УДК 004.932.4

С.Д. Погорілий, д-р техн. наук, проф.,
Б.В. Мигашко, студент
Київський національний університет ім. Тараса Шевченка
sdp77@i.ua
tuxoh992@gmail.com

Підхід до оцінки ефективності паралельної роботи програмно-апаратної платформи

Створено програмний комплекс для дослідження програмно-апаратної платформи Java. Запропоновано використання алгоритму Джонсона для тестування систем. Виконано формалізацію алгоритму засобами модифікованої системи алгоритмічних алгебр Глушкова. Розроблено паралельну схему алгоритму. Виконано експериментальне дослідження підвищення продуктивності при проведенні обчислень на різних платформах.

Ключові слова: benchmark, Java, алгоритм Джонсона, САА-М, паралелізація.

Вступ

При визначенні обчислювальної потужності суперкомп′ютера існує низка складнощів. По-перше, слід мати на увазі, що продуктивність системи може сильно залежати від типу виконуваного завдання. Зокрема, негативно позначається на обчислювальній потужності необхідність частого обміну даними між складовими комп′ютерної системи, а також часте звернення до пам′яті. Важливу роль відіграє також розрядність значень, що обробляються програмою (зазвичай мається на увазі формат дійсних чисел). Так, наприклад, у графічних процесорів NVIDIA Tesla перших двох поколінь максимальна продуктивність в режимі одинарної точності (32 біт) становить близько 1 терафлопс, однак при проведенні обчислень з подвійною точністю (64 біт) вона в 10 разів нижче.

Оцінка реальної обчислювальної потужності проводиться шляхом проходження спеціальних тестів (бенчмарків) — набору програм спеціально призначених для проведення обчислень і вимірювання часу їх виконання. Зазвичай оцінюється швидкість вирішення обчислювальною машиною великої системи лінійних алгебраїчних рівнянь, що обумовлюється, в першу чергу, хорошою масштабованістю цього завдання.

Найпопулярнішим тестом продуктивності є Linpack benchmark. Зокрема, HPL (альтернативна реалізація Linpack)[1] використовується при складанні Список Топ 500 суперкомп′ютерів у світі[2].

Іншими популярними програмами для проведення тестування є NAMD[4] (розв′язання задач молекулярної динаміки), HPCC (HPC Challenge benchmark), NAS Parallel Benchmarks[1].

Метою роботи є створення програмного комплексу для вимірювання продуктивності обчислювальних пристроїв, а також для оцінки приросту швидкодії при паралельній роботі. Як інструментальний засіб використовується мова програмування Java, яка підтримується на великій кількості платформ і має потужні вбудовані засоби паралельного програмування. В якості задачі пропонується знаходження найкоротшого шляху (англ. Shortest path problem) у графі за допомогою алгоритму Джонсона. Формалізацію алгоритму виконано за допомогою засобів модифікованої системи алгоритмічних алгебр Глушкова (САА-М).

Алгоритм Джонсона

Алгоритм Джонсона дозволяє знайти найкоротші шляхи між усіма парами вершин зваженого орієнтованого графа. Даний алгоритм працює, якщо в графі містяться ребра з позитивною або негативною вагою, але відсутні цикли з негативною вагою. Алгоритм Джонсона включає алгоритм Беллмана - Форда і Дейкстри.[4]

Алгоритм Беллмана-Форда як складова алгоритму Джонсона

Алгоритм Беллмана-Форда - алгоритм пошуку найкоротшого шляху в зваженому графі. За час O (| V | × | E |) алгоритм знаходить найкоротші шляхи від однієї вершини графа до всіх інших. На відміну від алгоритму Дейкстри, алгоритм Беллмана-Форда допускає ребра з негативною вагою.[4,7,8]

Як вихідні дані дано орієнтований або неорієнтовані граф зі зваженими ребрами G(V,E) де V – множина вершин графа, E – множина ребер графа. Довжиною шляху буде сума ваг ребер, що входять в цей шлях. Потрібно знайти найкоротші шляхи від вершини s до всіх вершин графа.

Зауважимо, що найкоротших шляхів може не існувати. Так, у графі, що містить цикл з негативною сумарною вагою, існує як завгодно короткий шлях від однієї вершини цього циклу до іншої

(кожен обхід циклу зменшує довжину шляху). Цикл, сума ваг ребер якого негативна, називається негативним циклом.

Вирішимо поставлену задачу на графі, в якому завідомо немає негативних циклів.

Для знаходження найкоротших шляхів від однієї вершини до всіх інших, скористаємося методом динамічного програмування. Побудуємо матрицю A_{ij}, елементи якої будуть позначати наступне: A_{ij} - це довжина найкоротшого шляху з s в i, що містить не більше j ребер.

Шлях, що містить 0 ребер, існує тільки до вершини s. Таким чином, A_{i0} дорівнює 0 при i = s, і $+\infty$ в іншому випадку.

Тепер розглянемо всі шляхи з s в i, що містять рівно j ребер. Кожен такий шлях є шлях з j-1 ребер, до якого додано останнє ребро. Якщо про шляхи довжини j-1 всі дані вже підраховані, то визначити j-й стовпець матриці не складає труднощів.

Алгоритм Беллмана-Форда дозволяє визначити, чи існує в графі G негативний цикл, досяжний з вершини s. Досить призвести зовнішню ітерацію циклу не | V | - 1, а рівно | V | раз. Якщо при виконанні останньої ітерації довжина найкоротшого шляху до якої-небудь вершини строго зменшилася, то в графі є негативний цикл, досяжний з s. На основі цього можна запропонувати такий підхід до оптимізації: відстежувати зміни в графі і, як тільки вони закінчаться, припинити виконання алгоритму.

Алгоритм Дейкстри як складова алгоритму Джонсона

Алгоритм Дейкстри знаходить найкоротші шляхи від однієї з вершин графа до всіх інших. Алгоритм працює тільки для графів без ребер негативного ваги. Алгоритм широко застосовується в програмуванні і технологіях, наприклад, його використовують протоколи маршрутизації OSPF і IS-IS.[4,9,10]

Кожній вершині з V зіставимо мітку - мінімальну відому відстань від цієї вершини до a. Алгоритм працює покроково - на кожному кроці він «відвідує» одну вершину і намагається зменшувати мітки. Робота алгоритму завершується, коли всі вершини відвідані.

Ініціалізація. Мітка самої вершини a покладається рівною 0, мітки інших вершин - нескінченності. Це відображає те, що відстані від a до інших вершин поки невідомі. Всі вершини графа позначаються як невідвідані.

Крок алгоритму. Якщо всі вершини відвідані, алгоритм завершується. В іншому випадку, з ще не відвіданих вершин вибирається вершина u, що має мінімальну мітку. Ми розглядаємо всілякі маршрути, в яких u є передостаннім пунктом. Вершини, в які ведуть ребра з u, назвемо сусідами цієї вершини. Для кожного сусіда вершини u, крім зазначених як відвідані, розглянемо нову довжину шляху, що дорівнює сумі значень поточної мітки u і довжини ребра, що з′єднує u з цим сусідом. Якщо отримане значення довжини менше значення мітки сусіда, замінимо значення мітки отриманим значенням довжини. Розглянувши всіх сусідів, помітимо вершину u як відвідану і повторимо крок алгоритму.

Повернемося до розгляду алгоритму Джонсона.

Дано граф G = (V, E) з ваговою функцією $\omega : E \to R$. Якщо ваги всіх ребер ω в графі невід′ємні, можна знайти найкоротші шляхи між усіма парами вершин, запустивши алгоритм Дейкстри один раз для кожної вершини. Якщо в графі містяться ребра з негативною вагою, але відсутні цикли з негативним вагою, можна обчислити новумножину ребер з невід′ємними вагами, що дозволяє скористатися попереднім методом. Новамножина, що складається з ваг ребер ω', має задовольняти наступним властивостям.

Для всіх ребер (u, v) нова вага $\omega'(u, v) > 0$.

Для всіх пар вершин u, v ϵ V шлях p є найкоротшим шляхом з вершини u у вершину v з використанням вагової функції ω тоді і тільки тоді, коли p - також найкоротший шлях з вершини u у вершину v з ваговою функцією ω'.

Лемма. (Зміна ваг зберігає найкоротші шляхи). Нехай дано зважений орієнтований граф G = (V, E) з ваговою функцією $\omega : E \to R$, і нехай $h : V \to R$ - довільна функція, що відображає вершини на дійсні числа. Для кожного ребра (u, v) ϵ E визначимо

$$\omega'(u, v) = \omega(u, v) + h(u) - h(v).$$

Нехай $p = < v_0, v_1, \ldots, v_k >$ - довільний шлях з вершини v_0 в вершину v_k. p є найкоротшим шляхом з ваговою функцією ω тоді і тільки тоді, коли він є найкоротшим шляхом з ваговою функцією ω', тобто рівність $\omega(p) = \delta(v_0, v_k)$ рівносильно рівності $\omega'(p) = \delta'(v_0, v_k)$. Крім того, граф G містить цикл з негативною вагою з використанням вагової функції ω тоді і тільки тоді, коли він містить цикл з негативною вагою з використанням вагової функції ω'.

Для даного графа створимо новий граф $G' = (V', E')$, де $V' = V \cup \{s\}$, для деякої нової вершини $s \notin V$, а $E' = E \cup \{(s, v): v \in V\}$.

Розширимо вагову функцію ω таким чином, щоб для всіх вершин v ϵ V виконувалося рівність $\omega(s, v) = 0$.

Далі визначимо для всіх вершин v ϵ V′ величину $h(v) = \delta(s, v)$ і нові ваги для всіх ребер $\omega'(u, v) = \omega(u, v) + h(u) - h(v) \geq 0$.

Ребра зберігаються у вигляді списків суміжних вершин. Алгоритм повертає звичайну матрицю $D = d_{ij}$ розміром | V | x | V |, де $d_{ij} = \delta(i, j)$, або видає повідомлення про те, що вхідний граф містить цикл з негативною вагою.

Формалізація алгоритму Джонсона за допомогою САА-М

Використовуючи математичний апарат САА-М розробимо схему для алгоритму Джонсона.

Функція генерування графу:

$$generateGraph(V,from,to) =$$

$$= \underset{i<V}{\{} \; \underset{j<V}{\{} \; (B)^{*} \; (C)^{*} A^{*} j{+}{+}\}^{*} i{+}{+}\}$$

$$adj[i][j]$$

де:
A - adjacency_matrix[i][j] = (int)Math.round(Math.random()*(to - from) + from);
B - adjacency_matrix[i][j] = 0; continue;
C - adjacency_matrix[i][j] = MAX_VALUE;

Функція розрахунку розширеного графу

$$computeAugmentedGraph(adj) =$$

$$= \underset{i<V}{\{} \; \underset{j<V}{\{} \; A^{*} j{+}{+}\}^{*} B^{*} i{+}{+}\}$$

де:
A - augmentedMatrix[i][j] = adjacencyMatrix[i][j];
B - augmentedMatrix[V+1][i] = 0;
augmentedMatrix[i][V+1] = MAX_VALUE;

Алгоритм Беллман-Форда

$$BellmanFord(source,aug) = \underset{i<V}{\{} A^{*} i{+}{+}\}^{*} B^{*}$$

$$^{*} \; \underset{n<V\text{-}1}{\{} \; \underset{i<V}{\{} \; \underset{j<V}{\{} \; ((C))^{*} j{+}{+}\}^{*} i{+}{+}\}^{*}$$

$$^{*} n{+}{+}\}^{*} \; \underset{i<V}{\{} \; \underset{j<V}{\{} \; ((D))^{*} j{+}{+}\}^{*} i{+}{+}\}$$

де:
A - distances[i] = MAX_VALUE;
B - distances[source] = 0;
C - distances[j] = distances[i] + adjacencymatrix[i][j];
D - System.out.println("The Graph contains negative egde cycle"); System.exit(0);
α - adj[i][j]!=MAX
β - dis[j]>dis[i]+adj[i][j]

Функція перезважування

$$reweightGraph(adj) = \underset{i<V}{\{} \; \underset{j<V}{\{} \; A^{*} j{+}{+}\}^{*} i{+}{+}\}$$

де:
A - result[i][j] = adjacencyMatrix[i][j] + potential[i] - potential[j];

Алгоритм Дейкстри

$$Dijkstra(source,adj) = \underset{i<V}{\{} A^{*} i{+}{+}\}^{*} i{+}{+}\}^{*} B^{*} \underset{\alpha}{\{} C \}$$

$$getUnsettled(unsettled) = D^{*} \; \underset{i<V}{\{} \; (count{+}{+})^{*} i{+}{+}\} \; \beta$$

$$getNodeWithMinimumDistatance(uettled) =$$

$$= E^{*} \; \underset{i<V}{\{} \; (F)^{*} i{+}{+}\} \; \gamma$$

$$evaluateNeighbours(node) = \underset{i<V}{\{} \; ((G^{*}(H)))^{*} I^{*} i{+}{+}\} \; \delta \; \varepsilon \; \xi$$

де:
A - distances[i] = MAX_VALUE;
B - unsettled[source] = true; distances[source] = 0;
C - evaluationnode = getNodeWithMinimumDistanceFromUnsettled(unsettled);
unsettled[evaluationnode] = false;
settled[evaluationnode] = true;
evaluateNeighbours(evaluationnode);
D - intcount = 0;
E - intmin = MAX_VALUE; intnode = 0;
F - node = vertex; min = distances[vertex];
G - edgeDistance = adjacencymatrix[evaluationNode][i]; newDistance = distances[evaluationNode] + edgeDistance;
H - distances[i] = newDistance;
I - unsettled[i] = true;
α - getUnsettled(unsettled)!=0;
β - unsettled[i]=true;
γ – unsettled[i]=true^distances[i]<min;
δ – settled[i]=false;
ε – adj[node][i]!=MAX;
ζ – newDistance<distances[i];

Запишемо повний алгоритм Джонсона

$$Johnson = generateGraph(V,from,to)^{*}$$

$$^{*} computeAugmentedGraph(adj)^{*}$$

$$^{*}BellmanFord(source,aug)^{*}reweightGraph(adj)^{*}$$

$$^{*} \underset{i<V}{\{} Dijkstra(i,adj)^{*} i{+}{+}\}$$

Формування паралельної схеми алгоритму Джонсона

Для розпаралелювання алгоритму Джонсона необхідно знайти такі кроки алгоритму, які працюють з різними даними незалежно. Одним з таких кроків є розрахунок за допомогою алгоритму Дейкстри, оскільки він рахується для кожної вершини окремо. Відобразимо це за допомогою САА-М.

$$Johnson = generateGraph(V,from,to)^{*}$$

$$^{*} computeAugmentedGraph(adj)^{*}$$

$$^{*}BellmanFord(source,aug)^{*}reweightGraph(adj)^{*}$$

$$^{*} \overset{V}{\underset{0}{\cdot}} \sum Dijkstra(i,adj)$$

Дослідження отриманих схем

У цій частині наведено експериментальні дані, отримані при тестуванні програмної реалізації алгоритму Джонсона. Тестування полягає в наступному: для кожної розмірності графа N і кількості потоків вимірюється 10 значень часу витраченого на обрахунок. В таблицю заноситься

середнє арифметичне виміряних значень. Для тестування було описано клас Program, який викликає класи і методи алгоритму Джонсона з різними параметрами. Для збереження даних у файл формату .xls використовувалась бібліотека ApachePOI. Тестування проводилось на чотирьох обчислювальних машинах з різною конфігурацією.

Тест №1

Конфігурація обчислювальної машини:

CPU: Intel Core 2 Quad Q6600 2.4GHz, RAM:2Gb, GPU: GeForceGT630, VRAM:2Gb, OS: Windows 8.1x64. Будуємо графік згідно отриманих даних (рис.1). З графіку можна сказати, що порівняно з послідовною версією алгоритму розподілення на 2 потоки дає невеликий приріст в швидкодії. Розподілення на 3,4,6,8 збільшує швидкодію, проте не суттєво.

Тест №2

Конфігурація обчислювальної машини:

CPU: IntelCorei5-3330 3.2GHz, RAM: 8Gb, GPU: GeForceGT210, VRAM:1Gb, OS: Windows10 x64. Будуємо графік згідно отриманих даних (рис.2). З графіку можна сказати, що порівняно з послідовною версією алгоритму розподілення на 2 потоки дає значний приріст в швидкодії. Розподілення на 3,4,6,8 збільшує швидкодію досить суттєво.

Тест №3

Конфігурація обчислювальної машини:

CPU: 2x2,5GHz, RAM: 2Gb, OS: Ubuntu 15.10. Будуємо графік згідно отриманих даних (рис.3). З графіку можна сказати, що порівняно з послідовною версією алгоритму розподілення потоки майже не дає приросту в швидкодії.

Тест №4.

Конфігурація обчислювальної машини:

LG G Pro Lite D 686, CPU: MTK 6577 1GHz x2, RAM: 1 Gb ,OS: Android 4.4. Будуємо графік згідно отриманих даних (рис.4). З графіку можна сказати, що порівняно з послідовною версією алгоритму розподілення потоки майже не дає приросту в швидкодії.

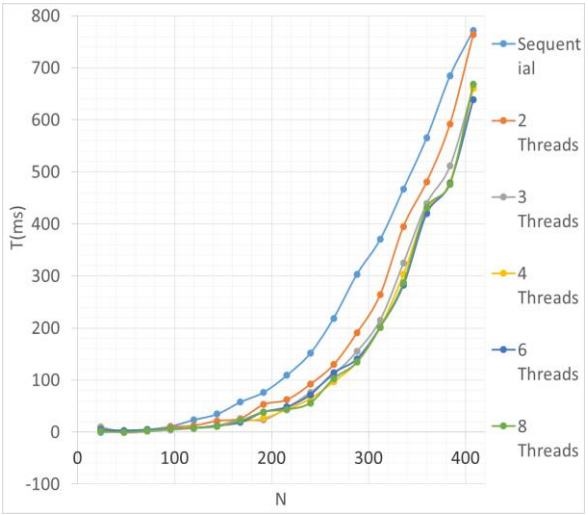

Рисунок 1 - Графік результатів тесту №1.

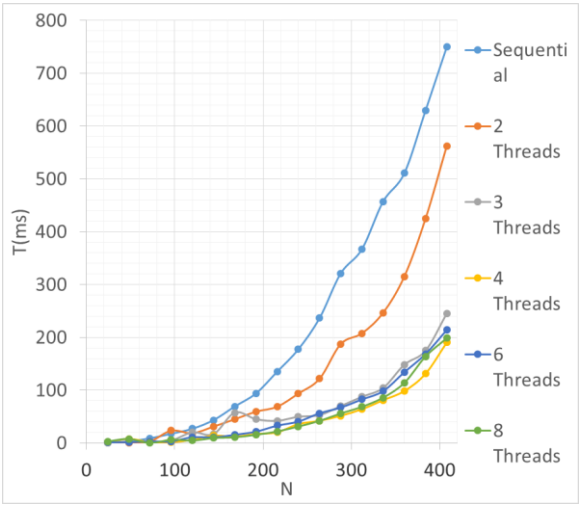

Рисунок 2 - Графік результатів тесту №2.

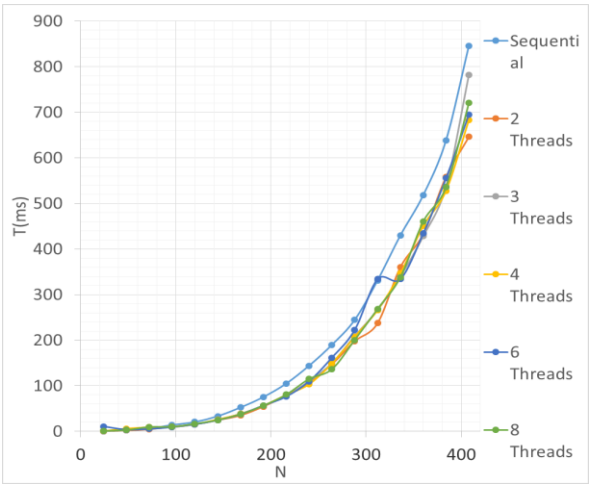

Рисунок 3 - Графік результатів тесту №3.

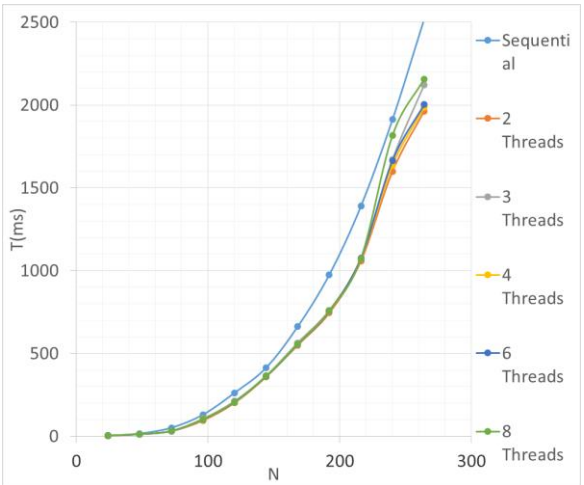

Рисунок 4. Графік результатів тесту №4.

Висновки

•Створено програмний комплекс для дослідження продуктивності обчислювальних систем (платформа Java), а також для визначення приросту продуктивності при паралелізації.

•Як функціональна складова комплексу використовується задача пошуку найкоротших шляхів в графах за допомогою алгоритму Джонсона.

•Алгоритм формалізовано за допомогою засобів модифікованої системи алгоритмічних алгебр Глушкова, що дозволяє абстрагуватись від конкретної платформи і особливостей реалізації. Розроблено паралельну схему алгоритму.

•Проведено тестування створеної схеми на платформах Java з різними конфігураціями. Аналіз результатів тесту показує суттєвий приріст в паралельній роботі. Найкращий результат отримано в тесті №2. Приріст при паралельному виконанні перевищує 4 рази.

Список використаної літератури

1. http://www.ixbt.com/cpu/cluster-benchtheory.shtml
2. http://www.top500.org/project/linpack/
3. http://www.ks.uiuc.edu/Research/namd/performance.html
4. Алгоритмы / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн // Построение и анализ.
5. Goetz Brian. Java Concurrency in Practice. — Addison Wesley, 2006. — ISBN 0-321-34960-1.
6. Lea Doug. Concurrent Programming in Java: Design Principles and Patterns. — Addison Wesley, 1999. — ISBN 0-201-31009-0.
7. Погорілий С.Д. Генетичний алгоритм розв'язання задачі маршрутизації в мережах / С.Д. Погорілий, Р.В. Білоус // Матеріали сьомої міжнародної науково-практичної конференції з програмування УкрПРОГ'2010. Проблеми програмування. – 2010. – №2-3. С. 171-177.
8. L. R. Ford, Jr., D. R. Fulkerson. Flows in Networks, Princeton University Press, 1962.
9. Dijkstra E. W. A note on two problems in connexion with graphs // Numer. Math — Springer Science+Business Media, 1959. — Vol. 1, Iss. 1. — P. 269—271. — ISSN 0029-599X, 0945-3245
10. Левитин А.В. Жадные методы: алгоритм Дейкстры / А.В. Левитин // Алгоритмы. Введение в разработку и анализ (глава 9). — М.: Вильямс, 2006. — С. 189-195. — 576 с.
11. Погорілий С.Д. Програмне конструювання: підручник / С.Д. Погорілий; за ред. О.В. Третяка. – 2-е вид. – К.: Видавничо-поліграфічний центр «Київський університет», 2005. – 483 с.
12. Pogorilyy S.D., Gusarov A.D. Paralleling of Edmonds-Karp Net Flow Algorithm. Applied and Computational Mathematics, vol. 5, 2006, no.2, p.p.121-130.
13. Богачёв К.Ю. Основы параллельного программирования / К.Ю. Богачёв. – М.: БИНОМ. Лаборатория знаний, 2003. – 342 с.
14. Майника Э. Алгоритмы оптимизации на сетях и графах / Э. Майника. – М.: Мир, 1981. – 324 с.

Надійшла до редакції 20.03.2016

С.Д. ПОГОРЕЛЫЙ, Б.В. МИГАШКО
Киевский национальный университет им. Тараса Шевченко

ПОДХОД К ОЦЕНКЕ ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛЬНОЙ РАБОТЫ ПРОГРАММНО-АППАРАТНОЙ ПЛАТФОРМЫ

Создан программный комплекс для исследования программно-аппаратной платформы Java. Предложено использование алгоритма Джонсона для тестирования систем. Выполнена формализация алгоритма средствами модифицированной системы алгоритмических алгебр Глушкова. Разработана параллельная схема алгоритма. Проведены экспериментальные исследования повышения производительности при проведении вычислений на разных платформах.

Ключевые слова: benchmark, Java, алгоритм Джонсона, САА-М, параллелизация.

S.D. POGORILYY, B.V. MYHASHKO
Taras Shevchenko National University of Kyiv

THE APPROACH TO THE ASSESSMENT OF SOFTWARE AND HARDWARE PARALLEL WORK EFFICIENCY

Software package for software and hardware Java platforms research was created. Johnson`s algorithm usage was proposed for testing systems. The formalization of algorithm was done by means of the modified system of Glushkov's algorithmic algebras. A parallel algorithmic scheme was developed. Experimental studies on performance improvement during doing calculations on different platforms were carried out.

Keywords: benchmark, Java, Johnson`s algorithm, SAA-M, parallelization.