УДК 519.6

ЕФЕКТИВНІСТЬ ДВОВИМІРНИХ БЛОЧНО-ЦИКЛІЧНИХ ПАРАЛЕЛЬНИХ АЛГОРИТМІВ

О.М. Хіміч, В.В. Полянко

Інститут кібернетики ім. В.М. Глушкова НАН України,
03680, Київ, проспект Академіка Глушкова, 40.
Тел.: 8(044)526 1196, dept150@insyg.kiev.ua

Досліджено прискорення двовимірних блочно-циклічних паралельних алгоритмів розв'язування систем лінійних алгебраїчних рівнянь та їх ефективна реалізація. Розглянуто у порівнянні різні способи розподілу даних по процесорам. Теоретичні викладки підтверджені результатами експериментальних досліджень.

An acceleration of two-dimensional block-cyclic parallel algorithms of solving of system of linear algebraic equations and their effective realization are researched. Different ways of data distribution among processors are considered in comparison. The theoretic expositions are confirmed by results of experimental researches.

Еволюція паралельних алгоритмів розв'язування СЛАР

Більшість відомих паралельних алгоритмів за деяким виключенням (наприклад, у випадку стрічкових матриць з вузькою шириною стрічки при розв'язуванні систем лінійних рівнянь), є прототипом відомих послідовних методів. При такому підході фактично в кожному процесорі одночасно реалізуються послідовні обчислювальні схеми над локальним блоком даних, що одержаний в результаті декомпозиції вихідних даних за тим чи іншим способом. Принципово важливо, що при такому адаптивному підході для паралельних алгоритмів зберігається наступництво результатів, одержаних в теорії обчислювальних методів (точність, збіжність, складність та ін.).

З цієї точки зору характерна еволюція, яку зазнали паралельні алгоритми (в задачах лінійної алгебри), що базуються на елементарних односторонніх і двосторонніх приведеннях Якобі (як відомо, не кращих за економічністю представників послідовних алгоритмів), до методів, що базуються на приведеннях Гівенса, Хаусхолдера – методам, які є головними в ієрархії списку за ефективністю серед ортогональних послідовних алгоритмів. Це пов'язано, з однієї сторони, з природнім паралелізмом алгоритмів, побудованих на обертаннях Якобі, а з іншої сторони, з впливом ефекту Гайдна [1], що знижує ефективність методів Гівенса та Хаусхолдера для задач лінійної алгебри при звичайному способі збереження (одновимірним блочним стовпчиковим способом розподілу). В цьому випадку кожний процесор містить лише один блок стовпчиків матриці. Стовпчик k розміщений у процесорі за номером k/tc, де $tc = n/p$ – максимальна кількість стовпчиків, розміщених у процесорі. Ця схема не дає достатнього балансування навантаження процесорів, тому що, як тільки перші tc стовпчики буде оброблено, 0-ий процесор виходить з роботи. Аналогічно не дає доброго балансування навантаження процесорів одновимірний блочний рядковий спосіб розподілу. Вирішенням питань покращення ефективності алгоритмів є циклічний спосіб декомпозиції вихідних даних та відповідної їх обробки.

Значним кроком до покращення балансування навантаження процесорів, а, отже, і ефективності обчислювальних алгоритмів є ідея циклічного способу збереження і обробки матриць, що приводить до збалансованої схеми алгоритму. Ідея стовпчиково-циклічного (рядково-циклічного) способу збереження та обробки матриць була незалежно проголошена в деяких роботах (див., наприклад, [2], в тому числі і в роботі авторів [3]). У відповідності, наприклад, з рядково-циклічною схемою матриця розподіляється по p процесорах так: в i-му процесорі розміщуються рядки за номерами $i, i+p, i+2p, ..., i+(p-1)p$. Ця схема, зберігаючи такий же, як і для послідовного алгоритму порядок обробки рядків (стовпчиків), передбачає зміну на одиницю номера процесора при переході від одного рядка (стовпчика) до наступного за ним. Отже, досягається майже однаковий обсяг обчислень у кожному процесорі при реалізації алгоритмів, тобто практично виключається вплив ефекту Гайдна. Третя схема, блочно-циклічна стовпчикова**,** є компромісом між дистрибутивними вище наведеними схемами. Вибираючи розмір блоку nb, ділимо стовпчики блоками розмірністю nb і розподіляємо їх циклічним способом. Це означає, що стовпчик k зберігається в процесорі за номером $[(k-1)/\ nb]\ mod\ p$. Фактично, ця схема включає в собі перші дві схеми зберігання даних при $nb = tc = n/p$ и $nb = 1$. Для nb більшого, ніж 1, це дає дещо гірше балансування процесорів у порівнянні з циклічним розподілом стовпчиків, але при цьому зменшується загальний час латентності системи, оскільки зменшується кількість обмінів між процесорами.

Схема двомірного блочно-циклічного розподілу стовпчиків включає у собі попередні схеми як окремі випадки. Одним з важливих факторів доцільності такого розподілу та обробки матриць одночасно з добрими показниками балансування і латентності є можливість використання процедур стандартних бібліотечних програм типу BLAS, у тому числі бібліотеки BLAS 3 [3].

Блочно-циклічний розподіл матриці

Спосіб розподілу даних між процесорами має велике значення для балансування та комунікаційних характеристик паралельного алгоритму. Блочно-циклічний варіант надає простий, але ефективний спосіб розподілу матриці.

Розглядаємо комп'ютер MIMD-архітектури з розподіленою пам'яттю та системою комунікаційних зв'язків типу «решітка». Подамо вхідну матрицю A розмірністю $n \times n$ у вигляді

$$A = \begin{pmatrix} A^*_{11} & A^*_{12} & \dots & A^*_{1p} \\ A^*_{21} & A^*_{22} & \dots & A^*_{2p} \\ \dots & \dots & \dots & \dots \\ A^*_{q1} & A^*_{q2} & \dots & A^*_{qp} \end{pmatrix}, \text{ де } A^*_{ij} = \begin{pmatrix} A_{ij} & A_{ij+p} & \dots & A_{ij+(p-1)p} \\ A_{i+qj} & A_{i+qj+p} & \dots & A_{ij+(p-1)p} \\ \dots & \dots & \dots & \dots \\ A_{i+(q-1)qj} & A_{i+(q-1)qj+p} & \dots & A_{i+(q-1)qj+(p-1)p} \end{pmatrix},$$

де A_{ij} – блок елементів матриці A розмірністю $s \times s$.

Розташуємо матрицю по процесорах так, щоб $A^*_{ij} \in P_{ij}$, де i та j – декартові координати процесора на решітці, розмірністю pxq. Не втрачаючи загальності, припускатимемо, що $n/(spq)$ – ціле число.

Для ілюстрації блочно-циклічного розподілу, наведемо приклад. Нехай, $n = 8$, $s = 2$, $p = 2$, $q = 2$. Тоді матриця A матиме вигляд

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{18} \\ a_{21} & a_{22} & \dots & a_{28} \\ \dots & \dots & \dots & \dots \\ a_{81} & a_{82} & \dots & a_{88} \end{pmatrix} \text{ або } A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{34} & A_{44} \end{pmatrix},$$

де $A_{ij} = \begin{pmatrix} a_{2i-1,2j-1} & a_{2i-1,2j} \\ a_{2i,2j-1} & a_{2i,2j} \end{pmatrix}$.

Блочно-циклічний розподіл для A має вигляд $A = \begin{pmatrix} A^*_{11} & A^*_{12} \\ A^*_{21} & A^*_{22} \end{pmatrix}$, де $A^*_{11} = \begin{pmatrix} A_{11} & A_{13} \\ A_{31} & A_{33} \end{pmatrix}$,

$A^*_{12} = \begin{pmatrix} A_{12} & A_{14} \\ A_{32} & A_{34} \end{pmatrix}$, $A^*_{21} = \begin{pmatrix} A_{21} & A_{23} \\ A_{41} & A_{43} \end{pmatrix}$, $A^*_{22} = \begin{pmatrix} A_{22} & A_{24} \\ A_{42} & A_{44} \end{pmatrix}$.

Тоді кожен з процесорів P_{ij} містить елементи матриці, що знаходяться у A^*_{ij}, тобто $A^*_{11} \in P_{11}$, $A^*_{12} \in P_{12}$, $A^*_{21} \in P_{21}$, $A^*_{22} \in P_{22}$.

Алгоритм *LU*-факторизації

При розв'язуванні системи лінійних алгебраїчних рівнянь $Ax=b$ алгоритм LU-факторизації зводить матрицю A до вигляду $A=PLU$, де P – вектор перестановок, L – нижня трикутна (з одиницями на головній діагоналі), U – верхня трикутна. Це дає змогу замінити розв'язування однієї системи з щільною матрицею, на розв'язування двох $Ly=b$, $Ux=y$ з трикутними матрицями.

Вважатимемо, що матриці A, L та U мають розмірність nxn. Дані зберігаються за блочно-циклічною схемою. Розмірність блоків становить sxs.

Спочатку розглянемо послідовний алгоритм [4]. Припускаємо, що знаходимося на $k1$-у кроці перетворення матриці, де $k1=k*s$, $k=0,1,2,..$ Розіб'ємо підматрицю розмірності $n–k*s \times n–k*s$, що містить останні $n-k*s$ рядки та $n-k*s$ стовпчики матриці, на чотири частини, як показано на схемі:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = P \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = P \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{pmatrix},$$

де блок A_{11} розмірністю $s \times s$, A_{12} – $s \times (n–(k+1)s)$, A_{21} – $(n–(k+1)s) \times s$, A_{22} – $(n–(k+1)s) \times (n–(k+1)s)$. Проводимо послідовність Гаусових перетворень на частині матриці, яку складають блоки A_{11} та A_{21}. У результаті отримаємо матриці L_{11}, L_{21} і U_{11}. За ними знаходимо невідомі U_{12} та $\hat{A}_{22}$ (остання утворюється на місці A_{22}):

$$U_{12} \leftarrow (L_{11})^{-1} A_{12}$$

$$\hat{A}_{22} \leftarrow A_{22} - L_{21}U_{12} = L_{22}U_{22}$$

У силу невеликого розміру блоку, обчислення оберненої матриці і множення матриць розмірністю s, вимагають менше часу, ніж проведення факторизації на всій підматриці.

Далі, якщо $k1<n$, то збільшуємо k на одиницю і повторюємо обчислення для нового значення k, інакше факторизацію матриці A вважатимемо завершеною.

Перейдемо до розгляду паралельного алгоритму.

Розглянемо роботу алгоритму на k–у кроці, акцентуючи увагу на міжпроцесорних обмінах. Частину матриці, для якої перетворення елементів ще не завершено, поділимо на чотири частини A_{11}, A_{12}, A_{21} і A_{22} , як вищезазначено. Назвемо процесор, що містить блок A_{11}, ведучим процесором. Алгоритм має такий вигляд:

1) відбувається факторизація блоків у A_{11} та A_{12} у циклі з s кроків. На кожному кроці проводиться пошук максимального елемента у ведучому стовпчику, обмін елементів ведучого рядка та рядка з максимальним елементом, перетворення ведучого рядка та всіх наступних рядків. При пошуку максимального елементу та виконанні перетворень елементів відбуваються обміни між процесорами, що містять частини A_{11} або A_{12}, а при перестановці ведучого рядка та рядка з максимальним елементом задіяні процесори двох рядків процесорної сітки, які містять відповідні елементи матриці. Після завершення на місці блоку A_{11} виникають L_{11} і U_{11}, а на місці A_{21} – L_{21};

2) проводиться пошук матриці $(L_{11})^{-1}$. Оскільки L_{11} повністю міститься у блоці A_{11}, то обчислення $(L_{11})^{-1}$ відбуваються в межах одного процесора. Після цього перетворений блок розсилається усім процесорам відповідного рядка процесорної сітки. Тобто, якщо P_{ij} ведучий процесор, то $(L_{11})^{-1}$ посилається процесорам $P_{il}, l = \overline{1, p}$;

3) процесори незалежно один від одного обчислюють добуток отриманого блоку $(L_{11})^{-1}$ на відповідні власні блоки, що знаходяться у A_{12}. У результаті, елементи A_{12} заміщаються елементами U_{12};

4) блоки з L_{21} розсилаються горизонтально по рядкам процесорної сітки, а U_{12} – вертикально по її стовпчикам. Після цього у кожному процесорі відбувається поблочне перетворення відповідних блоків матриці A_{22}.

Ефективність та прискорення алгоритму

Обчислимо коефіцієнти прискорення та ефективності наведеного алгоритму: $S_{pq}=T_1/T_{pq}$, $E_{pq}=S/pq$, де $p\times q$ – розмірність процесорної сітки, T_1 – час виконання на одному процесорі, T_{pq} – час виконання на pxq процесорах. Часи виконання обчислюватимемо як $T_1=Nt$, $T_{pq}=Nt+Mt_o+Qt_c$, де N – кількість арифметичних операцій (додавання та множення), t – час виконання однієї арифметичної операції, M – кількість обмінів, t_o – час виконання одного обміну, Q – кількість синхронізацій між процесорами, t_c – час однієї синхронізації.

Знайдемо кількість обмінів, що відбуваються на k-му кроці алгоритму. При перетворенні ведучого блоку потрібно надіслати усім процесорам значення максимального елементу та номер процесора – $2(q-1)+2(p-1)$ елементів, для розсилки ведучого рядка з номером процесора – $(q-1)(1+n/q)$, і, у випадку, коли максимальне значення знаходиться не у ведучому процесорі, ще $1+n/q$ елементів. Оскільки перетворення ведучого блоку відбувається у циклі з s кроків, то слід провести пересилання $M_1 \approx (3q+2p+n-4)s$ елементів. Для розсилки оберненої матриці, відповідно, $M_2=(p-1)s^2$, а при обчисленні частини матриці, що відповідає A_{22} у послідовному алгоритмі $M_3=((p-1)(n-ks)/q+(q-1)(n-ks)/q)s$ пересилок елементів. Тоді за k кроків отримуємо $M=\sum_{k=1}^{n/s}(M_1+M_2+M_3) \approx (n^2(p+q)(p+q-1))/(2pq)$. При цьому, на кожному кроці виконується $Q_1=4(p-1)+4(q-1)$ синхронізацій. Отже, загальна кількість складатиме $Q=\sum_{k=1}^{n/s}Q_1=4n(p+q-2)/s$.

Порахуємо кількість арифметичних операцій (множення та ділення) на k-му кроці для одно процесорного випадку. Для перетворення A_{11} та A_{21} і пошуку обернених матриць потрібно виконати $N_1 \approx 2/3(n-ks)^2s$ операцій, для множення оберненої $(L_{11})^{-1}$ на блоки A_{12} необхідно $N_2 \approx 2(n-ks)s^2$, а для перетворення блоку A_{22} необхідно $N_3 \approx 2(n-ks)^2s$ арифметичних операцій. Отже, за всі n/s кроків кількість арифметичних операцій, які необхідно виконати, становитиме $N=\sum_{k=1}^{n/s}(N_1+N_2+N_3) \approx 2\sum_{k=1}^{n/s}(1/3s^3+(n-ks)s^2+(n-ks)^2s) = 2s^2n/3+2n^2s+2n^2-sn(n-s)-2n^2(n-s)+n(n+s)(2n+s)/3 \approx 2/3n^3$. Тобто, загальний час роботи для однопроцесоного варіанту алгоритму дорівнює $T_1 \approx 2/3n^3t$.

Коли задача розв'язується на сітці з pxq процесорів, то кількості арифметичних операцій, вище введені, матимуть інші значення: $N_1 \approx 2/3s^3$, $N_2 \approx 2(n-ks)s^2/p$, $N_3 \approx 2(n-ks)^2s/(pq)$. Тому, на всіх n/s кроках необхідно виконати $N \approx 2n^3/(3pq)$ арифметичних операцій. Отже, маємо

$$T_{pq}=2n^3t/(3pq)+n^2(p+q)(p+q-1)t_o/(2pq)+4n(p+q-2)t_c/s,$$
$$S_{pq}=T_1/T_{pq}=pq(1+3(p+q)(p+q-1)\tau_o/(4n)+6pq(p+q-2)\tau_c/(sn^2))^{-1},$$
$$E_{pq}=S_{pq}/pq=(1+3(p+q)(p+q-1)\tau_o/(4n)+6pq(p+q-2)\tau_c/(sn^2))^{-1}.$$

Наведені формули дають змогу провести аналіз алгоритму та його параметрів. При збільшенні кількості процесорів прискорення паралельного двовимірного блочно-циклічного алгоритму зростає. Проте значне покращення у часі виконання настає при вдалому підборі розміру блоку, якщо обчислення, що проводяться в одному блоці повністю вміщаються у кеш-пам'ять ЕОМ. Це можливо з тієї причини, що швидкість виконання

операцій з даними, що містяться у кеш-пам'яті значно більша за швидкість виконання операцій з даними у оперативній пам'яті. Але задача такого підбору розмірів блоку, при якому вдається досягти цього ефекту, містить у собі додаткові труднощі, оскільки у такому випадку необхідно враховувати особливості архітектури ЕОМ, на якій працює програма.

Отримані залежності коефіцієнтів прискорення та ефективності від розмірності матриці та параметрів процесорної сітки вказують на те, що не завжди збільшення числа процесорів веде до зменшення часу виконання програми. Так, при відносно невеликих розмірностях матриці, на обміни даними між процесорами припадає значна частина часу роботи програми, тим більша, чим більша кількість процесорів.

З вигляду отриманих формул можна зробити висновки – залежність коефіцієнта ефективності та прискорення від розмірності процесорної сітки. Так, при одній і тій же загальній кількості процесорів сума $p+q$ може мати різні значення. Очевидно, що для наведеного алгоритму для більшого прискорення та ефективності доцільно брати такі розміри сітки, щоб ця сума була мінімальною, тобто, з якомога меншою різницею між горизонтальною та вертикальною складовими.

Результати експериментальних досліджень

Двовимірний бочно-циклічний алгоритм апробовано на паралельному MIMD-комп'ютері Ін парком [5]. На графіку, показаному на рис. 1, на основі отриманих результатів виведено залежність прискорення алгоритму розв'язування системи лінійних алгебраїчних рівнянь з матрицею розмірності 10000 від кількості залучених процесорів. На рис. 2 показано зміну значень ефективності з ростом кількості процесорів.

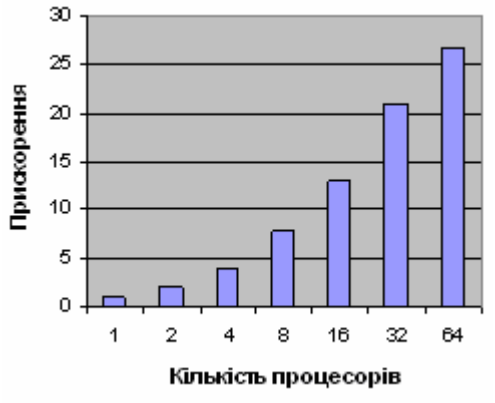

Рис. 1. Прискорення при n = 10000

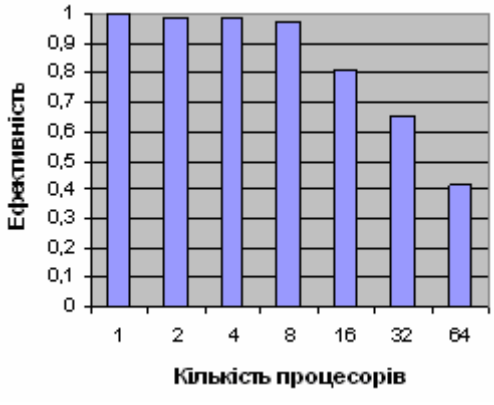

Рис. 2. Ефективність при n = 10000

З наведених графіків видно, що при збільшенні кількості використовуваних процесорів, прискорення зростає. Причому, найбільший ріст прискорення відбувається вже при використанні невеликої кількості процесорів.

Проте, якщо порядок системи лінійних алгебраїчних рівнянь не достатньо великий, то подальше нарощування кількості процесорів не даватиме зменшення часу. Це твердження показано графіками на рис. 3 та рис. 4.

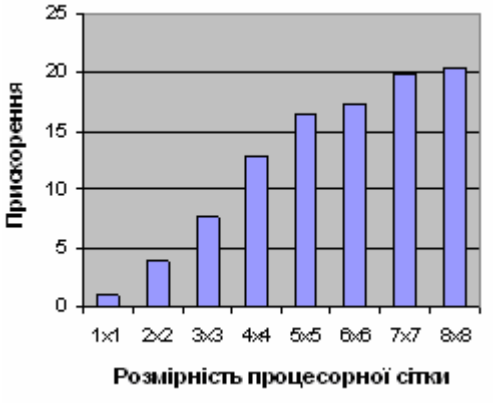

Рис. 3. Прискорення при n = 8000

Рис. 4. Ефективність при n = 8000

У цьому випадку розмірність матриці складала 8000. Помітно, що починаючи з розмірності процесорної сітки у 5 процесорів, тобто, коли загальна кількість процесорів складає 25 одиниць, збільшення їх кількості суттєво не збільшує прискорення, а ефективність іде на спад. Отже використовувати для розв'язанні задачі максимально можливу кількість процесорів доцільно лише для задач досить великих розмірностей.

Варіюючи дані програми, побудованої на основі описаного алгоритму, було визначено деякі залежності прискорення та ефективності від розмірів блоку. На рис. 5 показано залежність прискорення при сталій кількості процесорів. Різниця між кращим та гіршим варіантами досить суттєва, для них час виконання

відрізняється майже у два рази. Твердження, що найкращим відношенням сторін є таке, при якому різниця їх значень мінімальна, підтверджують і отримані дані щодо ефективності, показані на рис. 6.

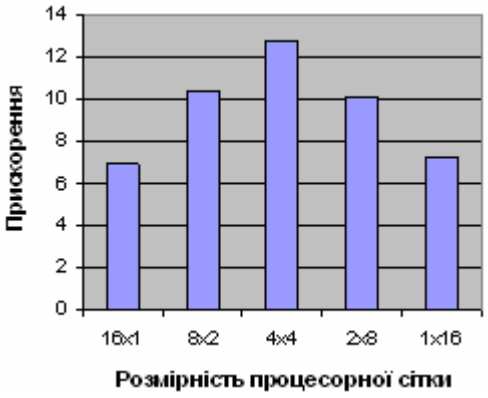

Рис. 5. Залежність прискорення від відношення сторін

Рис. 6. Залежність ефективності від відношення сторін

На рис. 7 відображено залежність часу виконання програми від розміру блоку. Щоб показати саме цю залежність, усі запуски проводилися на однаковій кількості процесорів. Зміна лише одного параметра, а саме, розміру блоку, на які розбивається матриця, може привести до значної зміни часу виконання.

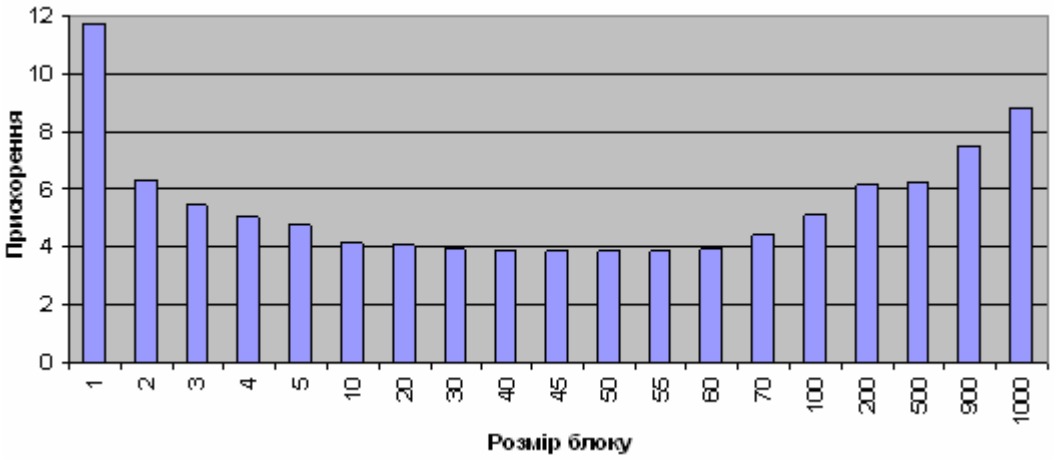

Рис. 7. Залежність прискорення від розміру блоку

Наведені результати досліджень підтвердили отримані теоретичним шляхом дані про прискорення та ефективність паралельного двовимірного блочно-циклічного алгоритму, їхні залежності від різних параметрів алгоритму.

1. *Брусникин Б.Н., Визнюк Г.И., … Химич А.Н. и др*. Решение на MIMD-компьютере некоторых задач вычислительной математики //Кибернетика и вычислительная техника: Республиканский межведомственный сборник научных трудов.– Киев: Ин-т кибернетики им. В.М. Глушкова АН Украины. – 1992. – Вып. 93. – С. 94–100.
2. *Ортега Дж*. Введение в параллельные и векторные методы решения линейных систем. – М.: Мир, – 1991. – 368 с.
3. *Михалевич В.С., Бик Н.А, … Химич А.Н. и др.* Численные методы для многопроцессорного вычислительного комплекса ЕС // Под ред. И.Н. Молчанова.– М.: Издание ВВИА им. Жуковского, – 1986. – 401 с.
4. *Jaeyoung Choi, Jack J. Dongarra, L. Susan Ostrouchov & others* The design and implementation of the scalapack LU, QR and Cholesky factorization routines. – Oak Ridge: Oak Ridge National Laboratory, – 1994. – 22 p.
5. *Химич. А.Н., Молчанов. И.Н., Мова В.И. и др.* Численное программное обеспечение MIMD–компьютера Инпарком. – Киев: Наук. думка, 2007. – 221 с.