue Problem for Block-Diagonal Bordered Matrices. *Cybern Syst Anal*. 2020. Vol. 56. P. 913–923. https://doi.org/10.1007/s10559-020-00311-z
8. Baranov A.Y., Popov A.V., Slobodyan Y.E., Khimich A.N. Mathematical modeling of building con-structions using hybrid computing systems. *Journal of Automation and Information Sciences*. 2017. Vol. 49, Iss. 7. P. 18–32. https://doi.org/10.1615/JAutomatInfScien.v49.i7.20
9. Haidar A., Tomov S., Dongarra J. Towards Batched Dense Linear Algebra Routines on GPUs: Reduction and Factorization. Concurrency and Computation: Practice and Experience. 2019. **31** (6), e4965.
10. NVIDIA Corporation. CuSolver and cuBLAS Libraries Documentation. NVIDIA Developer Portal. 2023.
11. Михалевич В.С., Бик Н.А., Брусникин Б.Н., Химич А.Н. и др. Численные методы для многопроцессорного вычислительного комплекса ЕС. Под редакцией И.Н. Молчанова. М.: Издание ВВИА им. проф. Н.Е. Жуковского, 1986. 401 с.
12. NVIDIA Corporation. NCCL 2.15: Multi-GPU Collective Communication Library. NVIDIA Developer Documentation. 2022.
13. Обчислювальний комплекс СКІТ ІК НАН України. http://icybcluster.org.ua/index.php?lang_id=2&menu_id=5 (звернення: 20.10.2025)

Одержано 20.10.2025

Хіміч Олександр Миколайович,
доктор фізико-математичних наук, академік НАН України,
заступник директора Інституту кібернетики імені В.М. Глушкова НАН України,
https://orcid.org/0000-0002-8103-4223

Сидорук Володимир Антонович,
кандидат фізико-математичних наук,
старший науковий співробітник Інституту кібернетики імені В.М. Глушкова НАН України,
https://orcid.org/0000-0003-0210-6020
wolodymyr.sydoruk@gmail.com

Павлюк Антон Володимирович,
аспірант Інституту кібернетики імені В.М. Глушкова НАН України.
https://orcid.org/0009-0002-4166-2003
kvadrumpl@gmail.com

UDC 519.6

Oleksandr Khimich, Volodymyr Sydoruk *, Anton Pavliuk

Multi-Gpu Two-Dimensional Block-Cyclic Algorithm for Factorization of Dense Matrix

V.M. Glushkov Institute of Cybernetics of the NAS of Ukraine, Kyiv
** Correspondence: wolodymyr.sydoruk@gmail.com*

This article presents an efficient parallel algorithm for LU factorization of large dense matrices based on a two-dimensional block-cyclic data distribution designed for multi-GPU computing environments. The proposed methodology addresses key challenges of large-scale linear algebra computations, including load balancing, minimizing communication overhead, and maximizing computational locality. By distributing matrix blocks cyclically across a GPU process grid, the algorithm ensures uniform workload distribution even for very large problem sizes, thereby avoiding idle time and reducing synchronization delays.

The implementation leverages state-of-the-art GPU computing technologies, including CUDA streams, cuBLAS and cuSolver libraries for local numerical kernels, and NCCL for high-performance collective communications using NVLink interconnects. A look-ahead scheduling strategy and overlapping of communication with computation further increase the degree of parallelism, enabling sustained high utilization of GPU resources throughout the factorization pipeline.

A detailed theoretical performance model is developed to analyze speedup, scalability, communication costs, and the impact of block size on total execution time. Numerical experiments conducted on an 8-GPU node with NVIDIA RTX 2080 Ti GPUs demonstrate excellent strong scaling, achieving up to 95 % parallel efficiency for matrices of size up to $N = 20{,}000$. The results confirm that the proposed multi-GPU LU factorization approach closely approaches the theoretical performance limits and significantly outperforms traditional CPU-based and hybrid CPU–GPU schemes.

The method is highly suitable for large-scale scientific and engineering applications requiring fast and robust solution of linear systems, including computational fluid dynamics, structural mechanics, numerical simulation of physical processes, and machine learning workloads. Future research directions include extensions to sparse matrices with adaptive load balancing, mixed-precision acceleration with error correction, and generalization to other matrix factorizations such as Cholesky, QR, and LDL^T.

Keywords: LU factorization, multi-GPU systems, block-cyclic data distribution, parallel computing, CUDA, cuBLAS, cuSolver, NCCL, high-performance computing, scalability.