Новітній швидкий алгоритм знаходження мінімальних опуклих оболонок

Погорілий С.Д., Потебня А.В.

Київський національний університет імені Тараса Шевченка, просп. Глушкова, 4-г, Київ, Україна

sdp@univ.net.ua, poteba@yandex.ru

Анотація. *Запропоновано новітній алгоритм формування мінімальних опуклих оболонок графа з використанням графічних прискорювачів. Висока швидкодія та лінійна складність такого методу досягається за рахунок розподілення вершин графа на окремі блоки та здійснення їх фільтрації. Керування обчислювальним процесом відбувається за допомогою допоміжних матриць. Виконано низку експериментальних досліджень алгоритму та показано придатність його застосування при обробці оболонок для задач великої розмірності. Встановлено, що швидкість нового методу є в 10 – 20 разів вищою порівняно з використанням функцій професійного математичного пакету Wolfram Mathematica.*

Ключові слова

Мінімальна опукла оболонка, гібридні системи CPU+GPU, технологія GPGPU, високопродуктивні обчислення, CUDA, граф.

1 Вступ

Знаходження мінімальної опуклої оболонки (МОО, від англ. *convex hull*) множини вершин графа є фундаментальною проблемою у багатьох сферах сучасних наукових досліджень [1]. Розв'язання цієї задачі передбачає формування найменшого опуклого набору, який містить всі вузли, наявні у графі (рис. 1а). Відомо, що МОО є поширеним інструментом в системах автоматизованого проектування та пакетах комп'ютерної графіки. Наприклад, криві Без'є *(Bezier curve)*, які застосовуються в програмах *Adobe Photoshop*, *GIMP* та *CorelDraw* для моделювання гладких ліній, повністю лежать в опуклій оболонці своїх контрольних вузлів (рис. 1б). Ця властивість значно спрощує знаходження точок перетину кривих та дозволяє здійснювати їх трансформації (перенесення, масштабування, обертання та інші) за допомогою відповідних контрольних вузлів. Формування деяких шрифтів та анімаційних ефектів у пакеті *Adobe Flash* також відбувається за допомогою квадратичних кривих Без'є у складі сплайнів [2].

Слід зазначити про застосування опуклих оболонок у географічних інформаційних системах *(Geographical Information Systems)*, алгоритмах маршрутизації та при визначенні оптимальних шляхів обходу перешкод. У [3] з їх використанням запропоновані методи розв'язання складних задач оптимізації.

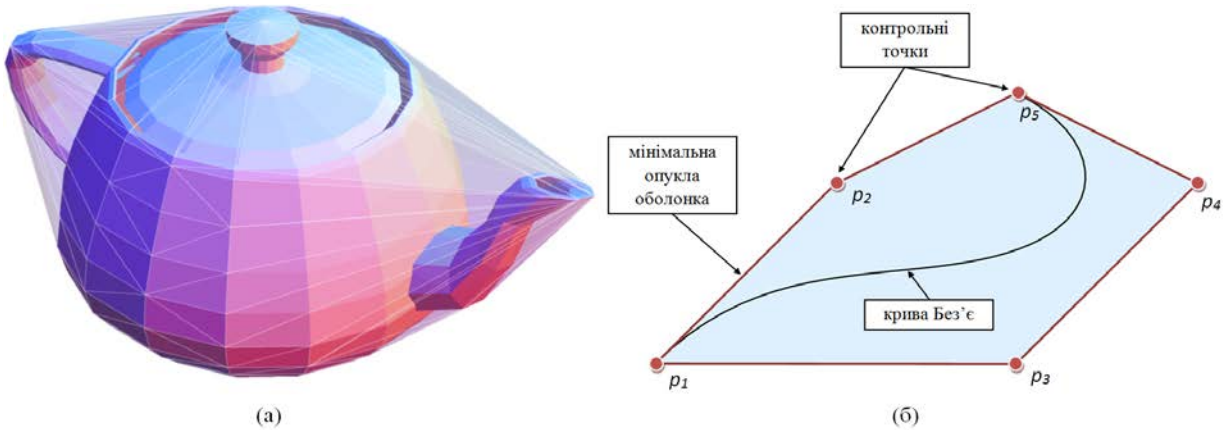

Рис. 1. Приклади мінімальних опуклих оболонок

Останні десятиріччя пов'язані з бурхливим зростанням обсягу даних в наукових дослідженнях, які обробляються інформаційними системами. За оцінками IBM, щодня в світі створюється близько 15 петабайт нової інформації [4]. Тому в сучасній науці виник окремий напрям *Big Data*, пов'язаний з дослідженням великих наборів даних [5]. Однак, більшість відомих алгоритмів формування МОО мають часову складність $O(n\log n)$, що обумовлює їх непридатність при формуванні розв'язків для графів великої розмірності. Тому виникає необхідність розробки ефективних алгоритмів зі складністю, близькою до лінійної $O(n)$.

Відомо, що Wolfram Mathematica є одним з найпотужніших математичних засобів для здійснення високопродуктивних обчислень. Функції цього пакету інкапсулюють цілу низку алгоритмів та, залежно від параметрів вхідної задачі, здійснюють вибір найпродуктивнішого з них. Тому, програму Wolfram Mathematica 9.0 застосовано для відстеження ефективності алгоритму, запропонованого в цій статті.

В останні роки надзвичайного поширення набули гібридні системи CPU+GPU (технологія GPGPU), що дозволяють забезпечити суттєве прискорення обчислень. На відміну від центрального процесора, який складається з декількох ядер, графічний процесор є багатоядерною структурою, кількість компонентів якої вимірюється сотнями. При цьому послідовні частини алгоритму виконуються на CPU, а паралельні – на GPU. Наприклад, графічні процесори NVIDIA останнього покоління «Fermi» містять 512 обчислювальних ядер, що дозволяє здійснювати впровадження новітніх алгоритмів з масовим паралелізмом [6]. Таким чином, застосування графічних прискорювачів NVIDIA забезпечує перетворення стандартних робочих станцій у потужні суперкомп'ютери з продуктивністю на рівні кластера.

Метою роботи є формування високошвидкісного алгоритму знаходження мінімальних опуклих оболонок з використанням графічних прискорювачів. Часова складність запропонованого методу є близькою до лінійної. На основі проведених досліджень встановлено оптимальні значення параметрів алгоритму, за яких використання ресурсів GPU є найбільш ефективним.

2 Алгоритми знаходження мінімальних опуклих оболонок

Попри інтенсивні дослідження, які тривали протягом останніх 40 років, проблема розробки ефективних алгоритмів формування МОО досі залишається відкритою. Основним досягненням є формування низки методів, які передбачають визначення екстремальних точок вихідного графа та побудову системи з'єднань між ними [7]. До них належать алгоритм Джарвіса *(Jarvis march)*, Грехема *(Graham Scan)*, швидкої оболонки *(QuickHull)*, «Розділяй і володарюй» *(«Divide and conquer»)* та багато інших. Особливості практичного використання таких алгоритмів містяться в таблиці 1.

Таблиця 1. Порівняння поширених алгоритмів формування МОО

Алгоритм	***Складність***	***Паралельні версії***	***Застосування для багатовимірних випадків***
Алгоритм Джарвіса	$O(nh)$, де h – кількість вузлів МОО	+	+
Алгоритм Грехема	$O(n\log n)$	-	-
Алгоритм швидкої оболонки	$O(n\log n)$, в найгіршому випадку – $O(n^2)$	+	+
Алгоритм «Розділяй та володарюй»	$O(n\log n)$	+	+

Для розпаралелювання найбільш придатним є алгоритм «Розділяй та володарюй», який передбачає випадкове розбиття вихідної множини вершин графа на підмножини, формування частинних розв'язків та їх з'єднання до загального [8]. Попри те, що етап сполучення оболонок має лінійну складність, він призводить до суттєвого уповільнення алгоритму та, як наслідок, непридатності його застосування при обробці оболонок для графів великої розмірності.

Найнижчу часову складність $O(n\log h)$ має алгоритм Чана, який є комбінацією більш повільних алгоритмів. Однак, він може працювати лише за відомого значення кількості вузлів, які містяться в оболонці. Тому, на сьогоднішній день його застосування на практиці є обмеженим [9].

У праці [8] наведені різноманітні засоби прискорення відомих алгоритмів формування МОО шляхом відсікання частини графа восьмикутником або прямокутником та здійснення в такий спосіб зменшення розмірності вихідної задачі. У роботах [10, 11] запропоновано чимало методів наближеного утворення опуклих оболонок, які мають лінійну складність. Такі алгоритми широко застосовуються для задач, в яких швидкодія є критичним параметром.

Наведені недоліки, властиві для відомих алгоритмів, демонструють необхідність впровадження новітніх високошвидкісних методів утворення опуклих оболонок для графів великої розмірності.

3 Новітній алгоритм формування опуклих оболонок

Нехай, задано неорієнтований граф G = (V, E). Новий алгоритм передбачає розбиття множини вершин вихідного графа на сукупність блоків $B = \langle B_1, B_2, ..., B_n \rangle$, $B_i \subseteq V$. Однак, на відміну від методу «Розділяй та володарюй» такий розподіл відбувається не випадковим чином, а з урахуванням просторового розміщення точок. Всі вузли графа мають бути розподілені за утвореними підмножинами, тобто $\bigcup_{i=1}^{n} B_i = V$. При цьому допускається наявність порожніх блоків, які не містять вершин $B_i = \varnothing$. Додатково забезпечується умова ортогональності розбиття, тобто одна вершина не може входити до складу різних блоків: $B_i \bigcap B_j = \varnothing, \forall i \neq j$. На рис. 2а наведено приклад розподілу з урахуванням наведених вимог.

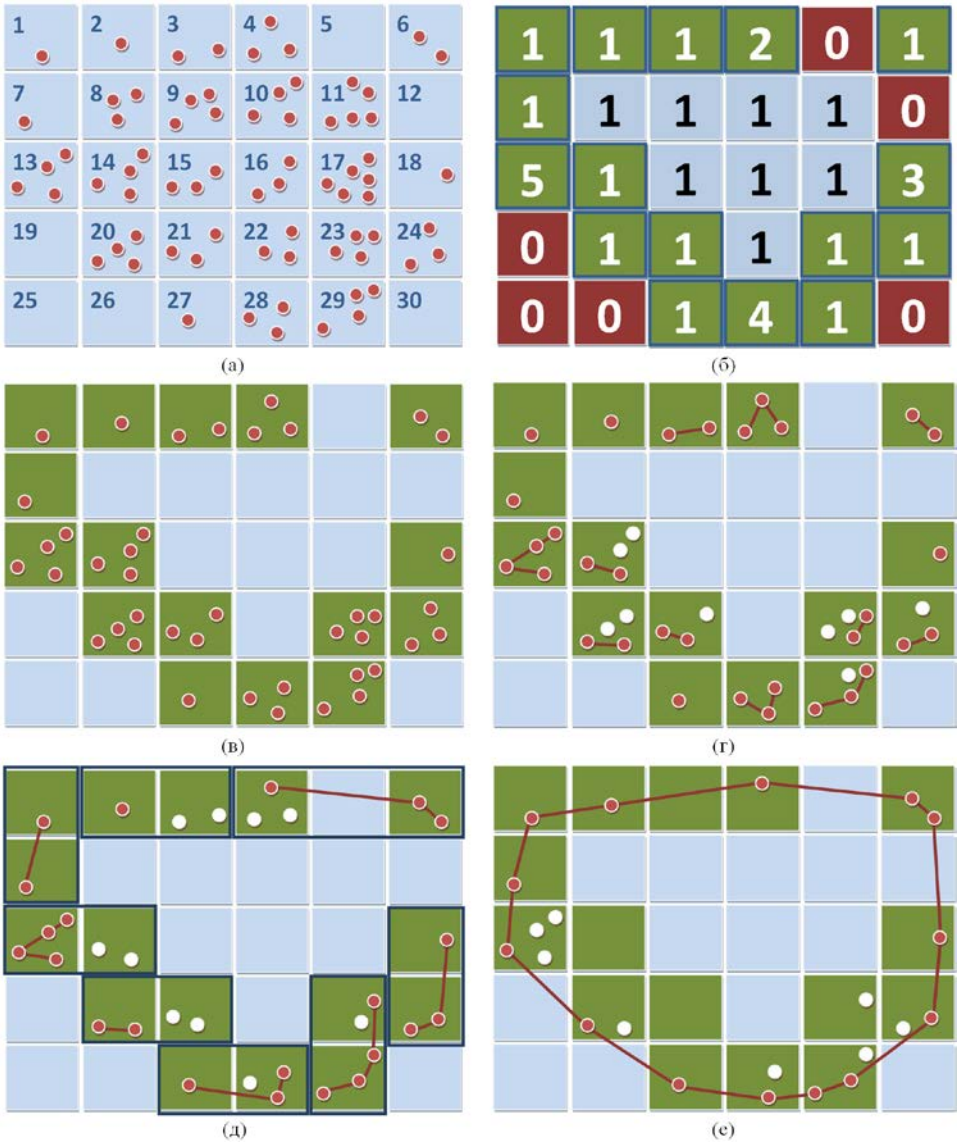

Рис. 2. Приклад роботи нового алгоритму

Наступний етап запропонованого алгоритму полягає у здійсненні формування допоміжної матриці на основі розподілення вузлів за блоками. Метою такої процедури є здійснення первинної фільтрації вершин графа, що забезпечує суттєве зменшення розмірності вихідної задачі. Крім того, за такими матрицями визначаються набори блоків для обчислення на наступних етапах алгоритму та послідовність їх сполучення до загального результату. Формування допоміжної матриці передбачає виконання наступних операцій:

1. Кожному блоку вихідного графа ставиться у взаємно однозначну відповідність одна комірка допоміжної матриці. Відповідно, розмірність такої матриці дорівнює $n \times m$, де n та m – кількість виділених блоків за відповідними напрямами.
2. Наступні операції забезпечують необхідне кодування комірок матриці. При цьому значення комірки $c_{i,j}$ дорівнює нулю, якщо відповідний блок $B_{i,j}$ вихідного графа не містить вершин. Важливим для роботи алгоритму є кодування блоків, які містять екстремальні вузли заданого набору. Відповідні комірки заповнюються цифрами від 2 до 5. Інші блоки, які є заповненими, але не містять екстремальних вершин кодуються одиницями в допоміжній матриці.
3. Надалі відбувається здійснення первинної фільтрації виділених блоків з використанням заповненої матриці. Порожні підмножини при цьому виключаються з розгляду. Блоки, які містять екстремальні вузли, визначають розподілення графа на частини, для яких застосовується процедура фільтрації. Розглянемо приклад відбору блоків для ділянки, обмеженої комірками 2 – 3. Якщо $c_{i,j} = 2$, то шляхом послідовного збільшення j відбувається пошук наступної ненульової комірки. В разі їх відсутності відбувається перегляд наступного $i+1$ рядка матриці. Відбір блоків закінчується, якщо значення чергової знайденої комірки $c_{i,j} = 3$. Дослідження інших частин вихідного графа відбувається за аналогічним принципом.

Для відібраних блоків здійснюється формування частинних розв'язків. Такі операції потребують утворення не повноцінних оболонок, а лише їх фрагментів, що забезпечує виконання вторинної фільтрації вершин графа. Останній крок алгоритму пов'язаний зі сполученням частинних розв'язків до загального результату. При цьому послідовне злиття локальних фрагментів здійснюється за принципом, подібним до алгоритму Джарвіса. Слід зазначити, що на цьому етапі механізм фільтрації спричинює значне зниження розмірності вихідної задачі. Тому при обробці оболонок для великих графів операції об'єднання складають близько 0,1% загального часу роботи алгоритму.

Розглянемо приклад виконання цього алгоритму. Нехай, множина вершин вихідного графа зазнала розподілення на 30 блоків (рис. 2а). Розрахована для такого випадку допоміжна матриця міститься на рис. 2б. Застосування первинної фільтрації забезпечило відбір 57% вузлів графа для дослідження на наступних етапах алгоритму (рис. 2в). Наступні операції потребують формування локальних оболонок (рис. 2г) та їх об'єднань, наведених на рис. 2д. Після виконання попарних сполучень така операція застосовується повторно, до одержання глобальної опуклої оболонки (рис. 2е).

4 Розробка гібридного CPU-GPU алгоритму

Відомо, що відеокарти мають значно більшу обчислювальну потужність порівняно з центральними процесорними елементами. Обчислювальні ядра графічного процесора працюють одночасно, що дозволяє застосовувати їх для розв'язання задач з великим обсягом даних. Створена компанією NVIDIA технологія CUDA *(Compute Unified Device Architecture)* покликана збільшити продуктивність звичайних комп'ютерів шляхом застосування обчислювальних потужностей відеопроцесорів [12].

Архітектура CUDA заснована на концепції SIMD *(Single Instruction Multiple Data)*, що передбачає можливість обробки сукупності даних однією функцією. Модель програмування передбачає об'єднання потоків *(thread)* у блоки *(block)*, а блоків – у сітку *(grid)*, що виконується одночасно. Відповідно, запорукою ефективного використання апаратних можливостей GPU є розпаралелювання алгоритму на сотні блоків, які здійснюють незалежні обчислення на відеокарті.

Відомо, що графічний процесор складається з кількох кластерів. В кожному з них розташовані текстурний блок та два потокових мультипроцесори, які містять по 8 обчислювальних пристроїв та 2 суперфункціональних блока. Крім того, мультипроцесори мають власні ресурси розподіленої пам'яті (16 КБ), що може бути застосована в якості програмованого кешу для зниження затримок при доступі обчислювальних блоків до даних [12, 13].

З наведених особливостей архітектури CUDA можна зробити висновок про потребу реалізації на відеокартах масивно-паралельних частин алгоритму, в той час як послідовні інструкції повинні виконуватися

на центральному процесорі. Відповідно, придатним до впровадження на GPU є етап формування частинних розв'язків, оскільки операції для кожного з численних блоків здійснюються незалежно.

Відомо, що функція, призначена для виконання на графічному процесорі, називається ядром *(kernel)*. Ядро новітнього алгоритму містить набір інструкцій для формування локальної оболонки будь-якого блока вихідного графа. Розрізнення окремих підзадач при цьому відбувається лише за допомогою номеру поточного потоку. Таким чином, розроблений гібридний алгоритм має наступні етапи виконання:

1. На центральному процесорі відбувається обчислення допоміжної матриці. Програма надсилає до відеокарти індекси комірок, які подолали процедуру первинної фільтрації та відповідні набори вершин.
2. На графічному процесорі відбувається формування частинних розв'язків, які записуються до глобальної пам'яті та надсилаються до центрального процесора.
3. Надалі здійснюється процедура об'єднання локальних оболонок та одержання загального результату.

Слід зазначити, що важливим недоліком гібридних алгоритмів є необхідність копіювання даних з центрального процесора до графічного та навпаки, що призводить до значних часових затримок [13]. Суттєве зменшення комунікаційних витрат відбувається за рахунок фільтрації вузлів заданого графа.

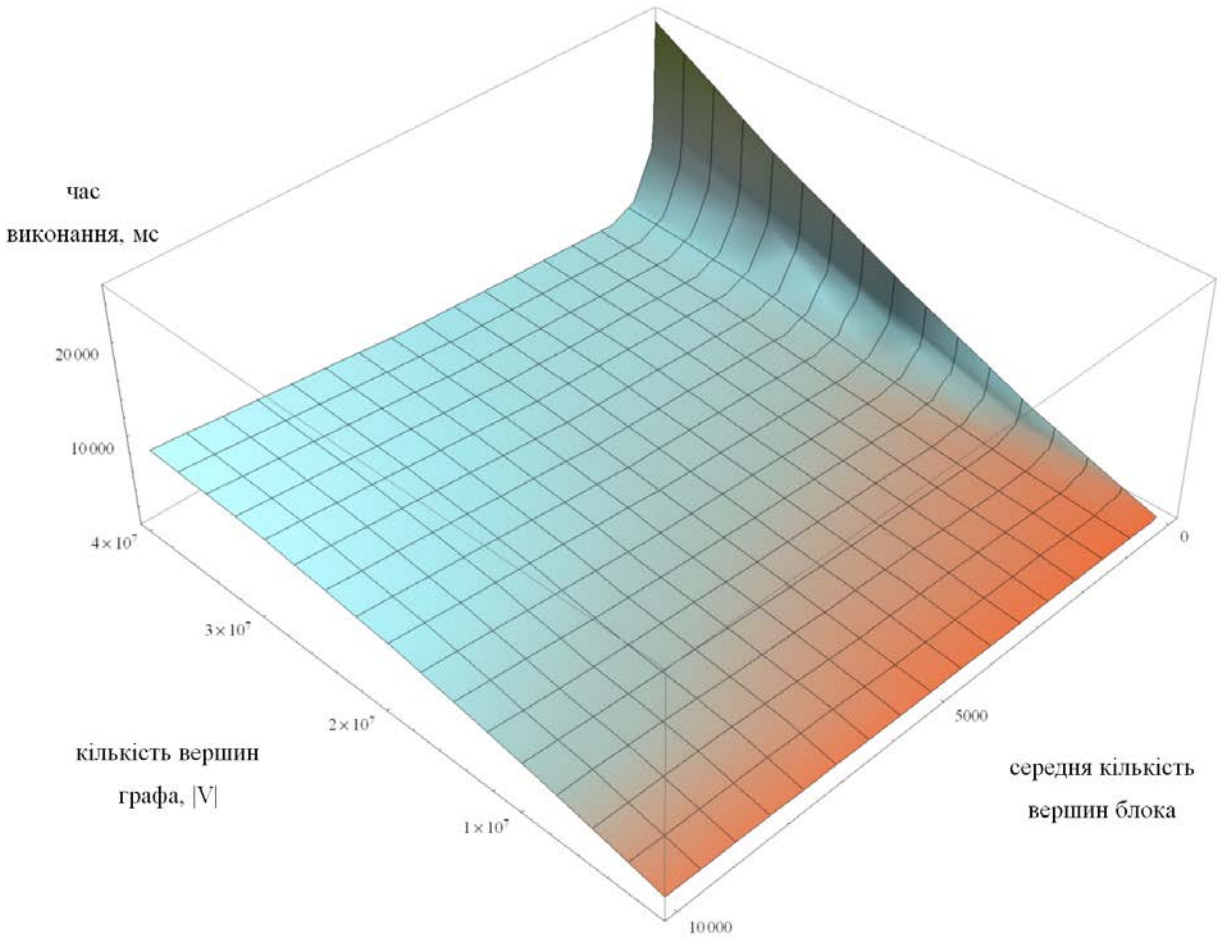

Рис. 3. Залежність часу виконання алгоритму від розмірності графа та кількості вершин виділених блоків

При розробці високопродуктивних алгоритмів для GPU важливою є організація правильного використання ресурсів пам'яті. Відомо, що звертання до глобальної відеопам'яті пов'язані зі значними затримками у декілька сотень тактів. Тому в розробленому алгоритмі вона застосовується лише в якості засобу комунікації між процесором та відеокартою. Результати проміжних обчислень для кожного з потоків записуються до розподіленої пам'яті *(shared memory)*, швидкість доступу до якої є значно вищою і складає 2 – 4 такти.

5 Експериментальні дослідження запропонованого алгоритму

Випробування проводилися для обчислювальної системи з процесором Intel Core i7-3610QM (2.3 ГГц), 8 ГБ RAM типу DDR3-1600 та відеокартою nVidia GeForce GT 630M (2ГБ відеопам'яті). При цьому графічний прискорювач містить 96 ядер CUDA, його тактова частота складає 800 МГц.

Відомо, що зі збільшенням розмірності оброблюваних графів кількість виділених блоків зростає лінійно. За аналогічним принципом збільшується складність обчислення відповідних допоміжних матриць. Багатоетапний процес фільтрації та відбір фрагментів локальних оболонок забезпечують значне спрощення операцій об'єднання частинних розв'язків до загального. Таким чином, складність розробленого алгоритму є лінійною $O(n)$.

Найгіршими для дослідження є графи, всі вузли яких входять до складу МОО. В цьому випадку операції фільтрації не надають необхідного прискорення і складність алгоритму дорівнює $O(n\log n)$. Однак, такі приклади мають суто теоретичне значення і на практиці не зустрічаються.

На рис. 3 наведено залежність часу виконання новітнього алгоритму від розмірності графа та кількості вузлів у виділених блоках. Отримані результати підтверджують лінійну складність запропонованого методу. Крім того, важливим є встановлення оптимального значення розмірності підмножин, виділених у вихідному графі. Вибір дрібних блоків (до 1000 вузлів) призводить до катастрофічного зростання часу роботи алгоритму.

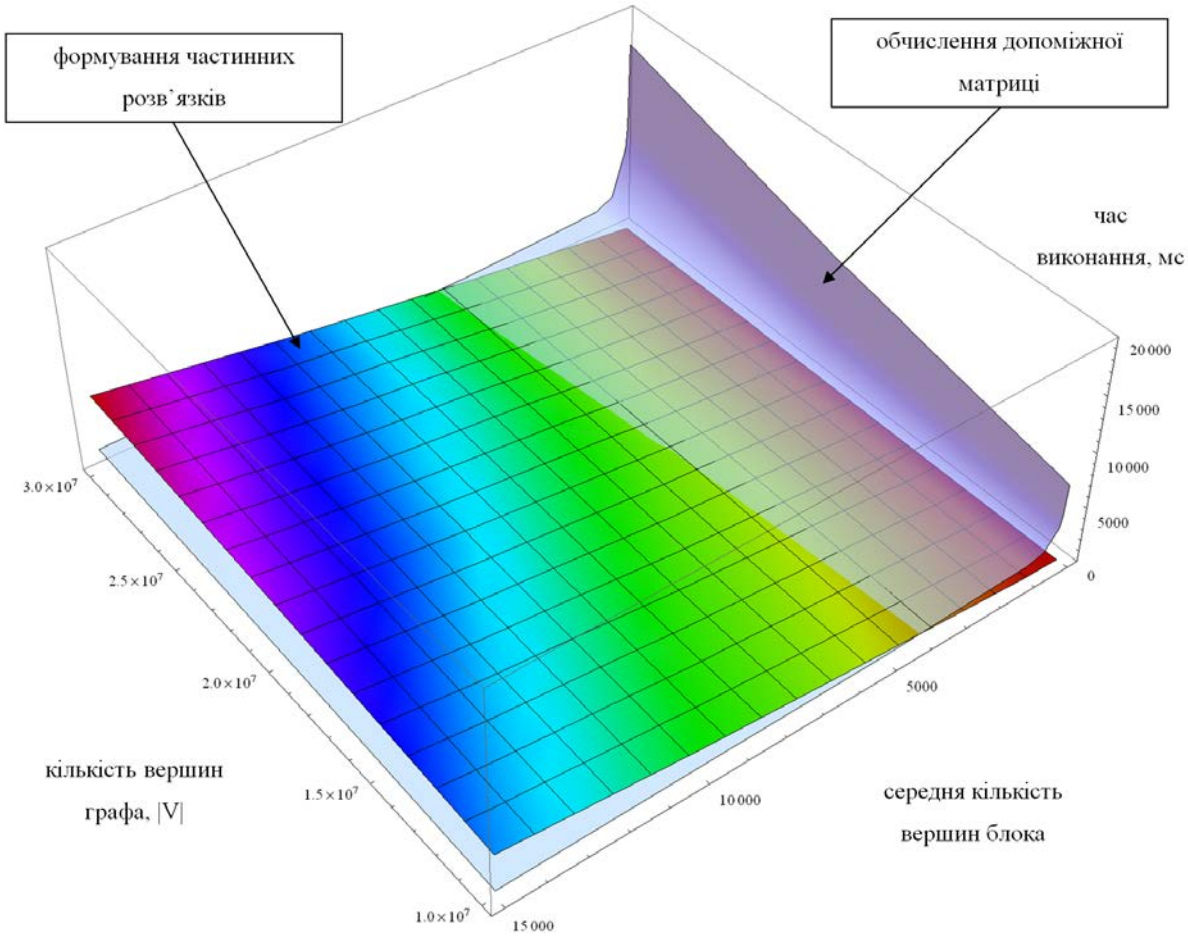

Рис. 4. Залежності часу виконання різних етапів алгоритму від розмірності графа та кількості вузлів виділених блоків

Причиною цього явища є значне збільшення розмірності допоміжних матриць, що ускладнює керування обчислювальним процесом (рис. 4). Проте, виділення великих блоків (більше 5000 вершин) спричинює втрату масивного паралелізму, неефективне використання ресурсів відеокарти та, як наслідок зростання часу виконання алгоритму. Таким чином, найшвидша робота запропонованого методу спостерігається за проміжних

значень розмірності блоків (1000 – 5000 вузлів). При цьому допоміжні матриці є порівняно невеликими, а другий етап алгоритму зберігає властивості масивного паралелізму.

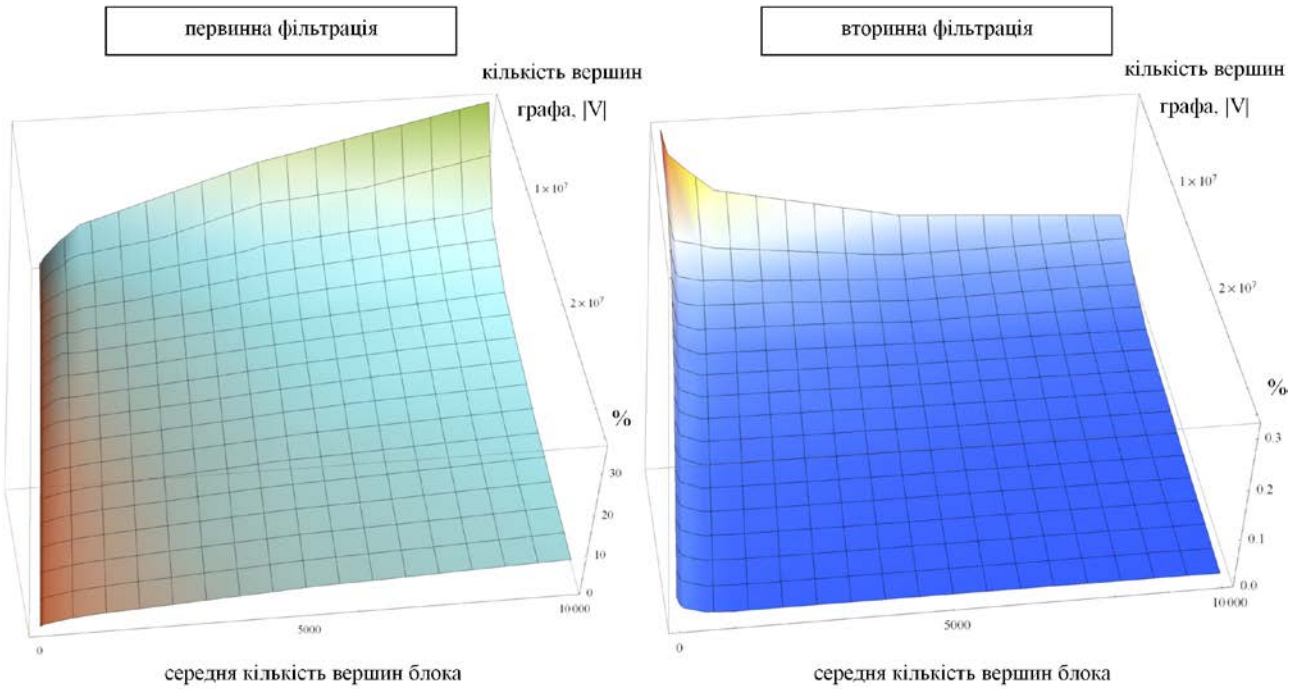

Рис. 5. Вплив фільтрації вузлів на зменшення розмірності вихідної задачі

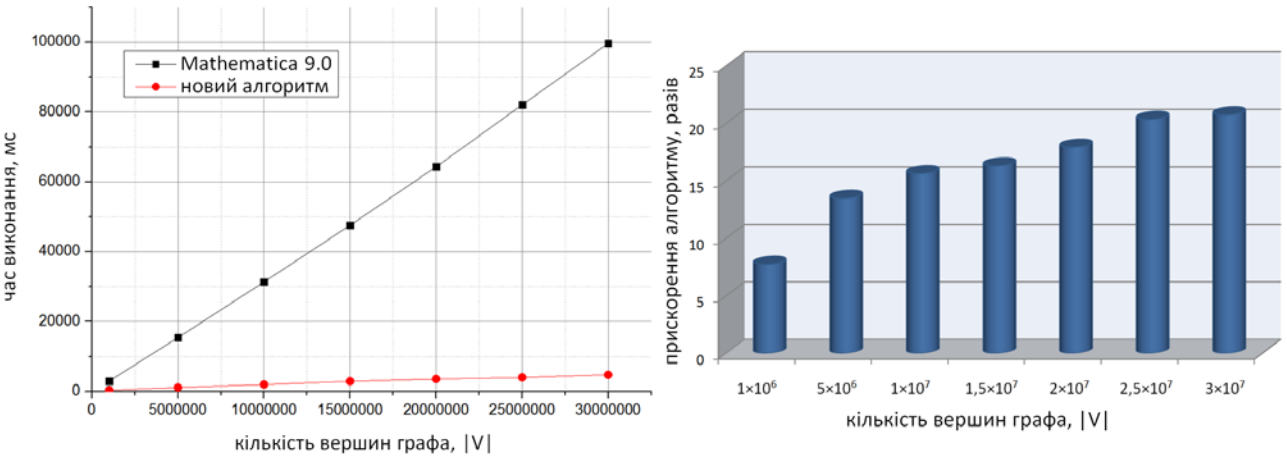

Рис. 6. Порівняння часу виконання новітнього алгоритму та вбудованих засобів пакету Wolfram Mathematica 9.0

Важливим засобом забезпечення швидкодії алгоритму є здійснення багатокрокової фільтрації вершин графа. На рис. 5 ліворуч наведена залежність якості первинного відбору від розмірності задачі та виділених підмножин. Отримані результати демонструють, що найбільш ефективно така фільтрація відбувається в разі застосування невеликих блоків. Крім того, кількість відібраних вузлів збільшується зі зростанням числа вузлів графа, забезпечуючи швидке розв'язання задач надвеликої розмірності. Таким чином, наступні операції алгоритму застосовуються лише до 1 – 3% початкових вершин.

Протилежними є результати виконання вторинного етапу фільтрації, наведені на рис. 5 праворуч. В цьому випадку найбільша якість відбору спостерігається при виділенні великих блоків. Однак, здійснення вторинної фільтрації відбувається значно повільніше, ніж первинної, тому найбільш ефективний відбір відбувається за проміжних значень розмірності блока. При цьому лише 0,05 – 0,07% вузлів початкового графа беруть участь в останньому етапі роботи алгоритму.

Для з'ясування ефективності нового алгоритму виконано його порівняння з вбудованими засобами математичного пакету *Wolfram Mathematica 9.0*. Випробування проводилися для графів, сформованих випадковим чином. В пакеті *Mathematica* для знаходження МОО застосовувалася функція *ConvexHull[]*, а визначення швидкодії здійснювалося за допомогою виразу *Timing[]*. Результати проведеного порівняння містяться на рис. 6. З них випливає, що новий алгоритм здійснює формування оболонок в 10 – 20 разів швидше, ніж стандартні функції програми *Mathematica*.

6 Висновки

У статті запропоновано новітній алгоритм формування мінімальних опуклих оболонок, який спирається на технологію GPGPU і використовує графічні прискорювачі. На відміну від своїх попередників, такий алгоритм є адаптованим до розв'язання задач великої розмірності та, як наслідок, придатним до застосування в умовах напряму *Big Data*. В порівнянні з класичними методами він має низку наступних переваг:

1. **Висока швидкість роботи та лінійна складність.** Для збільшення швидкодії в алгоритмі передбачені етапи розподілення вершин графа за блоками, здійснення фільтрації та застосування допоміжних матриць. Як наслідок, швидкість нового методу є в 10 – 20 разів вищою порівняно з використанням професійного математичного пакету *Mathematica*.
2. **Масовий паралелізм.** Формування частинних оболонок відбувається незалежно, що сприяє реалізації цього етапу алгоритму з використанням графічних процесорів.
3. **Можливість динамічної перебудови оболонок.** При додаванні нових вузлів до початкового набору обчислення здійснюються лише для блоків, що зазнали модифікації. Результати для інших частин графа залишаються попередніми.
4. **Можливість узагальнення на багатовимірні випадки.** При цьому виділені блоки являють собою *n*-вимірні куби, до яких застосовуються операції розробленого методу.

Наведені переваги можуть слугувати підставою для включення новітнього алгоритму до складу професійних математичних пакетів з метою поширення високопродуктивних обчислень серед їх користувачів.

Подальший напрямок досліджень пов'язаний з розробкою гібридних CPU+GPU версій алгоритму для складних систем з багатьма процесорами та відеокартами.

Перелік посилань

[1] *Min Tang, Jie-yi Zhao, Ruo-feng Tong, Dinesh Manocha*, Performance GPU accelerated Convex Hull Computation // Computers & Graphics. – 2012, Volume 36, Issue 5. – P. 498–506.
[2] Thomas W. Sederberg, Computer aided geometric design course notes. – 2011.
[3] *Preparata, D. F., Shamos, M.*, Computational Geometry. An Introduction. – Berlin: Springer, 1993. – 390 p.
[4] IBM: Next generation scalable storage [Electronic Resource]. – Mode of access: URL: http://www-03.ibm.com/systems/information_infrastructure/leadership/next_gen_storage.html.
[5] IBM: Analytics: The real-world use of big data [Electronic Resource]. – Mode of access: URL: http://www-03.ibm.com/systems/hu/resources/the_real_word_use_of_big_data.pdf.
[6] *Glaskowsky P. N.*, NVIDIA's Fermi: The First Complete GPU Computing Architecture. – NVIDIA, 2009. – 26 p.
[7] *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein*, Introduction to Algorithms, Second Edition. MIT Press, 2001. ISBN 0-262-03293-7. Section 33.3: Finding the convex hull, P. 947 – 957.
[8] *Chan T. M.*, Optimal output-sensitive convex hull algorithms in two and three dimensions // Discrete & Computational Geometry. – 1996, v. 16. P. 361 – 368.
[9] *Akl S.G., Toussaint G.T.*, A fast convex hull algorithm // Information processing letters. – 1978. – Vol. 7. – P. 219 – 222.
[10] *M. Zahid Hossain, M. Ashraful Amin*, On Constructing Approximate Convex Hull // American Journal of Computational Mathematics. – 2013. v. 3. – P. 11 – 17.
[11] *Ladislav Kavan, Ivana Kolingerova*, Jiri Zara, Fast approximation of convex hull // ACST'06 Proceedings of the 2nd IASTED international conference on Advances in computer science and technology. – 2006. – P. 101 – 104.
[12] *Jason Sanders, Edward Kandrot*, CUDA by Example: An Introduction to General-Purpose GPU Programming. – Addison-Wesley Professional, 2010. – 312 p.
[13] *Govindaraju, N.K., Larsen, S., Gray, J., Manocha, D.* A memory model for scientific algorithms on graphics processors // Proceedings of the 2006 ACM/IEEE conference on Supercomputing, 2006.