Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи)

магістр
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: Дослідження оптимального алгоритму та використання методів розпаралелювання для задач сингулярно-спектрального розкладу

Виконав: студент (ка) 6 курсу, групи СНнм-61
спеціальності (напряму підготовки)
122 «Комп'ютерні науки»
(шифр і назва спеціальності (напряму підготовки))

(підпис) Свистун І.В. (прізвище та ініціали)

Керівник (підпис) Приймак М.В. (прізвище та ініціали)

Нормоконтроль (підпис) Мацюк О.В. (прізвище та ініціали)

Рецензент (підпис) Марценюк В.П. (прізвище та ініціали)

м. Тернопіль – 2020

АНОТАЦІЯ

Дослідження оптимального алгоритму та використання методів розпаралелювання для задач сингулярно-спектрального розкладу // Дипломна робота ОР «Магістр» // Свистун Ігор Володимирович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно–інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНнм–61 // Тернопіль, 2020 // С. – 122, рис. – 11, табл. – 5, додат. – 8, бібліогр. – 61.

Ключові слова: ПАРАЛЕЛІЗМ ЗАДАЧ, СИНГУЛЯРНО-СПЕКТРАЛЬНИЙ АНАЛІЗ, СТРАТЕГІЯ, MRRR, СПЕКТРАЛЬНИЙ РОЗКЛАД МАТРИЦІ, OPENMP, INTEL TBB, QUARK, PLASMA.

Дипломна робота присвячена дослідженню методів реалізації паралельних алгоритмів для задачі сингулярно-спектрального розкладу на базі паралельних бібліотек.

У першому розділі описано метод сингулярно-спектрального аналізу, оглянуто парадигму «розділяй і владарюй», а також алгоритм MRRR для виконання спектрального розкладу матриці. Обґрунтовано вибір паралелізму як методу оптимізації програм. Проаналізовано можливі методи оптимізації програми;

У другому розділі було проаналізовано методи та технології паралельних обчислень, описано різні типи паралелізму і вибрано один із них – паралелізм на основі задач, найновіший. проведено огляд бібліотек для реалізації паралелізму і проаналізовано найпоширеніші бібліотеки паралельного програмування, які підтримують паралелізм на основі задач, було проведено їх якісний аналіз.

У третьому розділі було розглянуто адаптацію методу спектрального розкладу матриці для виконання в паралельному середовищі. Розроблено адаптацію алгоритму сингулярного розкладу матриці для паралельного виконання. Також було модифіковано стратегію паралелізму для алгоритмів «розділяй і владарюй» і MRRR, зроблено їх якісне порівняння на основі раніше проведених експериментів.

У четвертому розділі описано результати проведення обчислювального експерименту з використання паралельних методів. Розраховано сингулярний розклад для матриць на базі як реальних даних газоспоживання, так і випадково згенерованих даних, зроблений порівняльний висновок щодо отриманих даних. Наведено результати використання паралельних методів.

Об'єкт дослідження: паралельна реалізація алгоритму сингулярно-спектрального розкладу.

Предмет дослідження: паралельна модель, алгоритмічні та програмні методи розпаралелювання реалізації сингулярно-спектрального розкладу для аналізу часових рядів.

Мета роботи: вибір оптимального алгоритму та стратегії його розпаралелювання для задачі сингулярно-спектрального розкладу.

Основні результати: на основі аналізу наукових праць в роботі обґрунтовано актуальність і важливість науково-практичного завдання розпаралелення методу сингулярно-спектрального розкладу та його застосування для аналізу часових рядів. Проведено критичний аналіз та вибір ефективної техніки розпаралелювання і запропоновано її програмну імплементацію. Проведено адаптацію паралельної стратегії сингулярно-спектрального розкладу для аналізу часових рядів газоспоживання. Проведено обчислювальний експеримент та порівняння витрат часу та ресурсів пам'яті для випадкових та реальних даних.

ANNOTATION

Study of an optimal algorithm and parallelizing methods use for the problems of singular-spectrum analysis // Diploma work degree «Master» // Svystun Ihor // Ternopil Ivan Pul'uj National Technical University, Department of Computer Information Systems and Software Engineering, Department of Computer Science // Ternopil, 2020 // P. 122, Fig. – 11, Tab. – 5, App. – 8, Ref. – 61.

Thesis is devoted to the researching methods of implementing parallel algorithms for singular-spectral decomposition problem based on parallel libraries

In the work was presented implementation MRRR i Divide and Conquer algorithms. The first one was specifically designed for multi-core and many-core architectures. MR^3-SMP breaks the computation into tasks to be executed by in parallel multiple threads. The tasks are both created and scheduled dynamically. While a static division of work would introduce smaller overheads, the dynamic approach is flexible and produces remarkable work load balancing. For small and medium size matrices, MR^3-SMP matches or outperforms the fastest existing parallel eigensolver.

Also was described a new Divide and Conquer implementation, exploiting parallelism through a task-flow model. According to the experiments we conducted, our implementation outperforms existing Divide and Conquer implementations with the same accuracy. The task-based approach, using the runtime QUARK to schedule tasks, provides good performance, as seen previously with the implementation of the reduction to tridiagonal form. In addition, performance experiments showed that the implementation is competitive with the best MRRR implementation on shared-memory architectures. Considering the assets and the drawbacks of both algorithms, and the fact that the problem is mainly matrixdependent, choosing one eigensolver depends on the application. The extra amount of memory required by Divide and Conquer could be problematic, but the robustness of the algorithm ensures that we

obtain an accurate solution. In addition, our algorithm presents a speedup on matrices extracted from real-life data. Main improvement was to express more parallelism during the merge step where the quadratic operations become costly as long as the cubic operations are well parallelized.

The slower MR3-SMP is, the faster Divide and Conquer becomes, and, conversely, as stated by the theory. For most of the test cases, Divide and Conquer algorithm is faster than MR3-SMP and can be up to 25 faster except for some cases where it can be at max 2 times slower. Considering the improvement with respect to existing Divide and Conquer implementations, those results are interesting, because a larger set of matrices become faster to solve using Divide and Conquer than using MRRR. On an application where performance is the main asset, we can suppose that using Divide and Conquer or MRRR will depend on the matrix used. However, the main asset of Divide and Conquer is the accuracy provided, and it is better than the obtained accuracy with MRRR on both the orthogonality of the eigenvectors and the reduction of the tridiagonal matrix.

The uniqueness of the work is that the experiment was conducted on actual and real gas consumption data and results can speed up the introduction of profit forecasting consumption of natural resources and greatly simplify the process of economy and exploitation.

In the first chapter described singular spectral analysis method, examined the paradigm of Divide and Conquer and MRRR algorithms for performing matrix spectral decomposition. Substantiates the choice of parallelism as a method of optimization programs.

In the second chapter was analyzed the methods and technologies of parallel computing, chosen the latest type of parallelism and analyzed the most common parallel programming libraries that support this type, was conducted qualitative analysis.

In the third chapter examined the adaptation of the singular value decomposition method for the parallel execution environment. Was also developed a

parallel strategy for Divide and Conquer and MRRR algorithms, describe their qualitative comparison based on earlier experiments. «divide and conquer» is efficient when eigenvalues are clustered, whereas MRRR is fast when the eigenvalues are well separated. Divide and Conquer requires larger workspace, but its accuracy is better.

In the fourth described the results of computational experiments using parallel methods. Was calculated singular value decomposition based on real gas consumption data and on randomly generated data, a comparative opinion on the data.

Object of study: the parallel implementation of the singular value decomposition algorithm.

Subject of research: parallel model, algorithmic and program implementation methods to parallelize singular value decomposition for time series analysis.

Purpose: to choose optimal algorithm and its parallelization strategies for singular value decomposition problem.

Main results: based on the analysis of scientific papers in the work substantiated the actuality and importance to parallelize singular value decomposition method and its implementation on time series analysis. Conducted a critical analysis and was chosen an efficient parallelization techniques and was offered its software implementation. Was made an adaptation for parallel strategy of singular value decomposition on time series of gas consumption. Carried out a computational experiment and compared time and memory resources for casual and real data. The experiment was conducted on actual gas consumption and real data the results to speed up the introduction of profit forecasting consumption of natural resources and greatly simplify the process of saving and operation.

Keywords: TASK BASED PARALLELIZM, SSA, SVD, PARALLEL STRATEGY, DC, MRRR, OPENMP, INTEL TBB, QUARK, PLASMA.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

BI (англ. Bisection and Inverse Iteration) – алгоритм бісекції і зворотної ітерації.

DAG (англ. Directed Acyclic Graph) – орієнтований (напрямлений) ациклічний граф.

DC (англ. Divide and Conquer) – алгоритм «розділяй і владарюй».

dqds (англ. differential quotient-difference algorithm with shifts) - алгоритм знаходження сингулярних чисел двохдіагональної матриці

FIFO (англ. First In, First Out) – принцип «перший прийшов, перший пішов».

HT (англ. Hyper-Threading) – технологія «одночасної гіперпотоковості».

ILP (англ. Instruction-level Parallelism) – паралелізм на рівні команд.

JIT (англ. Just in Time) – компіляція «на льоту».

LIFO (англ. Last In, First Out) – принцип «останній прийшов, перший пішов».

MIMD (англ. Multiple Instruction, Multiple Data) – множинний потік команд, множинний потік даних.

MISD (англ. Multiple Instruction, Single Data) – множинний потік команд, одиночний потік даних.

MRRR (Multiple Relatively Robust Representation) – доопрацьована версія алгоритму зворотньої ітерації.

SIMD (англ. Single Instruction, Multiple Data) – одиночний потік команд, множинний потік даних.

SISD (англ. Single Instruction, Single Data) – одиночний потік команд, одиночний потік даних.

SMT (англ. simultaneous multithreading) – технологія «одночасної гіперпотоковості».

SSA (англ. Singular Spectrum Analysis) – аналіз сингулярного спектру.

SVD (англ. Singular Value Decomposition) – сингулярний розклад матриці.

TBB (англ. Threading Building Blocks) – бібліотека паралельного програмування на базі процесорів Intel.

ЕОМ – електронно-обчислювальна машина.

ЗМІСТ

ВСТУП

Актуальність роботи зумовлена використанням методу сингулярно-спектрального розкладу для аналізу часових рядів, що в свою чергу дозволяє виділити із нього корисний сигнал, який має чітке фізичне обґрунтування. Прогноз, який можна отримати в результаті такого аналізу, може значно підвищити ефективність та економію використання ресурсів, що, враховуючи сучасну світову економіку, є надзвичайно актуальним.

Метою роботи є вибір оптимального алгоритму та стратегії його розпаралелювання для задачі сингулярно-спектрального розкладу.

Для досягнення поставленої мети потрібно вирішити наступні *задачі*:

1. На основі аналізу відомих алгоритмів сингулярно-спектрального розкладу обґрунтувати актуальність та можливість його паралельного виконання.

2. Обґрунтувати вибір ефективної техніки розпаралелювання та його програмної імплементації.

3. Вибрати оптимальну стратегію розпаралелювання сингулярно-спектрального розкладу.

4. Провести обчислювальний експеримент з метою визначення затрат часу і ресурсів пам'яті для запропонованої паралельної реалізації.

Об'єктом дослідження є паралельна реалізація алгоритму сингулярно-спектрального розкладу.

Предметом дослідження є паралельна модель, алгоритмічні та програмні методи розпаралелювання реалізації сингулярно-спектрального розкладу для аналізу часових рядів.

Наукова новизна роботи полягає в тому, що вперше проведено адаптацію (конкретних) паралельних методів до аналізу часових рядів газоспоживання. Що дало можливість суттєво зменшити час виконання сингулярного розкладу.

Сингулярний розклад є ортогональним і показує геометричну структуру матриці, дозволяє наочно показати дані, які в ній містяться. Сингулярний метод використовується в найрізноманітніших задачах, таких як наближення методом найменших квадратів, рішення довільних СЛАР, стискання, шифрування і розпізнавання зображень, кластеризація табличних наборів даних, створення цифрових підписів в звукових файлах. В контексті роботи з часовими рядами, сингулярний розклад застосовують до ганкелевих матриць, що дає низку особливих властивостей, а саме: за умови правильного підбору параметра L і подальшого групування власних чисел з'являється можливість успішно розділяти адитивні складові ряду, тобто виділяти із складного часового ряду газоспоживання корисний сигнал(и), що має чітке фізичне обґрунтування.

Сингулярний розклад є основною математичною частиною сингулярно-спектрального аналізу, сучасного методу аналізу та прогнозу часових рядів і дає змогу працювати з широким спектром нестаціонарних випадкових процесів, аналізувати та прогнозувати як трендові, і циклічні компоненти. Можливе використання цього методу в задачах інтерполювання частини втрачених даних статистики. Оскільки сингулярний розклад матриці відбувається за тою ж схемою, що й знаходження власних пар в симетричній матриці, то ми будемо розглядати відомі алгоритми для розв'язку останнього.

Протягом останніх років спостерігалося бурхливе зростання продуктивності мікропроцесорів і обчислювальної техніки на їх основі. Багато завдань, що вимагають рішення, є вельми вимогливими до обчислювальних ресурсів. А це вимагає створення обчислювальних систем, що перевищують найсучасніші мікропроцесори по продуктивності у багато разів. Отже, єдиним виходом є використання багатопроцесорних технологій.

Дане дослідження було проведене проведена на кафедрі комп'ютерних наук ще в 2015 році [55].

1 ДОСЛІДЖЕННЯ СУЧАСНИХ МЕТОДІВ РЕАЛІЗАЦІЇ СИНГУЛЯРНОГО РОЗКЛАДУ МАТРИЦІ

1.1 Опис чисельного методу сингулярно-спектрального аналізу для аналізу часових рядів

Сингулярний розклад матриці – один із найвідоміших матричних розкладів. Розклад або факторизація – мультиплікативне представлення матриці у вигляді кількох матриць (зазвичай двох або трьох), які мають ті чи інакші властивості. Процес факторизації здійснюється на основі різних лінійних перетворень у відповідних просторах над векторами, які ототожнюються з стовбцями або рядками вихідних матриць, а також матриць проміжних етапів в застосовуваних алгоритмах.

Сингулярний розклад є ортогональним і показує геометричну структуру матриці, дозволяє наочно показати дані, які в ній містяться. Сингулярний метод має ряд властивостей, наприклад, властивість показувати ранг матриці, приближати матриці даного рангу, тощо. Завдяки їм цей метод набув широко розповсюдження і використовується в найрізноманітніших задачах, таких як «наближення методом найменших квадратів», рішення довільних СЛАР, стискання, шифрування і розпізнавання зображень, кластеризація табличних наборів даних, створення цифрових підписів в звукових файлах.

Сингулярно-спектральний аналіз – це один з сучасних методів аналізу та прогнозу часових рядів. Можливе використання цього методу в задачах інтерполювання частини втрачених даних статистики.

Однією із найбільших переваг є те, що SSA носить адаптивний характер реалізації процесу газоспоживання і не потребує апріорної інформації про його структуру, а регулюється лише одним параметром L, який підбирається експериментально, опираючись на досвід дослідника.

У багатьох природничих науках склалось уявлення про можливість опису природних процесів за допомогою функції, що складається з декількох доданків (загальна адитивна модель):

$$f(x) = f_T(t) + f_n(t) + f_r(t) + \varepsilon(t), t \in [0, T] \quad (1.1)$$

де повільна нерегулярна складова – тренд, $f_n(t)$ – періодична або сума періодичних складових (сезонні, добові тощо), $f_r(t)$ – швидкі нерегулярні малі варіації, в котрі, як правило, включають все, що не вкладається у формальну модель, інколи включають і випадкові шуми, $\varepsilon(t)$ – випадкова складова, що описується випадковим процесом певного типу.

Базовий метод сингулярно-спектрального аналізу полягає в перетворені часового ряду в багатовимірний шляхом побудови траєкторної матриці за допомогою векторів вкладення. Його часто називають процедурою «нарізки» часового ряду довжиною L, групуванням членів сингулярного розкладу нової матриці, що утворилася за наступним відновленням. Інтерактивність методу, а відповідно і його складність, полягає в правильному виділені (групуванні) адитивних складових вихідного «часового ряду газоспоживання», таких як тренд, періодичні та квазіперіодичні компоненти, а також стохастичний залишок (зміст якого часто розглядають як шум). Відповідно, якісь і точність альтернативних прогнозів залежить від досвіду дослідника (вибору параметра L). Завдяки цьому методу можна отримувати прогнозні значення за рекурентним та векторним алгоритмом (як модифікація рекурентного).

Коротко опишемо процедуру використання базового методу для аналізу «часового ряду газоспоживання» міста на річному інтервалі спостереження.

Крок 1. Процедура переходить в одновимірний часовий ряд газоспоживання в послідовність багатомірних векторів. Нехай L – деяке число в інтервалі від 2 до N/2, де N – довжина «часового ряду газоспоживання» ($N = 8760$ год на річному інтервалі спостереження). Процедура вкладення утворює $K = N - L + 1$ векторів вкладення $X_i = (f_{i-1} \dots f_{i+L-2}), 1 \le i \le K$, що мають розмірність L. Надалі, щоб підкреслити розмірність X_i будемо називати їх векторами L-вкладення.

L-траєкторна матриця (далі просто траєкторна матриця) часового ряду газоспоживання $F - X = [X_1 \dots X_k]$ складається з векторів вкладення у вигляді стовпчиків наступної матриці:

$$X = (x_{ij})_{i,j-1}^{L,K} = \begin{vmatrix} f_0 & f_1 & f_2 & \cdots & f_{K-1} \\ f_1 & f_2 & f_3 & \cdots & f_K \\ f_2 & f_3 & f_4 & \cdots & f_{K1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ f_{L-1} & f_L & f_{L+1} & \cdots & f_{N-1} \end{vmatrix} \quad (1.2)$$

Зауважимо, що $x_{ij} = f_{i+j-2}$ і матриця X має однакові елементи по діагоналях $i + j = \text{const}$. Таким чином, траєкторна матриця є ганкелевою (матриця в якої по всіх діагоналях перпендикулярних головні розміщені однакові елементи).

Крок 2. Результатом цього кроку є сингулярний розклад траєкторної матриці X часового ряду газоспоживання.

Нехай $S = XX^T$. Позначимо $\lambda_1, \dots, \lambda_L$ «власні числа» матриці S ранговані в порядку $\lambda_1 \ge \cdots \lambda_L \ge 0$, а також $U_1, \dots, U_L$ – ортонормовану систему власних векторів матриці S, що відповідають власним числам. Нехай

$d = \max\{i: \lambda_i > 0\}$. Позначимо $V_i = \frac{X^T U_i}{\sqrt{\lambda_i}}, i = 1, \dots, d$, тоді сингулярний розклад матриці може бути записаний

$$X = X_1 + \cdots + X_d \tag{1.3}$$

де $X_i = \sqrt{\lambda_i} U_i V_i^T$.

Кожна з матриць X_i має ранг 1, тому їх можна назвати елементарними матрицями Набір $(\sqrt{\lambda_i} U_i V_i^T)$ будемо називати і-тою власною трійкою сингулярного розкладу (1.3).

Крок 3. На основі розкладу (1.3) процедура групування ділить всю множину індексів $\{1, \dots, d\}$ на m підмножин $\{I_1, \dots, I_m\}$, що не перетинаються. Нехай $I = \{i_1, \dots, i_p\}$. Тоді результуюча матриця X_I, що відповідає групі I, визначається так: $X_I = X_{i1}, \dots, X_{ip}$.

Такі матриці обчислюються для $I_1, \dots, I_m$, тим самим розклад (1.3) може бути записаний в погрупованому вигляді:

$$X = X_{I1} + \cdots + X_{Im} \tag{1.4}$$

Процедура вибору множин $I_1, \dots, I_m$ називається групуванням власних трійок.

Крок 4. На останньому кроці базового алгоритму кожна матриця згрупованого розкладу (1.4) переходить у новий ряд довжиною N. Нехай Y – деяка матриця розмірністю $L \times K$ з елементами y_{ij}, де $1 \le i \le L$, $1 \le j \le K$.

Покладемо $L^* = \min(L,K)$, $K^* = \max(L,K)$ і $N = L + K + 1$. Нехай $y^*_{ij} = y_{ij}$, якщо $L < K$, і $y^*_{ij} = y_{ij}$, в іншому випадку діагональне усереднення переводить матрицю Y в ряд $g_0, \dots, g_{N-1}$ за такою формулою:

$$g_k \begin{cases} \frac{1}{k+1} \sum_{m-1}^{k+1} y^*_{m,k-m+2} \text{ для } 0 < k < L^* - 1 \\ \frac{1}{L^*} \sum_{m-1}^{L'} y^*_{m,k-m+2} \text{ для } L^* - 1 < k < K^* \\ \frac{1}{N-k} \sum_{m-k-K'+2}^{N-N'+1} y^*_{m,k-m+2} \text{ для } K^* < k < N \end{cases} \quad (1.5)$$

Рівняння (1.5) відповідає усередненню елементів матриці вздовж діагоналей $i + j = k + 2$. Для $k = 0$ отримуємо $g_0 = y_{11}$, для $k = 1$ $g_1 = (y_{12} + y_{21})/2$ тощо.

Зауважимо, якщо матриця Y є траєкторною матрицею деякого ряду $h_0, \dots, h_{N-1}$ (тобто є генкелевою), тоді $g_i = h_i$ для всіх i.

Застосовуючи діагональне усереднення (1.5) до результуючих матриць X_k, отримуємо ряди $F_0^{(k)} = (\tilde{f}_0^{(k)}, \dots, \tilde{f}_{N-1}^{(k)})$ і, відповідно, вихідний ряд $(f_0, \dots, f_{N-1})$ можна розкласти на суму рядів:

$$f_n = \sum_{k-1}^{m} \tilde{f}_n^{(k)} \quad (1.6)$$

Підсумуємо, сингулярний розклад є основною математичною частиною даного методу. Його застосовують до ганкелевих матриць, що дає низку особливих властивостей, а саме: за умови правильного підбору параметра L і подальшого групування власних чисел з’являється можливість успішно розділяти адитивні складові ряду, тобто виділяти із складного

часового ряду газоспоживання корисний сигнал(и), що має чітке фізичне обґрунтування.

Оскільки сингулярний розклад матриці відбувається за тою ж схемою, що й знаходження власних пар в симетричній матриці, то ми будемо розглядати відомі алгоритми для розв’язку останнього.

Для вирішення задачі знаходження власних значень можна застосувати наступні чотири алгоритми: алгоритм бісекції і зворотної ітерації [1], QR [2, 3], і алгоритму (DC) [4, 5], алгоритм MRRR [6].

Всі ці алгоритми представлені в [7], але порівняння продуктивності [8] показали, що DC і MRRR найшвидші з існуючих. Також, якісний аналіз дозволяє зазначити наступне. Як і алгоритм MRRR, метод зворотних ітерацій дозволяє розраховувати підмножини власних пар за зниженою обчислювальної вартістю, вимагаючи $O(nk^2)$ арифметичних операцій в найгіршому випадку. На противагу цьому, інші методи можуть тільки обчислити всі власні пари і їх обчислювальна складність в найгіршому випадку складатиме $O(n^3)$ [8]. Для всіх цих алгоритмів існують реалізації як для послідовної так і для розподіленої пам'яті. Всебічне обговорення паралельних існуючих алгоритмів і їх продуктивності можна знайти в [10]. З погляду точності, QR і DC, як правило, кращі ніж BI і MRRR; DC вимагає $O(n^2)$ додаткової пам'яті а, отже, набагато більше, ніж решта алгоритмів, які вимагають тільки $O(n)$ додаткового місця для зберігання; DC і MRRR набагато швидші, ніж QR і BI; незважаючи на те, що MRRR використовує найменшу кількість операцій з плаваючою комою, DC може бути швидшим на деяких класах матриці. Тобто те, який алгоритм, DC чи MRRR, буде швидшим, залежатиме від спектрального розподілу вхідної матриці.

1.2 Огляд алгоритму MRRR та його дерево представлення

Алгоритм MRRR це доопрацьована версія алгоритму зворотньої ітерації, що усуває необхідність процесу Грама-Шмідта для отримання чисельних ортогональних власних векторів. MRRR, розроблений Dhillon [13], також був вивчений для багатоядерних архітектур. Метою MRRR є знайти відповідний стандартний симетричний невизначений розклад LDL^T для трьохдіагональної матриці, такий, що невелика зміну L викликає невелику зміна в D. Таке представлення дозволяє обчислювати власні вектори з відносно високою точністю. З іншого боку, вона вимагає добре розділених власних значень. Спершу представлення розділяє власні значення спектра на підмножини близьких власних значень. Потім нове представлення $L'D'L'^T = LDL^T - \sigma Id$ обчислюється з σ, яка вибирається для того, щоб розірвати існуючі групи. Цей процес повторюється до тих пір, поки кожна група не буде містити тільки одне власне значення. Потім власні вектори можуть бути обчислені. Як наслідок, k власних пар точно обчислюються за $O(nk)$ операцій. Теоретичні аспекти алгоритму і межі його точності можна знайти в [14, 15] і їх посиланнях. Тут ми зосередимося на аспектах алгоритму, які важливі для розробки стратегії паралельних обчислень. Зокрема, ми орієнтуємося на так званому дереві представлення [14], і різних обчислювальних завданнях, що зустрічаються в алгоритмі. Приблизне дерево представлення показане на рисунку 1.1.

Ми припускаємо, що трьохдіагональна вхідна матриця T є нескороченою, тобто кожен не діагональний елемент є більшим або рівним за деякий поріг. Якщо T не є нескороченою, то завдання може бути розбитим на безліч дрібних нескорочених завдань.

Алгоритм починається з обчислення розкладу T, який визначає всі або набір власних векторів з високою відносною точністю. Таке розкладання називається Relatively Robust Representation (RRR) [16]. Кандидатами для RRR є двохдіагональний розклад виду $LDL^T - \sigma Id$, де $\sigma \in \mathbb{R}$ називається зсувом, і матриці L і D є нижніми блоками двохдіагональної і діагональної відповідно. Тому RRR вказаний по $2n-1$ нетривіальних входах D і L. У прикладі на рисунку 1.1, «кореневий вузол відповідає початковій RRR» $L_0 D_0 L_0^T = T - \sigma_0 I$. Це представлення визначає потрібні власні значення λ_i, де $i \in \Gamma = \{1,2,\dots,9\}$, з високою відносною точністю; цей RRR часто називають кореневим представленням.

Враховуючи RRR, можна отримати апроксимації для власних λ_i, такі що $|\lambda_i - \widehat{\lambda_i}| = O(\varepsilon \widehat{\lambda_i})$, де ε позначає машинну точність. Це може бути досягнуто за допомогою бісекції за $O(n)$ арифметичних операцій на власне число [17]. Якщо RRR є визначеним, всі «власні числа» можуть бути обчислені за швидким алгоритмом також за $O(n^2)$ операцій [18]. Після того, як апроксимація власних чисел є доступна, алгоритм MRRR переходить до обчислення відповідних власних векторів. Для кожного власного числа λ_i два випадки повинні бути виділені, залежно від того, чи вони розділені чи згруповані відносно своїх сусідів.

λ_i розділені, що означає, що їх відносний розрив reglap більший або рівний, заданого параметра tol, де

$$\mathrm{reglap}(\widehat{\lambda_i}) := \min_{j \neq i} \frac{|\widehat{\lambda_i} - \widehat{\lambda_j}|}{|\widehat{\lambda_i}|} = \frac{\mathrm{gap}(\widehat{\lambda_i})}{|\widehat{\lambda_i}|} \tag{1.6}$$

В цьому випадку λ_i також називають сингелтоном.

λ_i є частиною кластера, тобто $\mathrm{reglap}(\widehat{\lambda_i}) < \mathrm{tol}$. Ця умова керує групами розміром два і більше. Дерево представлення на рис. 1.1 показує приклади одинаків і кластерів. Діти кореневого вузла з індексами 1 та 5 добре розділені, в той час як «власні числа» з індексами 2, 3 і 4,і з індексами від 6 до 9 утворюють два скупчення, розміром три і чотири, відповідно.

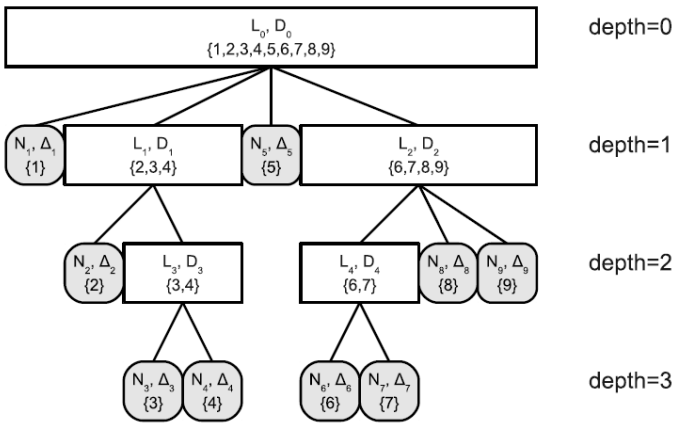


Рисунок 1.1 – Розвиток алгоритму MRRR у вигляді дерева RRR

Листки відповідають добре розділеним власним числам, у той час як внутрішні вузли представляють згруповані.

Різниця між розділеними і згрупованими власними числами відповідає різним шляхам обчислення. У той час як для сингелтонів асоційовані «власні числа» можуть бути обчислені безпосередньо, згрупування вимагають додаткової обробки, перш ніж обчислення власного числа може буде розпочате.

Для кожного скупчення, алгоритм MRRR обчислює новий RRR $L_c D_c L_c^T = LDL^T - \sigma_c I$, і уточнює «власні числа» в групі відносно до цього нового представлення. Оскільки відносний розрив reglap незмінний відносно зсуву, то зсув σ_c можуть бути обраним, щоб гарантувати, що принаймні одне

уточнене власне число стає добре розділеним. Розрахунок нового RRR здійснюється за диференціальною формою стаціонарного перетворення, це швидкий, але, за своєю суттю, послідовний алгоритм зі складністю $O(n)$. Уточнення власних чисел групи виконується бісекцією, і не вимагає $O(n)$ флоп на власне значення. Для згрупованих власних чисел процес повторюється доки всі кластери не розкладаються на одинаків.

У прикладі на рис. 1.1, «власні числа» $\widehat{\lambda_2}, \widehat{\lambda_3}$ і $\widehat{\lambda_4}$ кореневого представлення згруповані, як зазначено в полі {2, 3, 4} на глибині один дерева представлення. Як наслідок, новий RRR $L_1 D_1 L_1^T$ знайдений і три «власні числа» уточняються; в цьому новому RRR, $\widehat{\lambda_2}$ стає тепер сингелтоном як листок {2} на глибині два, тоді як $\widehat{\lambda_3}$ і $\widehat{\lambda_4}$ досі згруповані. Цей процес повторюється, і новий RRR $L_3 D_3 L_3^T$ обчислюється, в якому $\widehat{\lambda_3}$ і $\widehat{\lambda_4}$ тепер стали сингелтонами.

Для кожного сингелтона асоційований власний вектор $\widehat{z_i}$ безпосередньо обчислюється з RRR, розв'язуючи $(LDL^T - \widehat{\lambda_i} I)\widehat{z_i} = \gamma_r e_r$, де права частина системи є r-й вектор стандартного базису, масштабованого з γ_r. Рішення отримано за допомогою факторизації $N\Delta N^T = LDL^T - \lambda_i I$ за $O(n)$ флоп. Детальнішу інформацію про процедуру можна знайти в [15].

Лістинг 1.1 – Алгоритм MRRR

```
Вхід: симетрична трьохдіагональна матриця T_i з набором
індексів Γ бажаних власних пар.
Вихід: власні пари (λ̂_i, ẑ_i), де i ∈ Γ.
1.    Сформувати чергу задач Q
2.    Обчислити RRR_0 і RRR для всіх λ_i, де i ∈ Γ
```

```
3.    Обчислити «власні числа» λ̂_i, де i ∈ Γ
4.  Розділити Γ в наборі Γ = U_k Γ̂_k згідно з їх відносним
    розривом і поставити кожен елемент в чергу (Γ_k, RRR_0)
5.    while Q не пуста do
6.      Вийняти із черги нову задачу (Γ̂, RRR)
7.      If |Γ| > 1 then
8.          Обчислити RRR_new, всі RRR для λ̂_i з індексами i ∈ Γ
9.          Уточнити λ̂_i з індексами i ∈ Γ відповідно до RRR_new
10.         Розділити Γ̂ на підмножини Γ̂ = U_k Γ̂_k відповідно до їх
            відносного розриву і поставити кожен елемент в
            чергу (Γ̂_i, RRR_new)
11.     Else
12.         Обчислити λ̂_i, де i ∈ Γ з високою відносною точністю
            і асоційований з ним вектор ẑ_i
13.     End if
14.   End while
```

Цей узагальнений опис MRRR підсумований в лістингу 1.1. Кількість фактичних обчислень в значній мірі залежить від спектрального розподілу матриці, допуск, який обраний для класифікації сингелтонів, який зсув для обчислення нового RRR був вибраний.

1.3 Методи оптимізації програми

PGO (Profile-guided optimization) – це компілятори, які керуються технікою компіляції перед виконання [30]. Цей метод оптимізації компіляції базується на профілях середовища виконання, і схожий на статичний метод «середнього випадку», аналог динамічної методики адаптивної оптимізації.

Код, який сам модифікується, може змінити себе у відповідь на умови середовища виконання для того, щоб оптимізувати код; такий компілятор набув найбільшого поширення в програмах на асемблері [31].

Деякі структури CPU можуть виконувати деякі оптимізації під час виконання. Компілятори можуть допомогти програмі скористатися додатковими функціями процесора, наприклад, через планування команд.

Оптимізація коду також може бути широко класифікована на залежні і незалежні від платформи методи. У той час, як останній з них є ефективним для всіх або більшості платформ, методи залежні від платформи використовують специфічні властивості однієї платформи або покладаються на параметри, які залежать від однієї платформи або, навіть, від одного процесора. Тут може бути необхідне написання або виконання різних версій одного і того ж коду для різних процесорів. Наприклад, у разі оптимізації рівня компіляції, незалежні від платформи методи є універсальними методами, які впливають на більшість процесорних архітектур аналогічним чином. Як правило, вони служать для зменшення загальної довжини інструкції, необхідної для завершення програми та / або зменшують загальне використання пам'яті під час процесу. З іншого боку, методи, залежні від платформи, включають планування команд, паралелізм на рівні інструкцій та даних, методи оптимізації кешу і оптимальне планування інструкція, що може відрізнятися навіть на різних процесорах однієї архітектури.

1.4 Висновки до першого розділу

Отже, сингулярний розклад матриці – один із найвідоміших матричних розкладів. Розклад або факторизація – мультиплікативне представлення матриці у вигляді кількох матриць (зазвичай двох або трьох), які мають ті чи інакші властивості. Процес факторизації здійснюється на основі різних лінійних перетворень у відповідних просторах над векторами, які ототожнюються з стовбцями або рядками вихідних матриць, а також матриць проміжних етапів в застосовуваних алгоритмах. Тобто, принцип сингулярного розкладу матриці схожий з знаходженням власних чисел.

Для вирішення задачі знаходження власних значень можна застосувати алгоритми: BI, QR, DC, MRRR.

Алгоритм MRRR це доопрацьована версія алгоритму зворотньої ітерації, що усуває необхідність процесу Грама-Шмідта для отримання чисельних ортогональних власних векторів.

Порівняння їх продуктивності показали, що DC і MRRR найшвидші з існуючих. Також, те, який алгоритм, DC чи MRRR, буде швидшим, залежатиме від вхідної матриці.

Під час реалізації цих алгоритмів доцільно використовувати ряд оптимізації, відштовхуючись від особливостей програми і машини, на якій вона буде виконуватись. Алгоритми, вибрані для розв’язку задачі, добре піддаються розпаралелюванню, вимагаючи не високих затрат часу для їх перебудови, таким чином, було використано оптимізацію на рівні алгоритму. Також, сам сингулярний розклад матриці потребує певної перебудови на рівні дизайну. Отже, було застосовано оптимізації на рівні дизайну і алгоритмів. Оскільки, для вирішення завдання буде використовуватись багатоядерний комп’ютер, то є сенс використання паралельних технік, а саме паралелізму на рівні задач. Такий підхід дозволяє динамічно керувати виконанням, тим самим використовуючи оптимізацію на рівні середовища виконання.

2 МЕТОДИ ТА ТЕХНОЛОГІЇ ПАРАЛЕЛЬНИХ ОБЧИСЛЕНЬ

2.1 Типи паралелізму

Паралелізм можна забезпечити на декількох рівнях: бітовий, інструкцій, даних, задач.

Бітовий рівень паралелізму – форма паралельних обчислень на основі збільшення розміру машинного слова. Ця тенденція в цілому закінчилася з введенням 32-розрядних процесорів, яка була стандартом обчислень загального призначення протягом двох десятиліть. Тільки недавно, з появою x86-64 архітектур, 64-розрядні процесори стали звичайним явищем.

Паралелізм даних підкреслює розподілений характер даних.

Паралелізм на рівні задач (також відомий як функціональний паралелізм) є однією з форм розпаралелення комп'ютерного коду декількома процесорами в паралельному обчислювальному середовищі. Паралелізм завдань зосереджений на розподіленні конкретних завданнях для виконання процесором або потоком на різних паралельних обчислювальних вузлах. Це контрастує з паралелізмом даних.

Програмування на основі задач покликане спростити написання паралельного коду. Це техніка програмування, яка була прийнята великими постачальниками програмного забезпечення в якості основного підходу до багатоядерного програмування. Відомі комерційні продукти: Intel TBB, Intel Cilk++, OpenMP, Microsoft Task Parallel Library і Java Fork / Join поряд з дослідницькими проектами, такими як TaskMan, Wool і Nanos Mercurium

Програмування на основі задач пропонує альтернативу традиційному програмуванні на основі потоків виконання Паралелізм задач здійснюється за рахунок поділу додатка на невеликі і незалежні виконувані сутності (так званий дрібнозернистий паралелізм), тоді як традиційно управління відбувається явними потоками для обробки блоків задачі. Останній носить

назву крупнозернистий паралелізм. Середовище виконання програм на основі задач обробляє всі синхронізації і завдання планування неявно, тим самим дозволяючи програмісту більше зосередитись на алгоритмічній конструкції додатка. Код таких програм зазвичай більш простий, портативний і ефективний у порівнянні з кодом, в якому керують потоками. Було встановлено, що створення і знищення задач виконується у 18 разів швидше в порівнянні з створенням і знищенням потоків на системах Linux і в 100 разів швидше, ніж на системах Windows.

Ядром двигуна моделі на базі завдань є планувальник завдань, який використовує механізм «крадіжки» завдання, щоб збалансувати паралельне навантаження на наявні ядра процесора з метою підвищення використання основної і загальної продуктивності і масштабованості системи. Після фази ініціалізації, планувальник завдань розкидає навантаження на завдання і зберігає їх у розподілених чергах, як правило, одну чергу на ядро / потік. Потім планувальник призначає потік з пулу потоків для однієї з черг, в якій завдання очікують на обробку. Коли потік виконує всі завдання зі своєї черги, він «краде» роботу з черги іншого потоку. Потік-донор вибирається випадковим чином, а крадене завдання є останнім у черзі.

2.2 Огляд кросплатформенної бібліотеки шаблонів TBB

«Intel Threading Building Blocks» – C++ бібліотека шаблонів, яка підтримує дані паралельного програмування для розробки паралельних додатків, що працюють на базі багатоядерних процесорів. Творці TBB прагнуть зробити її незалежною він процесора, компілятора і операційної системи. Бібліотека складається з будівельних блоків «структур даних і алгоритмів», які звільняють програміста від деяких ускладнень, пов'язаних з використанням базових механізмів розпаралелювання, наприклад, створення потоків, синхронізації, і знищення потоків [35].

TBB дозволяє програмісту створити додаток в термінах об'єктів завдання. Паралелізм виражений явно за допомогою конструкцій TBB, а вкладений паралелізм допускається. Тим не менш, програміст відповідає за створення незалежних і поточно-безпечних завдань. TBB планувальник завдань відображає певні логічні завдання, дані користувачем на фізичні потоки в порядку один до одного, тобто, один програмний потік на один апаратний потік.

Планувальник завдань використовує ефективний і динамічний механізм балансування навантаження на основі механізму «крадіжки» завдання для поліпшення продуктивності пропускної здатності. З цією метою кожен потік підтримує чергу завдань. Потік виконує пошук в глибину при використанні його локальної черги як стеку і, таким чином, досягає низького втручання в простір і хорошого розташування даних. Якщо потік виконав свою роботу, допускається «вкрасти» задачу, як правило, великий шматок з хвоста чужої черги.

Бібліотека TBB складається з паралельних алгоритмів. Конструкція алгоритмів запрограмована мовою C++ в узагальнюючій формі і реалізована поверх планувальника завдань з рекурсивним поділом діапазонів. Бібліотека була розроблена для простоти, тому що паралелізм використовує основні ресурси машини без втручання програміста.

Крім того, бібліотека TBB забезпечує паралельні контейнери [36] (паралельна черга, паралельний вектор, і паралельна хеш-карта). Контейнери є потокобезпечними і для підвищення ефективності забезпечують блокування дрібнозернистих завдань. Бібліотека також містить паралельний розподіл пам'яті, різні механізми взаємного виключення і атомарні операції.

2.3 Модель програмування OpenMP

OpenMP є інструментом для написання багатопоточних додатків в «середовищі із загальною пам'яттю». Він складається з набору директив компілятора і бібліотеки препроцесора. Компілятор генерує багатопоточний код, заснований на вказаних директивах.

OpenMP спрощує задачу створення багатопоточної програми, зберігаючи зовнішній вигляд і відчуття послідовного програмування, що дозволяє навіть новачкам поступово рухатися від послідовного стилю програмування до паралельних програм. OpenMP розширює послідовний код, використовуючи директиви компілятора. Програмісту, знайомому з мовою (наприклад, C / C++) необхідно вивчити тільки невеликий набір директив. Додавання їх не змінить логічну поведінку послідовного коду; вони вкажуть компілятору, яка частина коду повинна бути розпаралелена і як це зробити: компілятор обробляє всю багатопотокову задачу.

Програма на OpenMP починається з одного потоку виконання, який називається потоком-майстром. Майстер потоків породжує команду потоків у відповідь на директиву OpenMP, які виконують роботу паралельно. Паралелізм таким чином додається поступово: послідовна програма перетворюється в паралельну. Директиви OpenMP вставляються в ключових місцях у вихідному коді. Ці директиви приймають форму коментарів у мові FORTRAN і директив в «C і C++». «Компілятор інтерпретує директиви» і створює необхідний код для розпаралелювання зазначеного завдання / регіону [37]. Паралельна область є основною конструкцією, яка створює групу потоків і ініціює паралельне виконання.

OpenMP надає кілька конструкцій для розподіленої роботи між потоками. Такі конструкції повинні бути розміщені усередині існуючої паралельної області. Щоб в результаті розподілити виконання пов'язаних з ними завдань серед існуючих потоків. Крім того, OpenMP надає ряд

конструкцій для синхронізації потоків і їх координації. Цього достатньо для багатьох потреб, але OpenMP також забезпечує набір функцій, які дозволяють блокувати потоки під час виконання. Вони використовуються для точного управління [38].

2.4 Бібліотека для роботи з системами зі спільною пам’яттю QUARK

QUARK – це середовище виконання для динамічного планування і виконання додатків, які складаються з обмежених пріоритетом ядер на багатоядерних, мультисокетних системах з спільною пам'яттю. Мета проекту полягає у забезпечення простого у використанні інтерфейсу прикладного програмування, з ефективною реалізацією, яка буде масштабуватися для великих платформ зі спільною пам'яттю. Основний принцип побудови взяв за основу модель потоку даних, де планування базується на залежностях даних між завданнями в графі завдань. Залежності даних виводяться під час аналізу неявних конфліктів даних, до яких звертаються ядра на стадії виконання.

Центральна увага під час розробки зосереджується на підтримці динамічних алгоритмів лінійної алгебри для проекту лінійного алгебри PLASMA [39]. Однак, QUARK здатний підтримувати інші додатки, які можуть бути розкладені на завдання з залежними даними.

Даючи на вхід послідовний код, що складається з викликів до ядра підпрограми, що працюють зі спільними даними, QUARK може бути використаний, щоб дозволити ядрам підпрограми виконуватись асинхронно і паралельно на архітектурі із спільною пам'яттю. Послідовний характер оригінального коду зберігається шляхом застосування залежностей даних між ядрами. Для того, щоб підтримати це, звертання до ядра повинні бути доповнені, щоб виразити обсяги даних і використання I / O. Ці залежності

формують неявний DAG, що з'єднує ядра підпрограми. На рисунку 2.1 показано ідеалізовану архітектуру середовища виконання QUARK.

Тут терміни функція, ядро і завдання можуть бути взаємозамінні для позначення функції, яка повинна бути виконана середовищем виконання QUARK. Крім того, в залежності від контексту, терміни аргумент, параметр або елемент даних можуть бути використані для позначення параметрів функції.

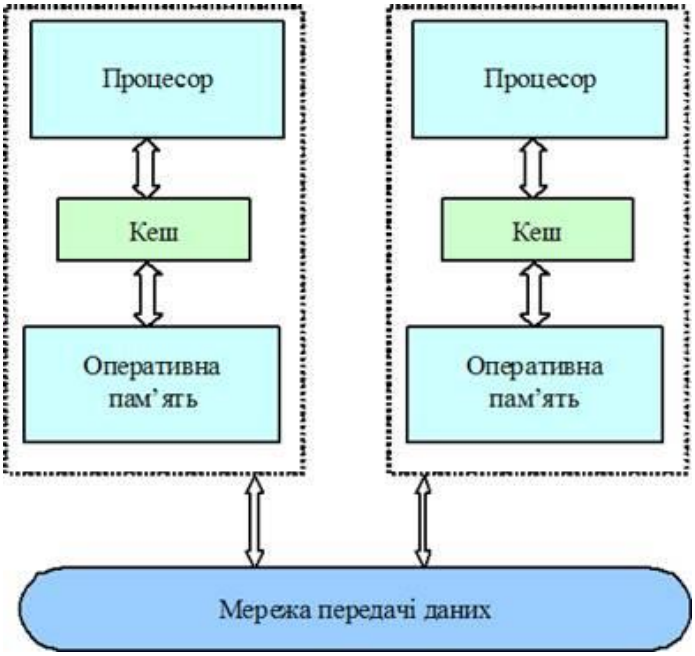


Рисунок 2.1 – Ідеалізована схема архітектури середовища виконання з спільною пам'яттю QUARK

Користувацький потік запускає послідовний код, виступаючи в якості майстер-потоку, який вставляє завдання в неявний орієнтований ациклічний граф на основі їх залежностей. Завдання можуть знаходитись в декількох станах: не готове, на черзі або виконане. Коли залежності задоволені, то завдання поставлені в чергу і виконані робочими потоками. Робочі потоки оновлюють залежності, коли завдання виконуються.

Так як розробка QUARK виходила з потреб бібліотеки лінійної алгебри PLASMA, то були поставлені дуже високі цілі щодо продуктивності.

Значна частина бібліотеки PLASMA може бути виконана з використанням статичного планувальника, а також динамічного середовища виконання QUARK. Також продуктивність QUARK повинна бути конкурентна з статичним планувальником. Враховуючи, що статичний планувальник не має додаткових витрат, QUARK не може перевищувати його продуктивність для основних алгоритмів. Тим не менше, використання QUARK дозволяє дещо оптимізувати (наприклад, злиття DAG, перебудову циклів), що не може бути зроблено за допомогою статичного планувальника. Крім того, підготовка статичного планувальника виконання складних алгоритмів є дуже складною. Програмувати за допомогою API QUARK набагато легше для більшості розробників і це дозволяє їм експериментувати з альтернативними конструкціями алгоритмів.

Широке використання багатоядерних процесорів в сучасному обладнанні привела до появи безлічі багатопоточних фрейморків, які охоплюють ідею планування завдань: Cilk [40], OpenMP [41], «Intel Threading Building Blocks» [42]. Одна особливо актуальна категорія це багатопоточні системи на основі принципу потоку даних, які представляють обчислення у вигляді DAG і планувальник виконання завдань під час виконання за допомогою вирішення конфліктів даних. Проект QUARK потрапляє в цю категорію. Два аналогічні академічні проекти це SMPSs [43] і StarPU [44]. У той час, коли ці три системи мають свої сильні і слабкі сторони, QUARK має важливі доповнення для використання в чисельній бібліотеці.

QUARK часто використовується в якості бібліотеки для послідовних додатків, де QUARK керує всіма потоками і ресурсами. Існує альтернативний режим, в якому QUARK не керує внутрішніми потоками, так що його можна використовувати як частину багатопоточної бібліотеки.

QUARK пропонує ряд функцій, які дозволяють більш тонкий контроль над системою виконання і планувальником завдань. Більшість з цих можливостей контролюються, або через прапори завдань, які передаються в

кожне завдання, або за допомогою аргументу прапорів, що надаються різними аргументами. Кілька глобальних функцій мають можливість використовувати змінні з середовища виконання.

Для того, щоб зрозуміти сенс деяких функцій, потрібно викласти стандартний метод планування завдань. На дуже високому рівні, QUARK використовує інформацію про місцезнаходження даних, щоб призначити завдання, чиї залежності задовольняються потоками, які можуть повторно використовувати ці дані. Потоки виконання назначають завдання, використовуючи принцип черги FIFO. Потік, який не має ніяких завдань, може «вкрасти» завдання із задньої частини черги іншого потоку, використовуючи політику крадіжки LIFO.

QUARK намагатиметься використовувати інформацію про знаходження даних для прийняття рішення про те, де планувати завдання. У поточній версії QUARK, аргументи будуть розглянуті, щоб визначити вихідні дані. Завдання будуть призначені за замовчуванням тому ж потоку, що раніше записав ці вихідні дані. Ця евристика повинна дозволити повторне використання кешу в багатьох практичних випадках.

Коли завдання вставляється в середовище виконання QUARK, його параметри визначаються як триплети. Розмір параметра в байтах, вказівник на параметр (навіть скалярні значення мають бути передані за посиланням), і деякі прапори, які використовуються, щоб вказати, як параметр буде використовуватися завданням і як залежності від цього параметра повинні бути вирішені. Параметр повинен завжди вказувати його використання в якості одного з наступних статусів: VALUE, INPUT, OUTPUT, INOUT, NODEP і SCRATCH.

VALUE. Цей параметр копіюється в завдання QUARK і не використовується для вирішення залежностей.

INPUT параметр використовується в якості вхідних даних тільки для функції; попередня операція запису цих даних повинна бути виконана перш, ніж ця функція може бути виконана.

INOUT параметр використовується в якості вхідних даних, а також вихідних для функції. Попередні операції читання і запису повинні бути виконані, перш ніж ця функція може бути виконана.

Параметр OUTPUT використовується в якості виходу. Попередні операції читання і запису повинні бути виконані перш, ніж ця функція може бути виконана.

Параметр NODEP оголошується програмістом, щоб не викликати яку-небудь залежність. Це дозволяє певну гнучкість для прискорення планування, однак слід використовувати його обережністю. Іноді він використовується, якщо програміст знає, що достатньо залежностей зберігається іншими параметрами.

Параметр SCRATCH оголошений як тимчасовий робочий простір і якщо вказівник на параметр NULL, QUARK виділить дані при необхідності і передасть їх функції.

Крім того, інформація може бути передана в допомогою прапорів параметра. Опишемо деякі з них.

Прапор GATHERV оголошує середовищу виконання QUARK, що дані будуть зібрані в параметрі даних в не конфліктному порядку. Ряд послідовних прапорів GATHERV, які отримують доступ до параметру даних, можуть безпечно виконуватись одночасно.

QUARK був розроблений, щоб співпрацювати з іншими багатопоточними бібліотеками, так що він може використовувати обчислювальні потоки, породжені поза ним. Це дозволяє бібліотеці PLASMA легко перемикатися між внутрішнім статичним планувальником та динамічним середовищем виконання, яке пропонує QUARK. Якщо розробник хоче керувати потоками за межами QUARK, майстер-потік

повинен викликати функцію QUARK_Setup для встановлення структур даних QUARK. Потім кожен робочий потік, який розробник створив зовні, повинен викликати функцію QUARK_Worker_Loop, після чого робочий потік чекатиме завдання. Головний потік додає завдання в звичайному порядку. Коли основний потік виконує додавання завдань, він викликає функцію QUARK_Waitall. Робочі потоки будуть закінчувати поточні завдання і повернуть керування програмою. Коли майстер-потік закінчить, він може викликати функцію QUARK_Free, щоб звільнити всі структури, які були виділені функцією QUARK_Setup.

QUARK забезпечує вторинний список стилю API для передачі аргументів в функцію. Це корисно в ситуації, коли є дуже велика кількість залежностей для функції. Наприклад, функція, яка буде виконана на GPU може приймати велику кількість елементів даних, і було б простіше, щоб дані передавалися за допомогою циклу за індексами [45]. Друга ситуація, коли список стилю API корисний, коли фактична кількість залежностей функції не відома до моменту виконання, так що не можливо використати стандартний метод Varargs, який базується на функції додавання аргументів QUARK_Insert_Task.

2.5 Порівняння Intel TBB, OpenMP та QUARK

OpenMP, Intel TBB і QUARK – паралельні бібліотеки програмування, які підходять для багатоядерних процесорів. У них є багато спільного, але вони були розроблені для різних моделей паралельного виконання. Технології використовують роздільну пам’ять і засновані на багатопоточному програмуванні для максимального використання багатоядерних процесорів. Крім того, модель ядра виконання TBB і QUARK базується на програмуванні завдань, тоді як OpenMP ґрунтується на підході

потокового програмування, підтримуючи паралелізм на основі завдань з версії 3.0 [38].

OpenMP – найпопулярніше паралельне розширення на сьогоднішній день. OpenMP складається з директив, процедур і змінних середовища для програм на мові Fortran і C. OpenMP дозволяє користувачам створювати паралельну програму і допомагає компілятору генерувати програму, що відповідає потребам програміста. Ці директиви є важливими досягненням, оскільки мови Фортран і C як правило запобігають автоматичному виявленню компілятором паралелізму в коді.

Стандарт OpenMP був вперше випущений в 1997 р. До 2006 р. практично всі компілятори мали певний рівень підтримки OpenMP. Зрілість реалізацій варіюється, але вони достатньо поширені, тому їх слід розглядати як природний супутник мов Fortran і C, і на них можна розраховувати, при програмуванні на будь-якій платформі.

OpenMP не звільняє програміста від більшості рутинних питань паралельного програмування. Програміст має багато розуміти, в тому числі: зв'язок між логічними потоками, базовими фізичними процесорами і ядрами: як потоки спілкуються і синхронізуються; як виміряти продуктивність в паралельному середовищі; джерела навантаження розбалансування. Програміст повинен перевіряти залежності, тупики, конфлікти, стан гонки та інші моменти, пов'язані з паралельним програмуванням. На відміну від цього, Intel TBB приховує від програміста деякі з питань і автоматизує декомпозицію даних і планування завдань.

Версія 3.0 включає в себе багатозадачність, яка звільняє OpenMP від використання виключно на довгих, регулярних структурах циклу, додаючи підтримку нерегулярних конструкцій, таких як цикли з умовою і рекурсивні структури. Intel реалізувала багатозадачність у своїх компіляторах в 2004 р. Поки ці нововведення не набули достатньо широкого використання, OpenMP нагадує Фортран програмування з мінімальною підтримкою для C++.

Використовуючи OpenMP програміст має вибрати один з трьох підходів планування (статичний, керований і динамічний) для ітерацій планування циклу. TBB не заставляє програміста думати про політику планування. TBB усуває це на користь єдиного, автоматичного підходу до планування: «розділяй і володарюй». Реалізація механізму «крадіжки» роботи (техніка для розподілу завдань із завантажених процесорів на вільні) вигідно відрізняється від динамічного або керованого планування, але без проблем централізованого дилера. Статичне планування іноді швидше працює на системах, які не розподілені з іншими процесами або кодом, який працює паралельно. Зазвичай, принцип «розділяй і володарюй» підходить і добре поєднується з вкладеною паралельністю.

Узагальнене програмування, яке охоплюється TBB означає, що паралелізм структури не обмежується вбудованими типами. OpenMP дозволяє виконати зведення тільки для вбудованих типів, в той час як TBB паралельно зводить роботу з будь-якого типу.

Намагаючись усунути слабкі місця в OpenMP, TBB призначений для C++, і, таким чином, забезпечує найпростіші можливі рішення для типів програм, написаних на C++. Отже, TBB не обмежується статичною областю дії вкладеного циклу. Більше того: TBB реалізує тонку, але критичну рекурсивну модель паралелізму задач і загальних алгоритмів.

Основний принцип побудови QUARK взяв за основу модель потоку даних, де планування базується на залежностях даних між завданнями в графі завдань. Залежності даних виводяться під час аналізу неявних конфліктів даних, до яких звертаються ядра на стадії виконання. Центральна увага під час розробки зосереджується на підтримці динамічних алгоритмів лінійної алгебри для проекту лінійного алгебри PLASMA [9]. Однак, QUARK здатний підтримувати інші додатки, які можуть бути розкладені на завдання з залежними даними.

Отже, якщо брати до уваги специфіку завдання, то бібліотека QUARK найкраще підходить для його вирішення, оскільки дозволяє не тільки використовувати паралелізм задач, динамічне керування під час виконання, а й будує наглядний граф, що спростить задачу відлагодження програми в цілому. Також, важливим фактором є адаптованість розробки під алгоритми лінійної алгебри.

2.6 Висновки до другого розділу

З недавньою появою архітектур з багатьма ядрами, алгоритми повинні бути перебудовані так, щоб їх можна було використовувати в нових апаратних рішеннях. Рівень паралелізму в таких випадках стає настільки значним, що, відповідно до закону Амдала, для деяких алгоритмів їх продуктивності виявляється різко зниженою через вартість їх послідовної частини. Крім того, модель розгалуження-злиття досягла своїх меж за рахунок вартості синхронізації по ряду ресурсів, які продовжують рости. Останнім часом багато середовищ виконання намагаються обмежити і знайти рішення для цієї проблеми. Підхід на основі завдань є способом уникнути багатьох з цих бар'єрів, оскільки вони можуть бути виражені як місцеві бар'єри відносно меншої кількості завдань. А керування завданнями під час виконання на декількох обчислювальних блоках, виконуватиметься відповідно до аналізу їх залежності. Таким чином, послідовна частина початкового алгоритму іноді може бути обчислена поряд з іншими завданнями. Крім того, використання блокових алгоритмів дозволяє нам скористатися перевагою локальних даних по конвеєру операцій, які використовують одні й ті ж дані. Спільна пам'ять, залежно від архітектури, може бути розділена в блоці таким чином, щоб забезпечити хорошу ступінь градації завдання. Поєднання обох підходів відображає більше паралелізму і використовує сучасні архітектури наскільки це можливо.

Intel TBB – бібліотека, яка підтримує підхід на основі задач Вона складається з будівельних блоків (структур даних і алгоритмів), які звільняють програміста від деяких ускладнень, пов'язаних з використанням базових механізмів розпаралелювання. Паралелізм виражений явно за допомогою конструкцій TBB, а вкладений паралелізм допускається. Планувальник завдань використовує ефективний і динамічний механізм балансування навантаження на основі механізму «крадіжки» завдання для поліпшення продуктивності пропускної здатності.

OpenMP є інструментом для написання багатопоточних додатків в середовищі із загальною пам'яттю. Він складається з набору директив компілятора і бібліотеки препроцесора. Компілятор генерує багатопоточний код, заснований на вказаних директивах. OpenMP спрощує задачу створення багатопоточної програми, зберігаючи зовнішній вигляд і відчуття послідовного програмування, що дозволяє навіть новачкам поступово рухатися від послідовного стилю програмування до паралельних програм.

QUARK – це середовище виконання для динамічного планування і виконання додатків, які складаються з обмежених пріоритетом ядер на багатоядерних, мультисокетних системах з спільною пам'яттю. Основний принцип побудови взяв за основу модель потоку даних, де планування базується на залежностях даних між завданнями в графі завдань. Центральна увага зосереджується на підтримці динамічних алгоритмів лінійної алгебри.

Якщо брати до уваги специфіку завдання, то бібліотека QUARK найкраще підходить для його вирішення, оскільки дозволяє не тільки використовувати паралелізм задач, динамічне керування під час виконання, а й будує наглядний граф, що спростить задачу відлагодження програми в цілому. Також, важливим фактором є адаптованість розробки під алгоритми лінійної алгебри.

3 РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО МЕТОДУ СИНГУЛЯРНОГО РОЗКЛАДУ

3.1 Адаптація алгоритму SVD для паралельного виконання

Алгоритм обчислює SVD трьохдіагональної матриці T наступним чином:

$$T = V\Lambda V^T_3 \; V^T V = I \tag{3.1}$$

Де V – вектор власних чисел, Λ – власні значення. Алгоритм DC складається з трьох фаз:

1. Поділ матриці T на p підзадач, рекурсивно формуючи дерева.

2. Підзадачі на листку дерева вважаються простими або маленькими задачами знаходження власних чисел. Кожна з цих проблем може бути розглянута в якості незалежної проблеми без будь-яких залежностей даних з інших листів дерева і вирішена за допомогою класичного методу ітерацій QR, який забезпечує точність рішення.

3. Злиття підзадач, які визначені за допомогою однорангової модифікації трьохдіагональної матриці, і перехід до наступного рівня дерева в напрямку знизу-вверх, як показано на рисунку 3.1.

Таким чином, основна обчислювальна частина алгоритму DC це процес злиття. Для того щоб спростити описання фази злиття, встановлюємо $p = 2$, але це легко узагальнити для будь-якого $p < n$, де n є розміром матриці. Поділ проблеми при p = 2 дає:

$$T = \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + \beta u u^T, \tag{3.2}$$

де T_1 і T_2 – дві трьохдіагональні підматриці, в яких перший і останній елемент змінений відніманням β відповідно, u – вектор, де $u_i = 1$ тільки тоді, коли $i = \frac{n}{2}$ або $i = \frac{n}{2} + 1$. Припускаючи це, розв'язок матриць T_1 і T_2 буде мати вигляд $T_1 = V_1 D_1 V_1^T$ і $T_2 = V_2 D_2 V_2^T$. Таким чином, фаза злиття полягає у розв'язку системи:

$$T = \tilde{V}(D + \beta z z^T)\tilde{V}^T, \tag{3.3}$$

де $\tilde{V} = \mathrm{diag}(V_1, V_2)$ і $z = \tilde{V}^T u$. Ця система вирішується за два кроки. Спершу знаходимо спектральну декомпозицію $R = D + \beta z z^T = X \Lambda X^T$ (це називається одноранговою модифікацією), а потім ми однозначно будуємо власний вектор V матриці T виконавши множення оновленого власного вектора X з попередньо обчисленим вектором (де $\tilde{V}$ це власний вектор двох синів), отже $V = \tilde{V} \times X$. Дж. Голуб [46], [47] доказав, що якщо d_i (компоненти D) є різними і ς_i (компоненти z) відрізняються від нуля для всіх i, тоді власні вектори або вектори R є нулями $\omega(\lambda)$, де

$$\omega(\lambda) \equiv 1 + \beta \sum_{i=1}^{n} \frac{\varsigma_i^2}{d_i - \lambda}. \tag{3.4}$$

Рівняння (3.4) відоме як характеристичне рівняння. Слід зазначити, що, коли d_i рівні (це означає, що D має кілька власних чисел) або коли $\varsigma_i = 0$, або $|\varsigma_i| = 1$, то проблема має знижений ранг, тобто деякі з його кінцевих

власних пар (λ, v) точно відомі і не потребують ніяких. В результаті, процес зниження рангу матриці запобігає обчисленню власних пар (λ, v) об'єднаної системи R, які підходять для батьківського вузла (в цьому прикладі, це T). Таким чином, це зменшує як кількість характеристичних рівнянь, так і кількість множень з вектором $\tilde{\mathrm{V}}$.

Нарешті, давайте докладно опишемо фазу злиття. Вона складається з семи кроків:

1. Знаходження всіх дефляцій.

2. Переставлення векторів $\tilde{V}$ таким чином, щоб мати два набори (Вектори з не зниженим і зниженим рангом).

3. Розв’язок характеристичних рівнянь частин з не зниженим рангом (обчислення оновлених власних чисел λ і компонентів оновлених власних векторів X).

4. Обчислення значення стабілізації для кожного власного вектора відповідно до техніки М. Ґу [48].

5. Переставляння назад дефляційних власних векторів.

6. Обчислення власних векторів X оновленої системи R.

7. Оновлення власних векторів батьківської системи $V = \tilde{V} \times X$.

Метод DC є послідовним і одним з найшвидших методів, якщо всі власні пари повинні бути обчислені [7]. Він також має привабливі властивості розпаралелювання, як показано в [49].

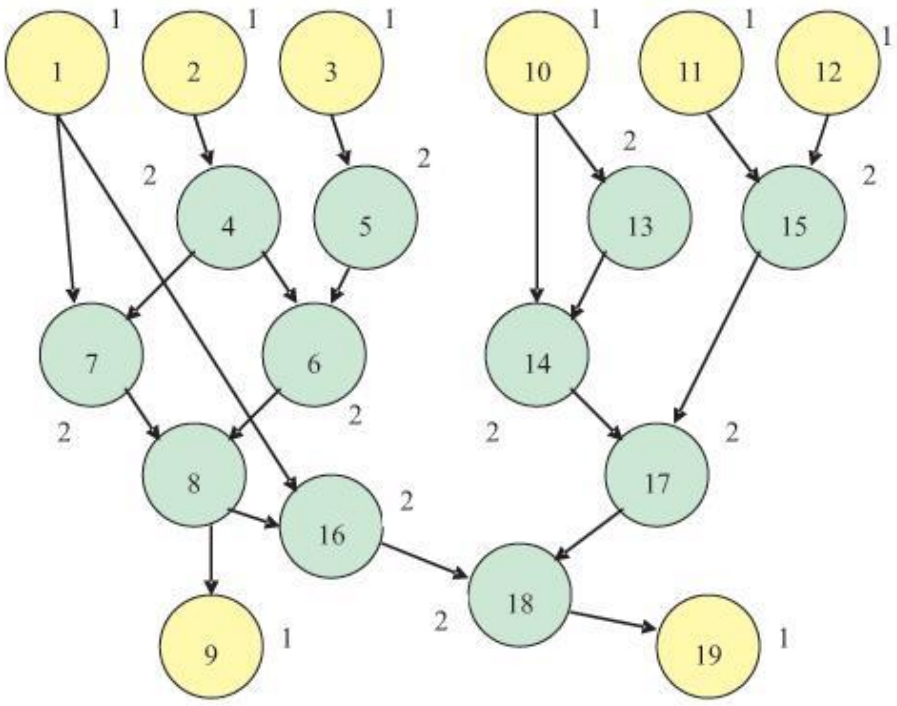


Рисунок 3.1 – Дерево злиття алгоритму DC

Враховуючи n кількість задач і k кількість не дефляційованих власних векторів, обчислювальна складність різних етапів фази злиття перерахована в таблиці 3.1.

В найгіршому випадку, коли ніякі «власні числа» не є дефляційованими, то загальна складність може бути виражена як $n^3 + 2(n/2)^3 + 2(n/4)^3 + \cdots = \sum_{i=0}^{\log(n)} n^3/2^{2i} = 4n^3/3 + \Theta(n^2)$. Варто відмітити, що в загальній складності домінує вартість останнього злиття, яке складає близько n^3 операцій.

Два передостанніх злиття вимагають $n^3/4$ операцій, а решта алгоритму вимагає тільки $n^3/12$ операцій. Враховуючи цей результат, рішення паралельно обчислити, і незалежні підзадачі, і злиття видається важливим.

Таблиця 3.1 – Обчислювальна складність операцій злиття

Операція	**Обчислювальна складність**
Обчислити кількість власних чисел з зниженим рангом	$\Theta(n)$
Переставити власні вектори (копіювання)	$\Theta(n^2)$
Розв'язати характеристичне рівняння	$\Theta(k^2)$
Обчислення значень стабілізації	$\Theta(k^2)$
Переставити власні вектори (зворотне копіювання)	$\Theta(n(n-k))$
Обчислити власні вектори X або R	$\Theta(k^2)$
Обчислити власні вектори $V = \tilde{V} \times X$	$\Theta(nk^2)$

Варто відзначити, що чим більша кількість дефляцій, тим менше число необхідних операцій, що призводить до більш високої продуктивності. Кількість дефляції залежить від розподілу власних значень, а також від структури власних векторів. На практиці більшість прикладних матриць, що використовуються в технічних областях, забезпечують розумну кількість дефляції, і тому алгоритм DC працює за $\Theta(n^{2,4})$ замість теоретичного $\Theta(n^3)$.

3.2 Стратегія паралелізму алгоритму SVD на базі парадигми «розділяй і владарюй»

Бібліотека PLASMA виражає алгоритми як послідовні. Залежності між завданнями описані за допомогою набору класифікаторів: INPUT, OUTPUT, і INOUT, які визначають тип доступу до даних. Головний потік підтверджує завдання для динамічного виконання бібліотеки QUARK [50]. Остання пізніше аналізує класифікатори для отримання залежностей між завданнями, а потім планує їх таким чином, щоб була можливість змінити

порядок виконання під час роботи. Потік приступить до роботи тоді, коли всі його залежності будуть виконані. Алгоритми, які використовують цей підхід, як правило, розділяють роботу між блоками, які покривають матрицю (квадратну або прямокутну). Блок дозволяє користувачам налаштовувати виконання як для декількох великих ефективних ядр, так і для численних дрібних ядр, що забезпечує більше паралелізму і покращену масштабованість. Дані часто розподіляються за однаковою схемою нарізки для поліпшення розміщення даних у ядрі ОС.

Оскільки обчислення кожного власного вектора в основному залежить від характеристичного рівняння (яке обчислює компоненти u_i власного вектора) з подальшою стадією стабілізації, яка також буде виконувати операції на векторі, було вирішено реалізувати алгоритм DC за допомогою розподілу панелі завдань. Операції над матрицями діляться на набір операцій на власних векторах. Це значно спрощує управління залежностями завдань у DC алгоритмі, і дозволяє напряму викликати внутрішні підпрограми LaPack, такі як LAED4 для обчислення характеристичного рівняння для кожного власного вектора. На відміну від класичного алгоритму, відстеження залежностей даних у потоці виконання є не таким простим завданням і вимагає більше уваги через динамічний розрахунок кількості не дефляційованих власних чисел. Максимального зменшення витрати пам’яті потрібно пам’ятати, що необхідний робочий простір алгоритму залежить від цього обчисленого значення дефляції і, таким чином, розмір робочого простору є динамічним. Для вирішення динамічності вихідного результату, було вирішено зберегти той же послідовний стиль Task Flow, який виглядає простим. Тим не менш, у цьому випадку завдання, пов'язані з пониженими чи не пониженмим рангами власних векторів є затвердженими. Це створює додаткове завдання без обчислювальної роботи. Останній здається маргінальним в порівнянні з обчислювальною складністю інших завдань. У результаті, згенерований завданням граф (DAG) є незалежним від матриці:

буде стільки ж завдань для матриці з великою кількістю дефляції, як і для матриці без дефляції.

Лістинг 3.1 Кроки злиття

```
ComputeDeflation (V)
for кожного власного вектора набору p do
        PermuteV (V_p,V_d)
        LAED4(V_p)
        ComputeLocalW(V_p,W_p)
end for
ReduceW(V, W )
for кожного власного вектора набору p do
        CopyBackDeflated(V_p,V_d)
        ComputeVect(V_p,w)
        UpdateVect(V_p,V_d)
end for
```

Більшість етапів фази злиття можуть бути перероблені паралельним чином шляхом поділу операцій з підмножини власних векторів, тобто панелі матриці V. Паралельний алгоритм, описаний в лістингу 3.1, складається з восьми частин

Compute deflation: знайти, якщо є можливість дефляції і згенерувати перестановку, щоб змінити власні вектори або числа на чотири групи: без дефляцій на підзадачі V_1, корельовані власні вектори між V_1 і V_2, без дефляцій на підзадачі V_2, дефляційовані «власні числа».

PermuteV: перевпорядкувати набір векторів з V в робочий простір в стислому вигляді: не зберігати нулі нижче V_1 і вище V_2 з рівняння (3.3). Дефляційовані вектори сортуються в кінці цього буферу.

LAED4: розв’язує характеристичне рівняння для кожного не дефляційованого власного числа, обчислюючи нове власне число λ для

однорагової модифікації задачі R і генеруючи компоненти нового власного вектора. Кожна задача являє собою панель і, таким чином, працює на множині nb векторів.

ComputeLocalW: етап стабілізації, розділений на два завдання. Перше це паралельний набір функцій ComputeLocalW, які незалежна для кожної панелі, а друге – функція ReduceW, яка виконує зведення, щоб згенерувати вектор стабілізації W. Отже, ComputeLocalW функція полягає в обчисленні локального внеску W_p до стабілізації вектора W.

ReduceW: обчислити стабілізаційний вектор W зі всіх панелей.

CopyBackDeflated: здійснити зворотне копіювання дефляційованих векторів до кінця батьківського власного вектора матриці V. На цьому рівні, коли попередні кроки закінчені, ми знаємо всі перестановки і розміри дефляцій, так що ми можемо виконати це зворотне копіювання, щоб дозволити використовувати робочий простір, який використовується в наступних кроках.

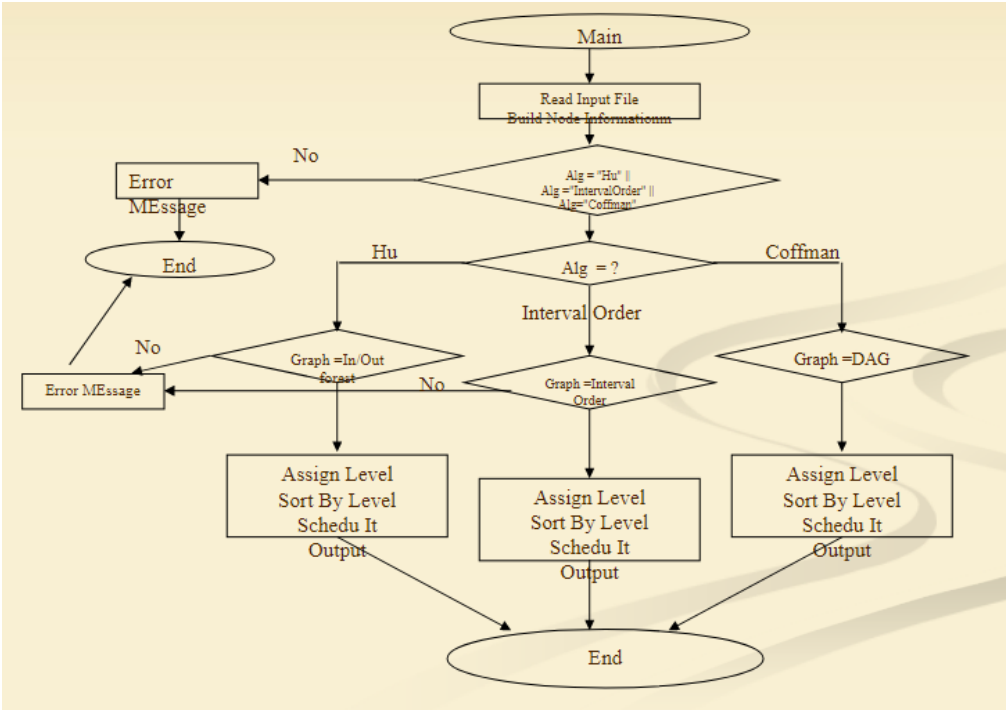
ComputeVect: стабілізує і обчислює нові не дефляційні власні вектори X. Завдяки вектору W.

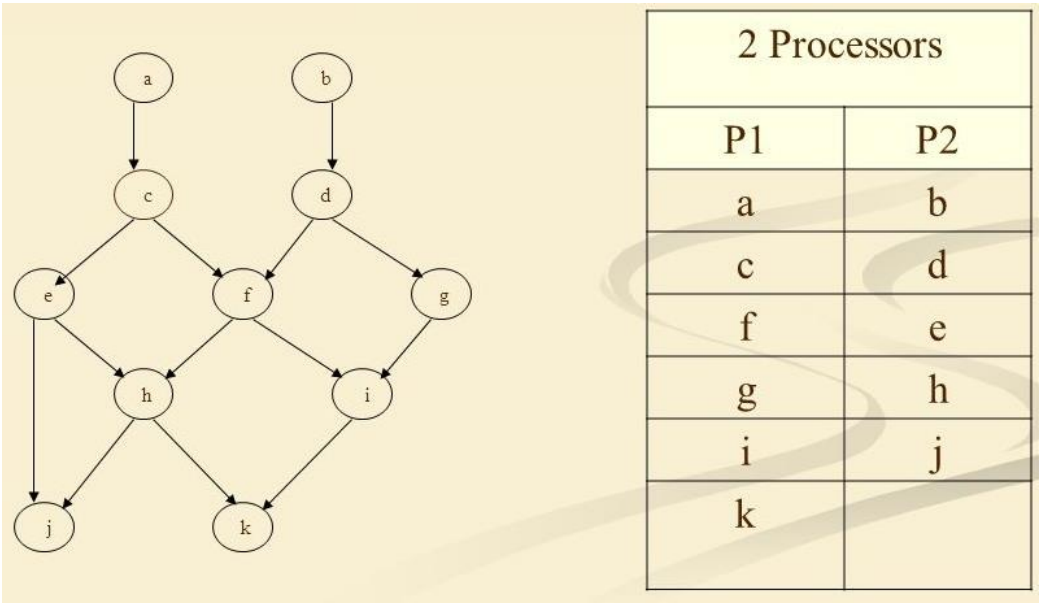
UpdateVect: обновляє не дефляційні власні вектори X, виконуючи множення матриць $V = \tilde{V} \times X$, описане в рівнянні (3.3).

Як показано в алгоритмі на лістингу 3.1, тільки два ядра залишаються послідовними і створюють операцію об'єднання: Compute deflation і ReduceW. Тим не менше, ці ядра дуже швидкі (вони становлять менше 1% від загального часу обчислень), і вони є локальними для кожної фази злиття. Таким чином дві незалежні фази злиття мають дві незалежні функції Compute deflation і ReduceW ядра, які можуть працювати паралельно. Це не випадок з моделі «розгалуження-злиття», представленій в реалізації MKL LAPACK, яка використовує багатопотокову бібліотеку BLAS. Глибокий аналіз моделі

потоку задач дозволяє припустити, що деякі завдання мають залежність від результатів обчислення матриці в кожній панелі і тому загальна складність складає $\Theta(n)$. Для того, щоб мінімізувати кількість відстежуваних залежностей, було розроблено класифікатор GATHERV для бібліотеки QUARK. Таким чином, всі завдання мають постійну кількість залежностей, тобто робота на панелі має додаткову залежність від повної матриці з прапором GATHERV, в той час як задача приєднання (Compute deflation і ReduceW) має одну залежність INOUT на повній матриці. Усі завдання з прапором GATHERV можуть бути заплановані для одночасного виконання, так як розробники гарантують, що вони працюють з незалежними даними, і завдання об’єднання чекають їх завершення перед тим як планувати подальшу роботу. Після того можна описати роботу, аналогічну до моделі розгалуження-злиття для кожної підзадачі з постійною кількістю залежностей на задачу. Це знижує складність алгоритму відстеження залежностей. Крім того, паралелізм завдань і ступінь деталізації цього методу керується розміром панелі nb, яка є параметром, який визначається розміром задачі, але також може бути обрана користувачем. Зазвичай nb повинен бути покращений, щоб скористатися такими критеріями, як кількість ядер (які визначають величину паралелізму, необхідного для повного завантаження ядер) і ефективність самого ядра (обчислювання ядра можу мати різну поведінку в залежності від малого чи великого значення nb). Залежності між фазами об'єднання повинні бути оброблені правильно, щоб дозволити незалежним підзадачам виконуватись паралельно. Оскільки по дереву йдуть знизу-вверх, то різні процеси злиття на тому ж рівні дерева завжди незалежні і, таким чином, можуть бути вирішені паралельно.

Крім того, ми можемо стверджувати, що процес об'єднання між двома різними рівнями дерева є незалежним, тоді і тільки тоді, коли вони не належать до однієї гілки дерева (див. рисунок 3.2).





2 Processors	
P1	P2
a	b
c	d
f	e
g	h
i	j
k	

Рисунок 3.2 – DAG алгоритму DC

Рис. 3.2 показує DAG і його залежності, отримані для задачі розміром 1000 елементів, рекурсивно розділені в глибину до досягнення підзадачі в нижній частині дерева розміром менше або рівне 300 елементів (мінімальний розмір розподілу це 300 елементів). Це зведеться до чотирьох підзадач

розміром 250 елементів кожна. Ми визначили розмір панелі nb = 500, це означає, що наші завдання розщеплюється на панелі по 500 стовпців. Перші дві вузли DAG є завданнями, які готують і розділяють задачу, тоді ми можемо побачити два підграфи (виділено прямокутними межами), які відповідають двом незалежним процесам об'єднання; слідуючи за підграфом в глибину ми побачимо остаточний процес злиття. І, нарешті, якщо вихідна матриця T була скорочена, ми повинні повернути її до початкових розмірів.

На початку, чотири незалежні власні значення обчислюються послідовно, використовуючи STEDC ядра. Потім, під час виконання, будуть плануватися паралельно два незалежні злиття процесів на тому ж рівні, без координації процесу кожного злиття. Рис. 3.2 дозволяє пояснити важливість розпаралелювання, яке було запропоноване.

Незважаючи на те, що алгоритм DC представляє дерево паралелізму, запропоноване розбиття панелі на комірки розміром nb створить додаткові завдання, що породжують більше паралелізму, ніж ми можемо використовувати чи то в нижній, чи в верхній частині DAG. Кожен рівень DAG складається з завдань, які можуть працювати паралельно. Наприклад, на рис. 3.2, протягом перших двох злиттів, п'ятий рівень DAG, приймаючи, що дефляції немає, кожен LAED4 має розмір 500 (це відповідає процесу злиття двох синів розміром 250 кожен).

Оскільки розглянута панель nb = 500, в результаті маємо одну LAED4 задачу злиття 1-го і 2-го вузлів, а інша паралельна задача LAED4 злиття вузлів 3 і 4 тоді як, при заміні nb на 100, можна створити десять завдань. Цікаво, що на малюнку 2 можна бачити, що завдання перестановки можуть працювати паралельно із завданням LAED4, в той час як в алгоритмі 1 вони є послідовними. Отже, після аналізу алгоритму виявлено, що можна створити додаткову паралельну задачу, вимагаючи додатковий робочий простір. Для цього ми інтегрували користувацький сценарій, що дозволяє використовувати алгоритм з додатковим робочим простором і, таким чином,

можна створити кілька паралельних завдань. Наприклад, перестановка може бути виконана паралельно з LAED4 якщо є додатковий робочий простір. Аналогічним чином, Copyback може бути виконана паралельно з ComputeVect, якщо можна використати додатковий робочий простір. На практиці ефект від цього варіанту можна побачити на машині з великою кількістю ядер, де потрібно породжувати якомога більше паралелізму.

3.3 Стратегія паралелізму для алгоритму MRRR

Алгоритм MRRR – перший стабільний метод, який обчислює k власних пар за O(nk) арифметичних операцій [14, 15]. Завдяки своїй низькій складності, MRRR є одним з найшвидших алгоритмів для обчислення власних значень [51]. У той час як для великих завдань середовище з розподіленою пам'яттю є необхідністю, малі та середні задачі, як правило, обробляються на одному процесорі, який зараз складається з декількох обчислювальних ядер, або на системах з спільною пам’яттю. Тим не менш, сучасні реалізації для розподілених систем не можуть повністю використовувати паралелізм, який пропонується багатоядерними архітектурами. MP^3-SMP спеціально розроблена, щоб скористатися особливостями багатоядерних і багатопроцесорних систем.

При виконанні на багатоядерних архітектурах, які алгоритм будуть швидшими додатково залежить від кількості наявного паралелізму. На рисунку 3.3 наводимо час для обчислення всіх власних пар як функцію від кількості потоків, які використовуються; вхідна матриця розміру n = 12387 це кількість закінчених елементів моделі автомобіля. Показана стандартна імплементація DC і MRRR, також нова бібліотека MP^3-SMP. BI обчислюється близько 2-х годин і QR більше ніж 6 годин. Це значно повільніше і тому не показано.

На рисунку 3.3 зліва показаний час виконання, а на рисунку 3.3 справа – прискорення. Нахил кривої MR^3-SMP для 24 потоків позитивний, вказуючи на те, що більше апаратного паралелізму дасть ще більш високе прискорення.

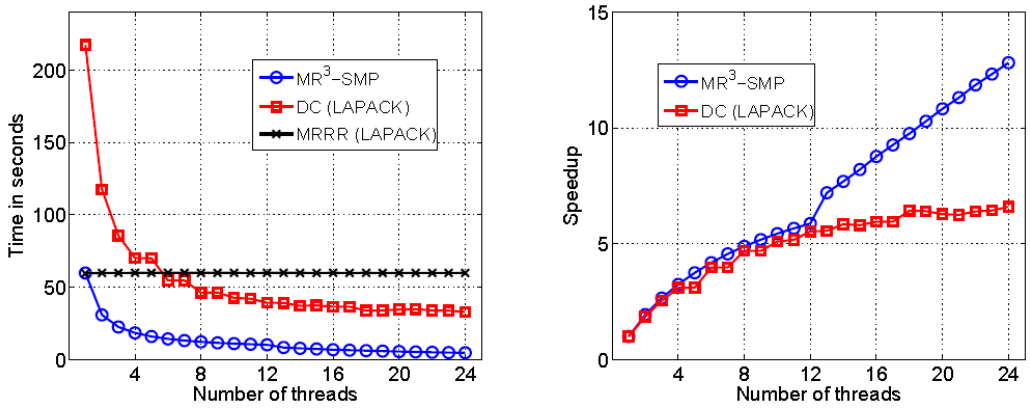


Рисунок 3.3 – Виконання MR3-SMP та DC і MRRR для матриці розміром 12387.

У цьому розділі представляємо конструкцію MR^3-SMP, паралельну версію алгоритму MRRR спеціально створену для багатоядерних процесорів і архітектур з спільною пам'яттю. Реалізація MR^3-SMP на C використовує потоки POSIX. Оскільки було запропоноване декілька оптимізації для підвищення точності алгоритму MRRR [52], MR^3-SMP було розроблено модульно, так щоб алгоритмічні зміни могли бути включені з мінімальними зусиллями.

Версія MRRR з розподіленою пам'яттю спрямована на мінімізацію зв'язку між процесорами при досягненні оптимального балансу вимог до пам'яті [53, 54]. Цей підхід виконується за рахунок балансування робочого навантаження, тоді як поділ праці здійснюється статично. Так як на

багатоядерних архітектурах зі спільною пам'яттю про баланс можна не турбуватись, метою є визначення правильної обчислювальної глибини, щоб краще збалансувати робоче навантаження.

Алгоритм, показаний на лістингу 1.1 розділяється на дві частини, відповідно до рядків 2-4 і 5-14 у відповідно.

- обчислення кореневого представлення і початкового наближення власних чисел;
- розрахунок власних векторів, і остаточне уточнення власних значень.

У першій частині обчислення кореня RRR (як і для кожного підкореня RRR) займає O(n) і виконується послідовно. Початкове наближення власних значень коштує O(n) для кожного власного числа і може бути виконане паралельно з алгоритмом бісекції. Тим не менш, в залежності від доступного паралелізму, послідовний алгоритм dqds швидший, ніж бісекція і йому слід віддати перевагу [53]. Приймаючи, що бісекція використовується для обчислення власних чисел з половиною машинної точності, а алгоритм dqds сходиться протягом трьох ітерацій до власного числа, алгоритм dqds є кращим, ніж бісекція, якщо число процесорів менше або рівне 12 |Г|/n.

Для другої частини алгоритму на лістингу 1.1, стратегія розпаралелювання полягає в розділенні обчислення на завдання, яке може бути виконане декількома потоками паралельно. Можна використовувати багато різних стратегій для поділу і планування обчислень.

Хоча власні числа уточняються у другій частині алгоритму, для простоти приймемо обчислення власного значення і обчислення власного вектора першою і другою частиною відповідно.

Оскільки дерево представлення характеризує розгортання алгоритму, природно зв'язати кожен вузол в дереві з блоком обчислень. Через таке відношення один до одного, можна взаємозаміняти поняття завдання і вузла. У відповідності з внутрішніми вузлами і кінцевими вузлами введемо два

типи C-завдання і S-завдання. C-завдання мають справу з обробкою згрупувань та створюють інші завдання, в той час як S-завдання відповідальні за кінцеві обчислення власних пар. C-завдання і S-завдання втілюють дві гілки умови в стрічці 7 алгоритму на лістингу 1.1: рядки 8-10 і 12 відповідно.

Незалежно від стратегії планування, ступінь деталізації цього першого підходу заважає паралельному виконанню завдань: Якщо згрупування достатньо велике, то потоку може не вистачити для обробки завдань і розбиття його на більш дрібні підзавдання Для прикладу, припустимо, що «власні числа» від λ_2 до λ_9 на рис. 1.1 є згрупованими і що ми хочемо вирішити задачу на 4-ядерному процесорі. У цьому випадку, одне з чотирьох ядер буде обчислювати сингелтон {1}, друге ядро буде опрацьовувати згрупування, а третє і четверте ядро будуть чекати, поки згрупування не розкладеться. Оскільки обчислення групи займає більше, ніж обробка одинака, перше ядро буде простоювати в очікуванні нових доступних завдань.

Щоб зменшити ступінь деталізації обчислень введемо третій тип завдань – R-задачі. Вони дозволяють декомпозицію уточнення власного числа – найдорожчий крок у C-завданні – створення завдання, яке можна негайно виконувати. R-задача приймає як вхідні дані RRR і список власних чисел. На вихід він повертає уточнені «власні числа».

Розрахунок пов'язаний з C-задачею зведено в лістингу 3.2, де позначки #left і #threads означають число власних пар, які ще не обчислені і кількість використовуваних потоків, відповідно.

Лістинг 3.2 – Алгоритм виконання C-задач

```
Input: RRR, «власні числа» λ̂_i і встановлений індекс Γ_c для
       згрупування
```

Output:S-задачі і C-задачі, які асоціюються з підзадачами групи $\Gamma_{c|}$.

1. Викликати підпрограму ComputeNewRRRforCluster
2. RefineEigenvaluesOfCluster
3. SplitIntoClustersAndSingeltons

ПІДПРОГРАМА: ComputeNewRRRforCluster
Input: RRR, «власні числа» $\hat{\lambda}_i$ і встановлений індекс Γ_c для згрупування
Output: RRR_{new}
1. Обчислити RRR_{new}, RRR для λ_i, де $i \in \Gamma_c$, використовуючи DLARRB

ПІДПРОГРАМА: RefineEigenvaluesOfCluster
Input: RRR_{new} «власні числа»$\hat{\lambda}_i$ і встановлений індекс Γ_c для згрупування
Output: Удосконалені «власні числа» λ_i, де $i \in \Gamma_c$.
1. if $|\Gamma_c| > \lceil \#left\#threads \rceil$ then
2. Декомпозиція удосконалених власних чисел наR-задачі
3. else
4. Удосконалити «власні числа» $\hat{\lambda}_i$ де $i \in \Gamma_c$, використовуючи DLARRB
5. end if

ПІДПРОГРАМА: SplitIntoClustersAndSingeltons
Input: RRR_{new}, «власні числа» $\hat{\lambda}_i$ і встановлений індекс Γ_c для згрупування
Output:S-задачі і C-задачі.
1. Розділити Γ_c на підмножини $\Gamma_c = U_i\Gamma_i$, асоційовані з відносними розривами
2. Вийняти з черги кожне (Γ_i, RRR_{new})

Для керівного принцип балансу навантаження, запевняємо, що кожного разу, коли створюється задача, її розмір не перевищує $S_{max} = \lceil \#left/\#threads \rceil$ власних пар, тому що з цього моменту кожен потік відповідатиме за обчислення не більше S_{max} власних пар. Як наслідок, згрупування власних чисел більших, ніж це число буде уточнюватися паралельно декількома потоками.

І навпаки, щоб уникнути занадто тонкої деталізації, розширюємо S-завдання для обчислення власних пар для декількох сингелтонів з тієї ж RRR; назвемо цю стратегію «комплектація». Подібно до R-задач, вибираємо

максимальний розмір комплекту пропорційним S_{max}. Комплектація має три переваги: краще використання кешованих даних; менше завдань і менша конкуренція на чергу задач; можливість зниження загальної вимоги пам'яті до алгоритму. Алгоритм з лістингу 3.3 узагальнює обчислення, які виконують S-завдання, де k_1 і k_2 відповідні константи, і RQC це скорочення від Релеєвського коефіцієнта поправки.

Нещодавно було показано, що обчислення багатьох щільних алгоритмів лінійної алгебри можуть бути ефективно перенесені на багатоядерні архітектури з використанням підходу завдання на основі черги [55, 56]. Як правило, кількість і тип завдання відомі ще перед виконанням, незалежно від конкретних вхідних даних. Це дозволяє планувати завдання, статично чи динамічно, беручи до уваги залежності. З іншого боку, розгортання алгоритму MRRR, тобто, форма дерева представлення, в значній мірі залежить від вхідної матриці і не може бути відомою до виконання. Крім того, завдання генеруються динамічно, під час обробки кінцевих вузлів. Далі проаналізуємо залежності даних в завданнях і їх можливе планування.

Лістинг 3.3 – Алгоритм виконання S-задач

```
Input: RRR (L, D) «власні числа» і множина індексів Γ_s
сингелтонів
Output: Власні пари (λ̂_i, ẑ_i), де i ∈ Γ_s
1.  for кожен індекс i ∈ Γ_s do
2.  while Власні пари (λ̂_i, ẑ_i), не прийняті do
3.  Викликати DLAR1V (LDL^T − λ̂_i I)ẑ_i = γ_r e_r
4.  Записати залишкову норму |γ_r|/‖ẑ_i‖_2 і RQC |γ_r|/‖ẑ_i‖_2^2
5.  If залишкова норма < k_1 nεgap(λ̂_i) або RQC < k_2 ε|λ̂_i| then
6.  Прийняти власну пару
7.  end if
8.  if RQC не поліпшує λ̂_i then
9.  Використати бісекцію, щоб удосконалити λ̂_i щоб підвищити
                відносну точність
10. end if
11. end while
```

12. Нормалізувати власний вектор $\hat{z}_i \leftarrow \hat{z}_i / \|\hat{z}_i\|_2$
13. end for
14. Збільшити на 1 #left $|\Gamma_s|$

Щоб диференціювати пріоритет завдання, ми формуємо три робочі черги для завдань високого, середнього та низького пріоритету відповідно. Кожен активний потік опитує чергу в порядку пріоритезації до тих пір, поки доступне завдання не буде знайдено. Тоді задача видалиться із черги і виконається. Процес повторюватиметься доти, доки всі власні пари не обчисляться.

На виході алгоритму в лістингу 1.1 утворюється вектор k власних чисел і матриця власних векторів $Z \in \mathbb{R}^{n \times k}$. Ці структури даних є спільними для всіх завдань, які пов'язані з обчисленням. Так як індекс множини Γ_j – асоційованих з C- і S-завдань, які можуть бути виконані паралельно – не перетинається, ніяких залежностей даних не виникає.

Залежності між завданнями виникають тільки в результаті керування RRR. Кожна задача в черзі задач вимагає свого батьківського RRR, який служить в якості входу в алгоритмах з лістингів 3.2 і 3.3, тому RRR повинні бути доступні, поки всі його підзавдання не обробляться. Одним з можливих рішень є динамічно виділяти пам'ять для кожного нового RRR, передавати адреси пам'яті для підзавдань і не тримати в пам'яті RRR, поки всі підзавдання виконуються. Приклад наведено на вузлі {6, 7, 8, 9} на рис. 1.1. RRR (L_2, D_2) повинен бути доступний до тих пір, поки дочірні вузли {6, 7}, {8} і {9} будуть оброблені. Це рішення має наступні переваги: коли кілька потоків виконують дочірні C-завдання, тільки одна копія батька RRR знаходиться в пам'яті і є загальною для всіх потоків; аналогічно, потрібна допоміжна величина $dl(i) = D(i,i) \cdot L(i+1,i)$ і $dll(i) = D(i,i) \cdot L(i+1,1)^2$ для всіх $i = 1, \ldots n-1$, оцінюються раз і є

спільною; в архітектурі зі спільним кешом, додаткові переваги можуть виникнути від повторного використання кешованих даних.

З іншого боку, цей підхід потребує додаткового простору $O(n^2)$ роботи в найгіршому випадку. Для того, щоб зменшити необхідний обсяг пам'яті, зробимо наступну поправку. Кожна задача може бути асоційоватися з частиною Z, відповідно до Γ_c в алгоритмі до лістину 3.2 або Γ_s в алгоритмі до лістингу 3.3 з локальним робочим простором, тобто, в розділ пам'яті не доступний для будь-якого іншого завдання в черзі задач. Продовжуючи приклад вузла {6, 7, 8, 9} на глибині один дерева на рис. 1.1, стовпці 6-9 вважаються локальними сховищами для відповідної задачі.

Якщо для RRR потрібно на вході зберігати завдання в місцевому робочому просторі, то ми називаємо таке завдання незалежним. Якщо всі дочірні завдання є незалежними, то завдання RRR більше не потрібне і може бути відкинутим. Крім того, всі дочірні завдання можуть бути виконані в будь-якому порядку без будь-якої залежності від даних. Тепер покажемо, що якщо всі C-завдання можуть бути незалежними, то практично всі S-завдання можна зробити незалежними теж.

Оскільки група складається як мінімум з двох власних чисел, то відповідна частина Z – використовується як локальний робочий простір – завжди досить великий, щоб вмістити RRR, визначається діагональними елементами D і під діагональними елементами L. Тому використовується Z, щоб зберегти батьківський RRR. На жаль, приведення завдання до незалежного стану супроводжується невеликим накладними витратами за рахунок: зберігання батьківського RRR в Z; отримання батьківського RRR з Z; перерахунку $dl(i)$ і $dll(i)$. Крім того, негативний вплив на продуктивність може бути викликаний відсутністю повторно використовуваних кешованих даних.

Такий же підхід є допустимим для S-завдань, коли принаймні, два сингелтони в комплекті. З іншого боку, підхід не може бути застосований в крайньому випадку, коли група розпадається на невеликі групи плюс тільки один сингелтон. Листок {2} на рис. 1.1 являє собою такий виняток. Всі інші S-завдання можуть бути об'єднані в групи з двох або більше, а, отже, можуть бути приведені до незалежної форми. Як і раніше, цей підхід вводить деякі додаткові витрати: зберігання батьківського RRR в Z; дублювання RRR, щоб зробити Z доступним для власних векторів; перерахунку dl(i) і dll(i).

Недоліком цього підходу є те, що, коли кілька дочірніх S-завдання одного вузла обробляються одночасно, кілька копій одного й того ж RRR перебувають в пам'яті, не дозволяючи повторно використовувати кешовані дані. На прикладі вузла {6, 7, 8, 9} на рис. 1.1, зберігання RRR (L_2, D_2) в стовпці 6 і 7, а також колони 8 і 9, створює незалежне C-завдання для вузла {6, 7}, і незалежне S-завдання {8, 9}, де два сингелтони {8} і {9} є в комплекті.

Відзначимо ще раз, що робота з незалежними завданнями супроводжується деякими накладними витратами, це вносить додаткову гнучкість у планування, а завдання тепер можуть бути заплановані в будь-якому порядку.

Багато стратегій можуть бути використані для динамічного планування завдань. В якості загальної рекомендації, в середовищі із спільною пам'яттю, що має досить завдань, першочергове значення для досягнення високої продуктивності має використання всіх обчислювальних ядер. Покажемо реалізацію двох простих, але ефективних стратегій, таких як балансування деталізації завдання, вимоги до пам'яті, а також створення завдань. В обох випадках було реалізовано три окремі черги FIFO, щоб уникнути конкуренції при використанні багатьох процесорів. Крім того, обидва підходи планують високопріоритетні R-задачі, так як вони виникають як частина декомпозиції великих груп, і породжують роботу у формі завдань.

Стратегія, де C-завдання, що стоять перед S-завдань, має свої особливості. Усі поставлені в чергу C-завдання і S-завдання виконуються незалежно (крім рідкісних випадків, коли S-завдання не може бути приведеним до незалежності, завдання виконується негайно, без черги). Як наслідок, всі C-завдання і S-завдання в чергах задач можуть бути заплановані в будь-якому порядку. Так як обробка C-задач створює нові завдання, які можуть виконуватися паралельно, вводимо C-завдання (середнього пріоритету) з черги швидше ніж S-завдання (низький пріоритет). Це упорядкування є окремим випадком з безлічі різних стратегій, в яких виконання C-завдань і S-задач чергуються. За сценарієм всі завдання є незалежними, отже можна досягти максимальної продуктивності.

Розглянемо випадок, якщо немає незалежного S-завдання, тобто другу стратегію. Для того, щоб отримати аналогічні до першого підходу вимоги до пам'яті, доводиться планувати S-завдання перед C-завдань. Причина в тому, що для кожного згрупування RRR повинен зберігатися в пам'яті, поки всі його дочірні S-завдання не виконаються. Сортування гарантує, що в будь-який момент часу буде не більше ніж #threads груп S-завдань в черзі. У той час як гнучкість у плануванні завдання зменшується, ми уникаємо всіх накладних витрат від створення незалежних S-завдань. На практиці другий підхід трохи швидший, на, близько, 5-8%, а тому використовується для всіх графіків порівняння по часу виконання.

На рисунку 3.4 показано використання пам’яті для двох матриць різного роду, Ермітова і симетрична Клементова матриці [57], виміряно пікове використання пам'яті для MR^3-SMP в порівнянні з наведеною вище оцінкою.

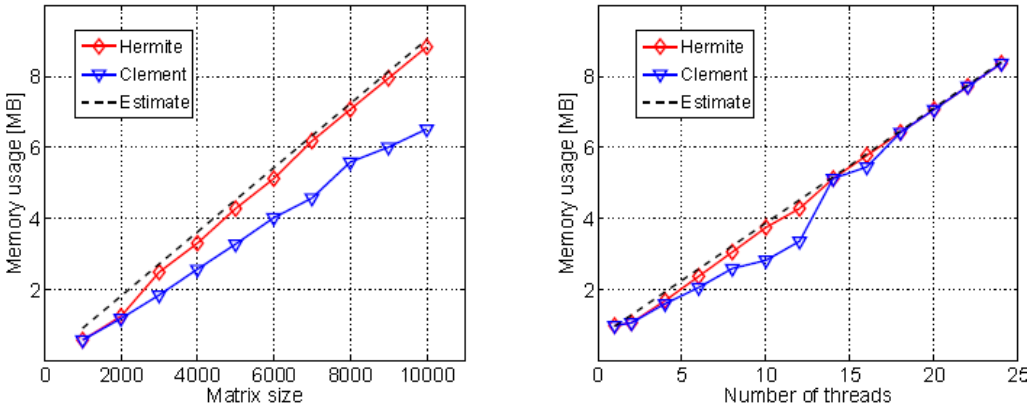


Рисунок 3.4 – Використання пам’яті алгоритмом MR^3-SMP для обчислення матриць різного роду

Графік зліва показує, що, встановивши #threads 12, додатковий обсяг оперативної пам'яті лінійно зростає відносно розміру матриці n. На графіку праворуч розмір матриці має фіксоване значення n = 5000 і відображає залежність необхідної пам'яті від числа потоків. Хоча вимоги до пам'яті зростають з кількістю потоків, навіть якщо використовуються 24 потоки, необхідне додаткове місце для зберігання становить менш ніж 5% від необхідної пам'яті для вихідної матриці Z.

MR^3-SMP ефективно використовує пам'ять за допомогою використання матриці власних векторів Z як тимчасового сховища. Крім того можна перезаписувати вхідну матрицю T для зберігання тільки RRR для вузла, який обробляється в даний момент. Цей підхід не можливий в паралельному програмуванні, тому що одночасна обробка декількох вузлів вимагає зберігання пов'язаних RRR окремо. Крім того, в багатопотоковій реалізації кожен потік вимагає свій власний робочий простір. Як наслідок, розпаралелювання і приріст продуктивності в наслідок того вимагає трохи більше пам'яті, ніж послідовна реалізація.

Так як алгоритм DC вимагає постійного O (n^2) для робочого простору подвійної точності, перевагою MRRR є потреба у меншій пам'яті, ніж DC.

3.4 Порівняння алгоритмів MRRR і DC

Порівняння DC з MRRR є досить складним. DC є ефективним, коли «власні числа» згруповані, в той час як MRRR швидкий, коли «власні числа» добре розділені. DC вимагає більшого робочого простору, але його точність вища. Всі ці параметри дозволяють припустити, що вибір одного алгоритму залежить від задачі. Обидві імплементації (бібліотека PLASMA і MR^3-SMP) завжди швидші, ніж відповідні імплементації в Intel MKL з такою самою точністю. Теоретично, чим повільніша MR^3-SMP, тим швидший DC і навпаки. Згідно з дослідженнями [58], для більшості тестів, покращений DC алгоритм швидший, ніж MR^3-SMP і ця різниця може становити до 25 разів швидше, а в окремих випадках він може бути повільнішим максимум в 2 рази. Використовуючи поліпшення отримуємо цікаві результати, оскільки більші матриці швидше обчислюються алгоритмом DC, ніж при використанні MRRR. В програмі, де продуктивність грає одну з найважливіших ролей, ми можемо припустити, що вибір DC або MRRR буде залежати від матриці. Тим не менш, основною перевагою DC є точність, яку він гарантує. Теоретично, для матриці розміром n і машинною точністю ε, DC має похибку O (n), тоді як похибка MRRR обчислюється як O ($n\varepsilon$). Крім того випадок з найгіршою точністю для MR^3-SMP є одночасно випадком, коли цей алгоритм працює найшвидше. Розглянута реалізація DC перевершує MR^3-SMP на майже у всіх випадках, забезпечуючи кращу точність. Тим не менше, експериментальні матриці є невеликими, і важко зробити висновок про масштабованість й ефективність.

3.5 Висновки до третього розділу

Отже, була проведена адаптація алгоритму SVD для паралельного виконання. Алгоритм був розділений на три фази і найзатратнішою стала фаза злиття, яка була розділена на 7 кроків. Також, був розроблений спеціальний DAG алгоритму, який дозволяє запобігти залежностям між даними на цій фазі і допомагає ясніше зрозуміти принцип роботи паралельних потоків на цьому етапі. Така імплементація алгоритму DC є новою і виражає паралелізм через модель потоку задач. Проведені експерименти показують, що ця імплементація перевищує по продуктивності існуючі DC реалізації з тією ж самою точністю.

Підхід на базі задач, використовуючи середовище виконання QUARK, забезпечує хорошу продуктивність. Крім того, експерименти по ефективності показали, що ця імплементація конкурентна з найкращою імплементацією MRRR для архітектур зі спільною пам’яттю. Враховуючи сильні і слабкі сторони обох алгоритмів, і той факт, що задачі в основному залежить від матриці, вибір того чи іншого методу залежить від програми. Додаткова пам'ять, яка необхідна DC може бути проблемою, але робастість алгоритму гарантує, що буде отримано точний результат. До того ж, розглянутий алгоритм забезпечує приріст обчислення матриці, яка відображає реальні дані.

4 ОБЧИСЛЮВАЛЬНИЙ ЕКСПЕРИМЕНТ ТА РЕЗУЛЬТАТИ ВИКОРИСТАННЯ ПАРАЛЕЛЬНОГО МЕТОДУ

4.1 Проведення обчислювального експерименту

Експеримент відбувся на сервері фірми Dell PowerEdge 9-го покоління. Сервери було розроблені згідно специфікації поведінки Dell, яка визначає послідовність комбінування обладнання і взаємодію з користувачем для всіх моделей, які були і будуть розроблені. Завдяки новітнім процесорам Intel Xeon, сервери PowerEdge 9-го покоління забезпечують очікувану потужність і продуктивність.

Сервер містить шість внутрішніх відсіків для жорстких дисків, забезпечуючи до 4,5 ТБ внутрішньої пам'яті, допомагаючи зберегти простір для додаткового розміщення необхідних компонентів. На використовуваному сервері стоїть 2 диски по 1 ТБ кожен. Також сервер включає в себе кілька вбудованих функцій, таких як dual Gigabit NIC та інтегрованй контролер пам'яті SAS, яка дає доступ до трьох слотів PCI Express для кращої пропускної здатності I / O. Вони дозволяють гнучке розширення для підтримки широкого спектру робочих навантажень, що дуже важливо при роботі з паралельною обробкою даних. Dell PowerEdge 2950 сервер підтримує до двох останніх 4-ядерних процесорів Intel Xeon і чіпсета Intel 5000X. Саме ці процесори забезпечили вісім фізичних ядер для паралельних обчислень. Додатково, пропускна здатність пам'яті становить 32 ГБ, що відповідає буферній DIMM пам'яті. Отже, ми можемо не зважати на додаткові витрати пам’яті при роботі певних алгоритмів.

Проведемо короткий підсумок необхідних приготувань для проведення експерименту. Було необхідно подати на вхід два типи даних: реальні дані по газоспоживанню з інтервалом 1 година, тобто часовий ряд довжиною 8760 елементів. Також, згенеруємо вектор випадкових чисел такої

ж довжини з розкидом в межах мінімального і максимального елементів реальних даних. Далі необхідно сформувати з нього траєкторну матрицю. Наступні кроки, тобто стратегія розпаралелення описана в 2 розділі. Крок гусениці L вибираємо N/2, де N – розмірність траєкторної матриці. На рисунку 4.1 бачимо заміри часу і обсягу пам’яті при використанні алгоритму MRRR для даних газоспоживання.

```
Login as: svystun
svystun@91.198.10.16' password:
Linux masenergy.tntu.edu.ua 3.2.0-4-int64 #1 SMP Debian 3.2.68-1+deb7u4 x86_64
The programs included with the Debian GNU/Linux system are free software; the
exact distribution terms for each program are described in the individual files
in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
Last login: Mon Apr 2 19:05:10 2020 from host-5-58-88-92.la.net.ua
svystungmasenergy:~$ cd svystun/MRRRR
svystungmasenergy:~/ svystun/MRRRR$
svystungmasenergy:-/ svystun/MRRRR$ gcc svd_mrrr.cpp -o svd_mrrr
svystungmasenergy:~/ svystun/MRRRR$ svd_mrrr -numthreads 8
memory_used 128008000 bite
time 3956 ms
svystungmasenergy:~/ svystun/MRRRR$
```

Рисунок 4.1 – Спектральний розклад матриці на базі алгоритму MRRR для даних газоспоживання

На рисунку 4.2 показник пам’яті майже не відрізняється і це зрозуміло, адже розмірність матриці теж не змінюється, а от час виконання став трішки кращим. Пояснити це можна тим фактом, що, зазвичай технічні, не випадкові дані є більш дефляційованими, ніж реальні дані. Перший алгоритмом буде MRRR з бібліотеки SMP-MR[3]. На вхід будемо подавати реальний і випадковий часовий ряд. На вихід ми отримаємо спектральну декомпозицію траєкторної матриці, записану в файли, а також заміри часу і обсягу пам’яті, виведені на екран.

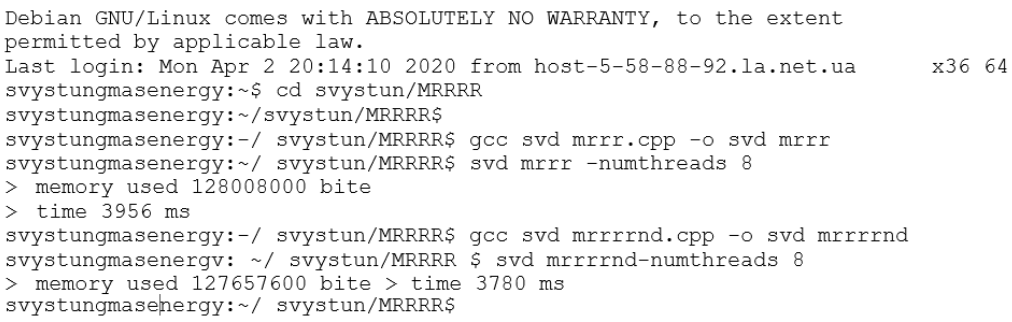


Рисунок 4.2 – Спектральний розклад матриці на базі алгоритму MRRR для випадкових даних

Наступним алгоритмом буде DC на базі бібліотеки плазма і паралельного середовища для роботи зі спільною пам’яттю QUARK. Вхідні і вихідні дані для нього будуть тими ж, що і для першого експерименту.

Наступним алгоритмом буде DC на базі бібліотеки плазма і паралельного середовища для роботи зі спільною пам’яттю QUARK. Вхідні і вихідні дані для нього будуть тими ж, що і для першого експерименту.

На рисунку 4.3 можна побачити, що, хоча пам'ять, яка необхідна для алгоритму є більшою, але час виконання все ж набагато менший. Витрачена пам’ять тут є несуттєвою, адже, незважаючи на різницю в майже 5 разів, все ж вони не досягають 50 Мб.

```
svystungmasenergv:~/svystun/DC$ gss svd dc.cpp -o svd dc
svystun@masenergv:~/svystun/DC$ svd dc -numthreads 8
> memory used 52409700 bite
> time 1850 ms
svystungmasenerav:~/svystun/DC$   gss   svd dcrnd.cpp   -o   svd dcrnd
svystun@masenergv: ~/svystun/DC$ svd dcrnd-numthreads 8
> memory used 52507600 bite
> time 2035 ms
svystun@masenergv:~/svystun/DC$ I
```

Рисунок 4.3 – Спектральний розклад матриці на базі алгоритму DC для випадкових даних

```
svystun@masenergy:~$ cd svystun/MRRRR
svystun@masenergy:~/svystun/MRRRR$
svystungmasenergy:~/svystun/MRRRR$  gcc  svd mrrr.cpp  -o  svd mrrr
svystungmasenergv:~/svystun/MRRRR$ svd mrrr -numthreads 8
>   memory used 12800800 bite
>   time 3956 ms
svystungmasenergv:~/svystun/MRRRR$ cd . .
svystungmasenergv:~/svystun$ cd DC
svystungmasenergv:~/svystun/DC$ gss svd dc.cpp -o svd dc
svystungmasenergv:~/svystun/DC$ svd dc -numthreads 8
>   memory used 52409700 bite
>   time 1850 ms
svystungmasenergy:~/svystun/DC$ I
```

Рисунок 4.4 – Спектральний розклад матриці на базі алгоритму DC для даних газоспоживання

Для часу ми спостерігаємо прогнозовану закономірність, що у випадках гіршої роботи MRRR, DC працює краще.

Швидкість роботи на випадкових даних є трошки гіршою, що теж власне закономірно і вказує на їх більш не дефляційний характер. Отже, ми підтвердили таким чином, що швидкодія обох алгоритмів залежність від характеру матриці.

Також було проведено експеримент залежності швидкодії алгоритму від розміру матриці. Результати відображені на рисунку 4.5.

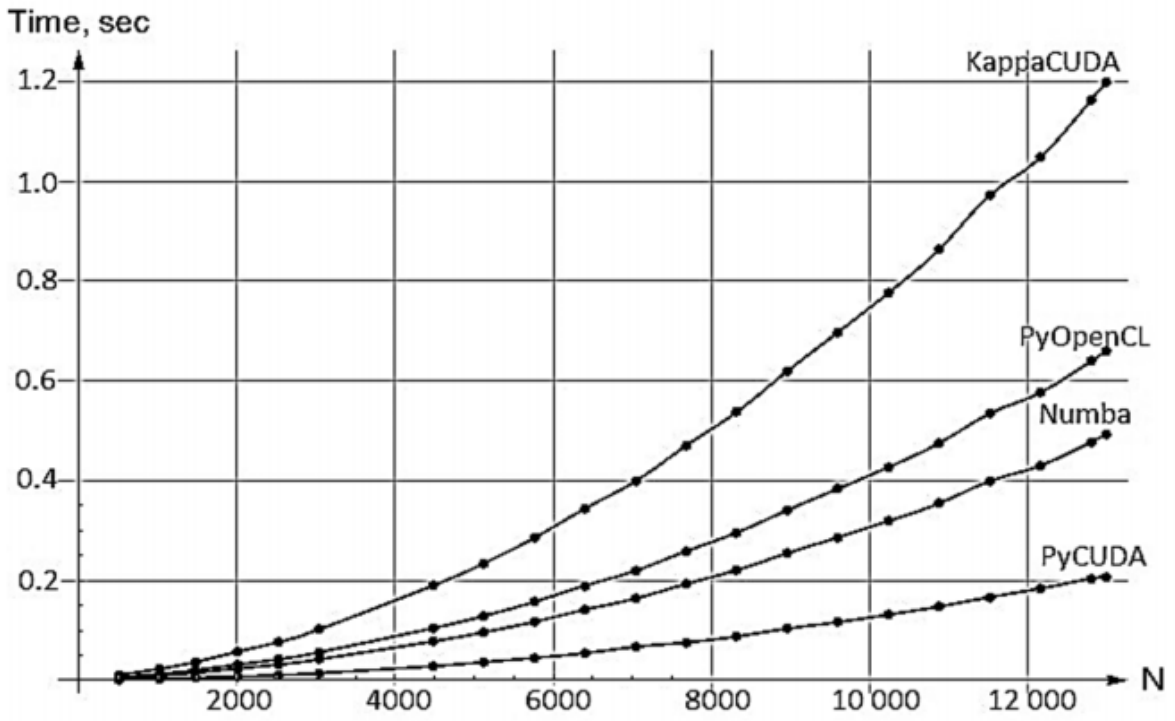


Рисунок 4.5 – Залежність часу роботи алгоритмів від розмірності матриці

Отже, як ми бачимо, алгоритм MRRR очікувано швидкий на матрицях малого і середнього розміру. Алгоритм DC є швидшим за нього практично в усіх випадках і зі збільшенням розміру матриці його швидкодія зростає. Варто додати, що в експерименті були використані ті ж дані газоспоживання.

4.2 Висновки до четвертого розділу

Отже, було представлено імплементацію алгоритмів MRRR і DC. Перший був спеціально розроблений для мультиядерний і багатопотокових архітектур. MR^3-SMP розбиває обчислення на задачі, які потім виконуються потоками паралельно. Задачі створюються і плануються динамічно. Хоча статичний поділ роботи супроводжується невеликими витратами, динамічний підхід забезпечує гнучкість і виробляє чудове балансування навантаження Для матриць малого і середнього розм іру MR^3-SMP є таким ж, а то й кращим рішенням, ніж найшвидша з існуючих паралельних бібліотек для знаходження власних чисел.

Також було розглянуто нову імплементацію знаменитого принципу «розділяй і владарюй», який використовує паралелізм на базі задач. Попередні експерименти показали, що ця реалізацію перевершує існуючу реалізацію DC з тим самим рівнем точності. Підхід на базі задач, який використовує середовище виконання QUARK, щоб планувати завдання, забезпечує високу швидкодію, як можна бачити на рис. 4.3 і 4.4. Крім того, експеримент, відображений на рис. 4.1 і 4.2, показує, що ця імплементація є конкурентною до найкращої з існуючих імплементацій MRRR для архітектур зі спільною пам’яттю Головним поліпшенням було впровадження більшої кількості паралелізму на кроці злиття, де операції є дорогими і потребую квадратно, а то й кубічної обчислювальної складності.

Новизна роботи полягала в тому, що експеримент проводився на актуальних реальних даних газоспоживання і його результати поможуть пришвидшити впровадження прогнозування споживання природніх ресурсів і значно спростити процес економії і експлуатації.

5 СПЕЦІАЛЬНА ЧАСТИНА

5.1 Орієнтований ациклічний граф

У математиці та інформатиці, орієнтований ациклічний граф (DAG), являє собою орієнтований граф без орієнтованих циклів. Тобто, він формується сукупністю вершин і орієнтованих ребер, кожне ребро з’єднує одну вершину з іншою, так що немає не можливо з деякої вершини V пройти деяку послідовність ребер і знов повернутися до V [49].

DAG можуть бути використані для моделювання багатьох різних видів інформації. Відношення досяжності в DAG утворює частковий порядок, а будь-яке кінцевий частковий порядок може бути представлений у вигляді DAG за допомогою відношення досяжності. Набір завдань, які повинні формувати послідовність, за умови, що деякі завдання повинні виконуватись раніше, ніж інші, можуть бути представлені у вигляді DAG з вершиною для кожного завдання і ребрами для кожного обмеження; алгоритми топологічного впорядкування можуть використовуватися для генерації валідної послідовності. Крім того, DAG може бути використаний як просторово-ефективне представлення набору послідовностей з підпослідовностями, які перекриваються. DAG також використовується для представлення системи подій або можливих подій і причинно-наслідкових зв'язків між ними. DAG також можуть бути використані для моделювання процесів, в яких потоки даних в рухаються через мережу процесорів, або станів сховища в системі керування версіями.

Відповідна концепція множини неорієнтованих графів називається лісом – неорієнтований граф без циклів. Вибір орієнтації для лісу виробляє

особливий вид спрямованого ациклічного графа, що називається орієнтованим деревом. Крім того, кожен неорієнтований граф має ациклічну орієнтацію, розподілення напрямку для його ребер, що робить його орієнтованим ациклічним графом.

DAG має наступні математичні властивості: доступність; транзитивне замикання; транзитивне скорочення.

Кожен орієнтований ациклічний граф породжує частковий порядок ⊆ для його вершин, де u ⊆ v, коли існує орієнтований шлях з u в v в DAG. Тим не менше, багато різних DAG можуть призвести до цього ж відношення досяжності співвідношенням. Наприклад, DAG з двома ребрами a → b і b → c має те ж відношення досяжності, що й граф з трьома ребрами a → b, b → c, a → c. Нехай G це DAG, його транзитивне скорочення це граф з найменшою кількістю ребер, що представляє те ж відношення досяжності, що і G, і його транзитивне замикання це граф з більшістю ребер, який представляє те ж відношення досяжності.

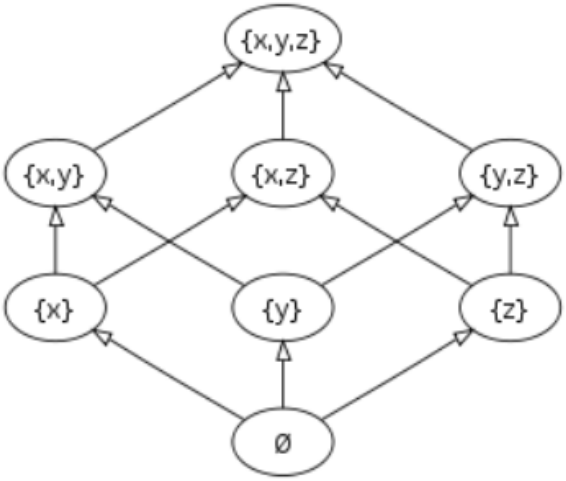


Рисунок 5.1 – Діаграма Гассе, яка представляє відношення часткового порядку серед множини підмножин елементів дерева

Транзитивне скорочення і транзитивне замикання однозначно визначені для DAG; на відміну від цього, для орієнтованого графа, що не є

ациклічним, не може бути більше, ніж одного мінімального підграфа з тим же відношенням досяжності.

Транзитивне замикання G має ребро u → v для зв’язаної кожної пари u ≤ v різних елементів в відношенні досяжності G, і, отже, може розглядатися як орієнтоване представлення відношення досяжності ⊆ в термінах теорії графів: кожна частково впорядкована множина можуть бути перетворена в DAG в цьому випадку. Якщо DAG G являє собою часткове відношення ⊆, то транзитивне скорочення G є підграфом G з ребром u → v для кожної пари часткового порядку ⊆; транзитивне скорочення корисне в візуалізації часткового порядку, тому що вони мають менше ребер, ніж інші графи, що представляють той самий порядок, і тому спрощують креслення графа. Діаграма Гассе, зображена на рисунку 5.1 часткового порядку являє собою креслення транзитивного скорочення, в якому орієнтація кожного ребра показана шляхом розміщення вихідної вершини ребра на нижчу, ніж кінцева вершина ребра, позицію.

5.2 Застосування орієнтованого ациклічного графа

Кожен орієнтований ациклічний граф має топологічне впорядкування вершин таке, що початкова вершина кожного ребра проходить раніше в сортуванні, ніж кінцева. Загалом, це сортування не є унікальним; DAG має унікальне топологічне упорядкування тоді і тільки тоді, якщо він має орієнтований шлях, що містить всі вершини, в цьому випадку сортування таке ж, як у порядку, в якому вершини з'являються на шляху.

Комбінаторне завдання перелічення графа для підрахунку спрямованих ациклічних графів вивчав Робінсон в 1973 р.. Кількість DAG на позначеному вузлі n, де n = 0, 1, 2, 3, ..., (ці цифри можуть з'являтися в будь-якому порядку в топологічної впорядкованості DAG) становить 1, 1, 3, 25,

543, 29281, 3781503, ... Це також можна обчислити за допомогою рекурентного співвідношення

$$a_n = \sum_{k=1}^{n} (-1)^{k-1} \binom{n}{k} 2^{k(n-k)} a_{n-k} \qquad (5.1)$$

Ерік У. Вайнштайн припустив, а Маккей в 2004 р. довели [60], що ті ж числа будуть отримані в результаті обчислення бінарної матриці, для якої всі «власні числа» є додатними і дійсними.

Орієнтоване дерево це спрямований граф, утворений шляхом орієнтації ребра вільного дерева. Кожне орієнтоване дерево є DAG. Зокрема, це відноситься і до деревоподібних структур, сформованих орієнтацією всіх ребер в напрямку від кореня дерева. Multi-tree-structured граф являє собою орієнтований граф, в якому існує не більше одного орієнтованого шляху (в обидва напрямки) між будь-якими двома вузлами; еквівалентом є DAG, в якому для кожного вузла v, множина вузлів, досяжних з v утворює дерево.

Є ряд задач, які можна вирішити за допомогою DAG, оскільки він грає ключову роль в алгоритмі розв’язку. В роботі розглянуто саме такі алгоритми і тому ці випадки потребують детальнішого аналізу.

Алгоритмічна задача знаходження топологічного сортування може бути вирішена за лінійний час. Алгоритм Кана для топологічного сортування знаходить порядок вершин безпосередньо, шляхом збереження списку вершин, які не мають ребер, що з'єднують їх з вершинами, які ще не були перераховані, і повторно додає одну з таких вершини до кінця списку, який будується. Крім того, топологічний порядок може бути побудований за допомогою алгоритму пошуку в глибину. Також можна перевірити, який із даних орієнтованих графів є DAG за лінійний час декількома способами: або спробувати знайти топологічне сортування, а потім перевірити для кожного ребра чи є результат дійсним, або, як альтернатива, для деяких топологічних

алгоритмів сортування, перевіривши, що алгоритм успішно сортує всі вершини не видаючи помилку умови.

Також складною задачею є побудова DAG. Будь-який неорієнтований граф може бути перетворений в DAG, якщо вибрати загальний порядок його вершин і орієнтацію кожного ребра з початкової кінцевої точки до більш пізньої кінцевої точки. Тим не менше, різні загальні порядки сортувань можуть привести до однієї ациклічної орієнтації. Кількість ациклічних орієнтацій рівна $|\mathrm{X}(-1)|$, де X є хроматичним многочленом даного графа.

Будь-який орієнтований граф може бути перетворений в DAG видаленням набору зворотних вершин або набору зворотних дуг. Тим не менш, знаходження найменшого такого набору це NP-складна задача. Довільний орієнтований граф може бути перетворений в DAG, це називається стисненням, шляхом перетворення кожного з його сильно зв’язних компонентів в одну супер вершину. Коли граф вже є ациклічним, його найменша множина зворотних вершин і множина зворотних дуг порожня, і його стиснення є самим графом.

Транзитивне замикання даного DAG, з n вершинами і m ребрами, може бути побудоване за час O(mn) за допомогою або пошуку в ширину, або пошуку в глибину, щоб перевірити досяжність з кожної вершини. Крім того, воно може бути вирішене за час $O(n^{\omega})$, де $\omega < 2{,}373$ є показником для алгоритмів швидкого множення матриць; це теоретичне поліпшення в порівнянні з граничним O(mn) для щільних графів.

У всіх цих алгоритмах транзитивного замикання можна виділити пари вершин, яких можна досягнути шляхом довжиною два або більше з пар, які можуть бути пов'язані шляхом одиничної довжини. Транзитивне скорочення складається з ребер, що утворюють шлях одиничної довжини, які є єдиними шляхами, що з'єднують їх кінцеві точки. Таким чином, транзитивне скорочення може бути побудоване в тих же межах асимптотичного часу, що і транзитивне замикання.

Задача замикання приймає в якості вхідних даних орієнтований ациклічний граф з вагами на його вершинах і шукає мінімальну (максимальну) вагу закриття набір вершин з яких не виходять ребра. ЇЇ можна вирішити за поліноміальний час, використовуючи скорочення задачі про максимальний потік.

Деякі алгоритми спрощуються при використанні DAG замість загальних графів, який базується на принципі топологічного сортування. Наприклад, можна знайти найкоротший шлях і найдовший шлях від деякої вершини DAG за лінійний час шляхом обробки вершин в топологічному порядку, і обчислення довжини шляху для кожної вершини, який буде мінімальною або максимальною довжиною, отримується за допомогою будь-якого з його вхідних ребер. На відміну від цього, для довільних графів знаходження найкоротшого шляху може вимагати повільніших алгоритмів, таких як алгоритм Дейкстри або алгоритм Беллмана-Форда, а знаходження найдовшого шляху в довільному графі має NP-складність.

Представлення DAG часткового порядку часто використовується в задачах планування для систем завдань з обмеженим впорядкуванням. Наприклад, DAG можуть бути використані для опису залежностей між клітинами таблиці: якщо одна комірка обчислюється за формулою з участю значення другої комірки, потрібно намалювати ребро DAG з другої комірки до першої. Якщо вхідні значення для таблиці змінюються, всі інші значення таблиці можуть бути перераховані за допомогою однієї оцінки на клітку, шляхом топологічного сортування клітинки і переоцінки кожної клітинки згідно обрахованого порядку. Аналогічні проблеми впорядкування завдань виникають в make-файлах для компіляції програми, планування інструкцій для низькорівневої оптимізації програми і планування технічного оцінювання та аналізу програми для управління великими проектами. Залежні графи без циклічних залежностей формують орієнтовані ациклічні графи.

Орієнтований граф може використовуватися для представлення мережі процесорних елементів; в цьому формулюванні, дані надходять в процесорний елемент через його ребро і виходять через його вихідне ребро.

Графіки, які мають вершини, що представляють події, і ребра, що представляють причинно-наслідкові зв’язки між подіями, часто ациклічні – приведення вершин до лінійного порядку часу, всі стрілки вказують у тому ж напрямку, що й час, від батьків до дітей, і таким чином не мають петель. Наприклад, баєсові мережі являють собою систему імовірнісних подій, представлених вузлами в орієнтованому ациклічному графі, в яких ймовірність події може бути обчислена з ймовірності своїх попередників у DAG [51]. У цьому контексті, «моральний граф» DAG є неорієнтованим графом, сформованим шляхом додавання ребер між усіма батьками того ж вузла, а послідовної заміни всіх орієнтованих ребер на неорієнтовані.

Генеалогічні дерева можуть також розглядатися як спрямовані ациклічні графи, з вершиною для кожного члена сім'ї і ребрами для кожного зв’язку батько-нащадок. Незважаючи на назву, ці графи не обов'язково є деревами через можливість «шлюбів» між далекими родичами (наприклад, нащадок має спільного предка зі сторони матері і батька), що спричиняє редукцію предків. Через те, що ніхто не може стати своїм же предком, ці графи є ациклічними. З тієї ж причини, історія версій розподіленої системи керування версіями, як правило, має структуру орієнтованого ациклічного графа, в якому є вершина для кожної перевірки і ребра, що з'єднують пари перевірок, які безпосередньо виплили одна з одної; загалом, через злиття, вони не є деревами.

У багатьох рандомізованих алгоритмах обчислювальної геометрії, алгоритм зберігає історію DAG, що представляє історію версій геометричних структур у послідовності змін у структурі. Наприклад, в рандомізованому зростанні алгоритму тріангуляції Делоне, зміни тріангуляції шляхом заміни одного трикутника трьома меншими трикутниками, коли кожна точка буде

додана, і операція «перевернути» замінить пари трикутників різними парами трикутників.

5.3 Висновки до п’ятого розділу

Отже, тип застосування спрямованих ациклічних графів виникає в стислому поданні безлічі послідовностей у вигляді шляхів в графі. Наприклад, орієнтований ациклічний граф слова являє собою структуру даних в інформатиці, утворену орієнтованим ациклічним графом з одним джерелом і з ребрами, поміченими літерами або символами; шлях від джерела до приймача в цьому графіку являє собою набір рядків, таких як англійські слова. Орієнтований ациклічний граф слова економить простір порівняно з префіксним деревом, дозволяючи шляхам розходитися і сходитися таким чином, що набір слів з тими ж ймовірними суфіксами може бути представлений в одному вузлі дерева.

Та ж ідея використання DAG для представлення сімейства шляхів, використана в бінарній діаграмі рішень, структури даних, які базуються на DAG використовуються для представлення двійкових функцій. У бінарній діаграмі рішень кожна вершин, яка не є приймачем, позначена ім'ям двійкової змінної, а кожен приймач і кожне ребро позначене 0 або 1. Значення функції для будь-якої підстановки змінної є значенням приймача, знайденим по шляху від однієї вершини джерела до кожної вершини не приймача по вихідному ребру, що має міткою значення змінної в початковій вершині. Так само, як спрямовані ациклічні графи слів можна розглядати як стислий вигляд префіксних дерев, бінарні діаграми рішень можна розглядати як стислі форми дерев рішень, які економлять простір, дозволяючи шляхам з’єднуватись, коли вони погоджуються з результатами всіх інших рішень.

6 ОБГРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

Метою цього розділу дипломної роботи є здійснення економічних розрахунків, спрямованих на визначення економічної ефективності проведення розпаралелювання алгоритму сингулярного-спектрального розкладу, а також прийняття рішення щодо його подальшого впровадження.

Для здійснення оцінки потрібно зробити розрахунки трудомісткості кожної операції, що мала місце при проведенні дослідження.

6.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

Витрати часу по окремих операціях технологічного процесу відображені в таблиці 6.1.

Таблиця 6.1 – Операції технологічного процесу та час їх виконання

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1.	Витрати праці на опис задачі	інженер	16
2.	Витрати праці на дослідження алгоритмів для реалізації SVD	інженер	10
3.	Витрати праці на дослідження технік паралелізму	інженер	16
4.	Витрати праці на розробку алгоритму, який буде обраховувати SVD паралельно	інженер	45
5.	Витрати праці на підготовку документації	інженер	40
6.	Витрати праці на відлагодження програми на ЕОМ	інженер	40
Р а з о м			**167**

Отже, витрати часу по окремих операціях технологічного процесу становлять 167 год.

6.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Відповідно до Закону України «Про оплату праці» заробітна плата – це «винагорода, обчислена, як правило, у грошовому виразі, яку власник або уповноважений ним орган виплачує працівникові за виконану ним роботу».

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його праці та господарської діяльності підприємства. Заробітна плата складається з основної та додаткової оплати праці.

Основними ставками, відрядними розцінками чи посадовими окладами і не залежить від результатів господарської діяльності підприємства.

Додаткова заробітна плата – це складова заробітної плати працівників, до якої включають витрати на оплату праці, не пов’язані з виплатами за фактично відпрацьований час. Нараховують додаткову заробітну плату залежно від досягнутих і запланованих показників, умов виробництва, кваліфікації виконавців. Джерелом додаткової оплати праці є фонд матеріального стимулювання, який створюється за рахунок прибутку.

При розрахунку заробітної плати кількість робочих днів у місяці слід в середньому приймати – 20,8 дні/міс., або ж 167 год./міс. (тривалість робочого дня – 8 год.).

Місячний оклад кожного працівника слід враховувати згідно існуючих на даний час тарифних окладів. Рекомендовані тарифні ставки: керівник дослідження – 15,0...18,0 грн./год., інженер – 9,0...14,0 грн./год.,

консультант – 8,0...13,0 грн./год., технік – 7,0...10,0 грн./год., лаборант з вищою освітою – 7,0...10,0 грн./год.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = Т_с \cdot К_г, \quad (6.1)$$

де $Т_с$ – тарифна ставка, грн., $К_г$ – кількість відпрацьованих годин.

Оскільки всі види робіт в даній роботі виконує інженер, то основна заробітна плата буде рівної.

$$З_{осн.} = 12 \cdot 167 = 2004 \text{ грн.}$$

Додаткова заробітна плата становить 10–15% від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{дод.}, \quad (6.2)$$

де $К_{дод.}$ – коефіцієнт додаткових виплат працівникам, 0,1–0,15 (візьмемо його рівним 0,15).

$$З_{дод.} = 2004 \cdot 0,15 = 300,6 \text{ грн.}$$

Звідси загальні витрати на оплату праці ($В_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} \cdot З_{дод.} \quad (6.3)$$

$$В_{о.п.} = 2004 + 300,6 = 2304,6 \text{ грн.}$$

Крім того, слід визначити відрахування на соціальні заходи: фонд страхування на випадок безробіття – 1,3%; фонд по тимчасовій втраті працездатності – 2,9%; пенсійний фонд – 32,3%.

У сумі зазначені відрахування становлять 37,5%.

Отже, сума відрахувань на соціальні заходи буде становити:

$$В_{с.з.} = ФОП \cdot 0,375, \quad (6.4)$$

де *ФОП* – фонд оплати праці, грн.

$$В_{с.з.} = 2304,6 \cdot 0,375 = 864,23 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці зведемо у таблицю 6.2.

Таблиця 6.2 – Зведені розрахунки витрат на оплату праці

№ п/п	**Категорія працівни к-ків**	**Основна заробітна плата, грн.**			**Додаткова заробітна плата, грн.**	**Нарахув. на ФОП, грн.**	**Всього витрати на оплату праці, грн. *6=3+4+5***
		Тарифна ставка, грн.	**К-сть відпра-цьов. год.**	**Фактично нарах. з/пл., грн.**			
А	*Б*	*1*	*2*	*3*	*4*	*5*	*6*
1	інженер	12	167	2004	300,6	864,23	3168,83

Отже, витрати на оплату праці становлять 3168,83 грн.

6.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$M_{Bi} = q_i \cdot p_i, \quad (6.5)$$

де q_i – кількість витраченого матеріалу *i*-го виду, p_i– ціна матеріалу i-го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$З_{м.в.} = \sum M_{B\,i} \quad (6.6)$$

Проведені розрахунки занесемо у таблицю 6.3.

Таблиця 6.3 – Зведені розрахунки матеріальних витрат

№ п/п	Найменування матеріальних ресурсів	Од. виміру	Факт. витрачено матеріалів	Ціна 1-ці, грн.	Загальна сума витрат, грн.
1	Папір формату А4	шт.	200	0,12	24
2	Папір формату А1	шт.	10	4	40
3	Інструменти для малювання та креслення	шт.	2	5	10
Р а з о м			**212**	**3,04**	**74**

Отже, матеріальні витрати становлять 74 грн.

6.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S, \quad (6.7)$$

де W – необхідна потужність, кВт, T – кількість годин роботи обладнання, S – вартість кіловат-години електроенергії.

Вартість кіловат-години електроенергії слід приймати згідно існуючих на даний час тарифів (0,456 грн. + 20% ПДВ за 1 кВт). Отже, 1 кВт з ПДВ коштує 0,5472 грн.

Для цього дослідження потрібно використати два комп'ютери. Перший при розробці і написанні алгоритму, другий – для, власне, виконання і отримання необхідних результатів. Потужність першого і другого рівна і становить– 250 Вт, кількість годин роботи обладнання згідно таблиці 6.1 – 167 годин.

Тоді

$$З_е = 0{,}25 \cdot 167 \cdot 0{,}5472 = 22{,}85 \text{ грн.}$$

6.5 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення.

Комп’ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60% (квартальна – 15%).

Для визначення амортизаційних відрахувань застосовуємо формулу:

$$А = Б_В \cdot Н_А/100\%, \quad (6.8)$$

де $А$ – амортизаційні відрахування за звітний період, грн., $Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн., $Н_А$ – норма амортизації, %.

Для цього дослідження засобами виконання будуть два комп’ютери. Вартість першого становить 7000 грн, вартість другого – 40000 грн. Оскільки дослідження буде виконуватись на різних машинах, то амортизаційні відрахування для першої будуть рівні:

$$А_1 = (7000 \cdot 5\%)/100\% = 350 \text{ грн.}$$

А для сервера становитимуть:

$$А_2 = (4000 \cdot 5\%)/100\% = 200 \text{ грн.}$$

Робота на першому комп’ютері виконувалась 127 годин, а на другому – 40 годин, то амортизаційні відрахування будуть становити:

$$А = ((350 \cdot 127) + (2000 \cdot 40))/150\% = 829{,}67 \text{ грн.}$$

6.6 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління спілкою та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60% від суми основної та додаткової заробітної плати працівників.

$$Н_в = В_{о.п.} \cdot 0,2 \dots 0,6, \quad (6.9)$$

де $Н_в$ – накладні витрати.

Отже, накладні витрати:

$$Н_в = 2304,6 \cdot 0,2 = 460,92 \text{ грн.}$$

6.7 Складання кошторису витрат та визначення собівартості НДР

Результати проведених вище розрахунків зведемо у таблицю 6.4.

Таблиця 6.4 – Кошторис витрат на НДР

Зміст витрат	**Сума, грн.**	**В % до загальної суми**
Витрати на оплату праці (основну і додаткову заробітну плату)	2304,6	50,59
Відрахування на соціальні заходи	864,23	18,97
Матеріальні витрати	74	1,62
Витрати на електроенергію	22,85	0,5
Амортизаційні відрахування	829,67	18,2

Продовження таблиці 6.4

Накладні витрати	460,92	10,12
Собівартість	**4556,27**	**100**

Собівартість ($С_в$) проведення дослідження розрахуємо за формулою:

$$С_в = В_{о.п.} + В_{с.з.} + З_{мв.} + З_в + А + Н_в \quad (6.10)$$

Отже, собівартість дослідження дорівнює:

$$С_в = 2304{,}6 + 846{,}23 + 74 + 22{,}85 + 829{,}67 + 460{,}92 = 4556{,}27 \text{ грн.}$$

6.8 Розрахунок ціни дослідження

Ціну НДР можна визначити за формулою:

$$Ц = (С_в \cdot К(1 + Р_{рен}) + К \cdot В_{н.і.})/К \cdot (1 + ПДВ), \quad (6.11)$$

де $Р_{рен}$ – рівень рентабельності, 30%, $К$ – кількість замовлень, од. (встановлюється лише при розробці програми та мікропроцесорних систем), $В_{н.і.}$ – вартість носія інформації, грн. (встановлюється лише при розробці програми), $ПДВ$ – ставка податку на додану вартість, (20%).

Звідси ціна на дослідження складе:

$$Ц = ((4556{,}27 \cdot 10 \cdot (1 + 0{,}3) + 10 \cdot 3)/10 \cdot (1 + 0{,}2) = 7111{,}38 \text{ грн.}$$

6.9 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Економічна ефективність ($Е_р$) полягає у відношенні результату виробництва до затрачених ресурсів:

$$Е_р = П/С_в, \quad (6.12)$$

де $П$ – прибуток;

$С_в$ – собівартість.

Плановий прибуток ($П_{пл}$) знаходимо за формулою:

$$П_{пл} = Ц - С_в \quad (6.13)$$

Розраховуємо плановий прибуток:

$$П_{пл} = 7111{,}38 - 4556{,}27 = 2555{,}11 \text{ грн.}$$

Тоді,

$$Е_р = 2555{,}11/4556{,}27 = 0{,}56.$$

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень ($Т_р$):

$$T_р = 1/E_р \quad (6.14)$$

Термін окупності дорівнює:

$$T_р = 1/0{,}56 = 1{,}8 \text{ років.}$$

6.10 Висновки до шостого розділу

В розділі економічного обґрунтування ефективності дипломної роботи розраховано основні техніко-економічні показники проведення розпаралелювання алгоритму сингулярного-спектрального розкладу (див. таблицю 6.5).

Розраховане значення економічної ефективності становить 0,56, що є високим значенням.

Так само прийнятним є термін окупності. Для проведення цього дослідження він становить 1,8 років.

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

7.1 Охорона праці

7.1.2Вимоги до природного та штучного освітлення робочих місць користувачів ПК

Всеохоплююча інформатизація суспільства як глобальний, загальносвітовий процес та багатопланові зміни в організації праці спонукає багатьох людей використовувати різноманітні засоби електронно-обчислювальної (комп'ютерної) техніки та опановувати відповідні інформаційні технології. При цьому, упровадження комп'ютерних засобів в усі сфери життєдіяльності людини виявило не тільки позитивні, а й негативні наслідки їх використання, про що свідчать скарги користувачів комп'ютера. Зокрема, щодо погіршення власного здоров'я та зниження функціональності організму через неправильне освітлення.

Правильно спроектоване і виконане виробниче освітлення покращує умови зорової роботи, знижує стомлюваність, сприяє підвищенню продуктивності праці, благотворно впливає на виробниче середовище, надаючи позитивну психологічну дію на працюючу, підвищує безпеку праці і знижує травматизм.

Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. неправильний напрям світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть привести до нещасного випадку або профзахворювань, тому такий важливий правильний розрахунок освітленості.

Існує три види освітлення - природне, штучне і комбіноване (змішане, тобто природне і штучне разом).

При виконанні робіт категорії високої зорової точності (найменший розмір об’єкту розрізнення 0,3 - 0,5мм) величина коефіцієнта природного освітлення (КПО) повинна бути не нижчою 1,5%, а при зоровій роботі середньої точності (якнайменший розмір об’єкту розрізнення 0,5 - 1,0 мм) КПО повинен бути не нижчим 1,0%. У якості джерела штучного освітлення звичайно використовуються люмінесцентні лампи типу ЛБ, або ДРЛ, які попарно об’єднуються в світильники, які повинні розташовуватися рівномірно над робочими поверхнями.

Джерела штучного світла рекомендується розташувати з обох сторін від екрану паралельно напрямку зору. Щоб уникнути світових блисків в напрямку очей необхідно застосовувати антиблискові сітки, спеціальні фільтри для екрану.

Необхідно використовувати систему вимикачів, що дозволяє регулювати інтенсивність штучного освітлення залежно від інтенсивності природного, а також дозволяє освітлювати тільки потрібні для роботи зони приміщення. Для забезпечення нормованих значень освітлення необхідно очищати віконне скло та світильники не рідше ніж 2 рази на рік, та своєчасно проводити заміну ламп, що перегоріли [58].

Як відомо, тривала робота за комп’ютером та з документами при недостатньому рівні освітленості може призвести до значного перенапруження зору, тому вимоги до освітлення є досить важливими.

Вимоги до освітлення встановлено ДБН В.2.5-28-2006 «Природне і штучне освітлення», затвердженими наказом Мінрегіону від 15.05.2006 р. № 168.

Відносно вікон робоче місце необхідно організовувати так, щоб природне світло було з лівого боку (п. 4.3 ДСанПіН 3.3.2.007-98). Робоче місце необхідно розміщувати таким чином, щоб уникнути попадання прямого світла в очі. Для забезпечення захисту і досягнення нормованих рівнів комп’ютерних випромінювань необхідне застосування приекранних

фільтрів, локальних світлофільтрів (засобів індивідуального захисту очей) та інших засобів захисту, що пройшли випробування в акредитованих лабораторіях і мають щорічний гігієнічний сертифікат (п. 4.19 ДСанПіН 3.3.2.007-98).

Штучне освітлення приміщення має здійснюватись системою загального рівномірного освітлення (п. 3.2.2 ДСанПіН 3.3.2.007-98). У приміщеннях при переважній роботі з документами допускається використання системи комбінованого освітлення, тобто встановлення світильників місцевого освітлення додатково до загального.

Як джерела штучного освітлення необхідно використовувати люмінесцентні лампи. Згідно з п. 3.2.5 ДСанПіН 3.3.2.007-98 система загального освітлення має бути у вигляді суцільних або переривчатих ліній світильників, що розташовані збоку від робочих місць (зазвичай ліворуч) паралельно лінії зору працівників.

Допускається застосування ламп розжарювання у світильниках місцевого освітлення та, у разі влаштування відбитого освітлення у виробничих чи адміністративно-громадських приміщеннях, металогалогенних ламп потужністю 250 Вт.

Коефіцієнт пульсації не повинен перевищувати 5% (п. 3.2.14 ДСанПіН 3.3.2.007-98). Рівень освітленості на робочому столі в зоні розташування документів має бути в межах 300–500 лк. Світильники місцевого освітлення слід встановлювати таким чином, щоб не створювати відблисків на поверхні екрана, а освітленість екрана має не перевищувати 300 лк.

Для забезпечення нормованих значень освітленості у приміщеннях відповідно до п. 3.2.15 ДСанПіН 3.3.2.007-98 необхідно мити вікна і світильники не рідше 2 разів на рік, а також своєчасно замінювати лампи, що перегоріли [59].

Колір приміщень і меблів повинен сприяти створенню сприятливих умов для зорового сприйняття та гарного настрою.

Джерела світла, такі як світильники і вікна, які дають віддзеркалення від поверхні екрану, значно погіршують точність сприйняття знаків на екрані монітору чи клавіатури і спричиняють за собою перешкоди фізіологічного характеру, які можуть виразитися в значній напрузі, особливо при тривалій роботі. Віддзеркалення, включаючи віддзеркалення від вторинних джерел світла, повинне бути зведено до мінімуму. Для захисту від надмірної яскравості вікон можуть бути застосовані штори і екрани.

Виробниче середовище, як вже було зазначено, суттєво впливає на працездатність людини в системі, тому дизайн приміщень, інших елементів середовища є особливо важливим.

Зважаючи на те, що більшість систем пов’язана з виконанням людиною розумової праці, розглянемо дизайн офісу як елемента середовища. Офіс, робоче приміщення, призначене для виконання працівниками певних функцій (трудової діяльності) з максимальною повнотою й у визначені строки. Офіс, як і промислове приміщення, створюється за участю технологів, фахівців, які працюватимуть у ньому, щоб максимально наблизити дизайн-проект до чітких технологічних вимог.

Незважаючи на специфікацію, багатоманітність функцій, виокремлюють дві основні групи офісної роботи: концентрація та спілкування.

Залежно від основної специфікації офісу співвідношення цих категорій змінюється, а пошук оптимального освітлення, щоб урівноважити спілкування та концентрацію — головне завдання дизайн-проекту зовнішнього середовища.

7.1.2 Шкідливі фактори при роботі з комп’ютерною технікою

Працівники, задіяні на роботах, пов'язаних з періодичною або постійною роботою за комп'ютером, піддаються впливу факторів виробничої небезпеки, основними з яких є:

1. Фізичні.

– Підвищений рівень напруги в електричному ланцюзі, замикання якої може пройти через тіло працюючого.

– Підвищений рівень рентгенівського випромінювання.

– Підвищений рівень ультрафіолетового випромінювання.

– Підвищений рівень інфрачервоного випромінювання.

– Можливість ураження статичною електрикою.

– Запиленість повітря робочого приміщення.

– Підвищений вміст важких (+) аероіонів.

– Нерівномірний розподіл яскравості в полі зору.

– Підвищений рівень пульсації світлового потоку.

2. Хімічні. - Підвищений вміст у повітрі вуглекислого газу, озону, аміаку, фенолу, формальдегіду та ін.

3. Психофізіологічні.

– Напруга зору.

– Напруга пам'яті.

– Напруга уваги.

– Тривале статичне напруження.

– Відносно великий обсяг інформації, що обробляється в одиницю часу.

– Монотонність праці в окремих випадках.

– Нераціональна організація робочого місця.

До основних шкідливих факторів при роботі з комп'ютером відносять: тривале сидяче положення, електромагнітне випромінювання, навантаження на зір, перевантаження кистьових суглобів, можливість захворювань органів дихання, алергії, порушення нормального перебігу вагітності та ін.

Тривале сидяче положення приводить до напруги м'язів шиї, голови, рук і плечей, остеохондрозу, у дітей - ще й до сколіозу. Тривале сидяче положення ще приводить до застою крові в тазових органах і, як наслідок, до

простатиту й геморою. Не секрет, що малорухливий спосіб життя призводить до ожиріння. Остеохондроз виникає при порушенні міжхребцевих дисків, яке призводить до випинання в яку або сторону (грижі міжхребцевого диска). Грижа може зашкодить спинний мозок і нервові відростки. Наслідки можуть бути найрізноманітнішими, від болів в спині і кінцівках, до паралічу кінцівок і смерті. Одна з поширених причин остеохондрозу - дистрофія м'язів спини. Людина, провідний в основному сидячий спосіб життя, цілком може захворіти остеохондрозом. Ознаки початку захворювання: дискомфорт у спині та больові відчуття, головні болі, порушення роботи внутрішніх органів.

До факторів ризику захворювання гемороєм відносять: сидячий спосіб життя, ожиріння, надмірне вживання копчених, гострих, солоних і пряних продуктів, запальні захворювання малого таза та ін. Ожиріння виникає через нераціональне харчування, малорухомого і в тому числі сидячого способу життя, неадекватної реакції на стресові ситуації, надмірно довгий сон, застосування гормональних препаратів, перевантаження організму харчовими жирами і ін. Ожиріння призводить до збільшення навантаження на серце, зміни конфігурації та положення серця в грудній порожнині, підвищення вмісту холестерину в крові, в результаті він відкладається на стінках судин (атеросклероз). Підвищений скупчення жиру всередині грудної порожнини впливає на роботу органів дихання, що призводить до появи задишки та гіпоксії органів і тканин.

Навантаження на зір. Людське око реагує на найдрібнішу вібрацію тексту і на мерехтіння екрану. М'язи ока, керуючі кришталиком, перебувають у постійній напрузі, що обов'язково призводить до втрати гостроти зору. Немаловажне значення для профілактики зорових дисфункцій надають: правильний чи рекомендований підбір кольору, шрифтів, компоновки вікон у використовуваних додатках, орієнтація дисплея монітора. Тривала робота за комп'ютером - це величезне навантаження на очі, оскільки зображення на

моніторі складається не з безперервних ліній, як на папері, а з окремих точок, які світяться і мерехтять. У користувача неминуче погіршується зір, очі починають сльозитися, з'являється головний біль, втома, зображення двоїться і спотворюється.

Перевантаження суглобів кистей рук приводить головним чином до такого явища, як синдром зап'ястного каналу.

Робота за комп'ютером і стреси. Стрес - це емоційні переживання, внутрішнє напруження, викликані подіями в житті. Стрес виникає, в першу чергу, при втраті або пошкодженні інформації. Причини: відсутність резервних копій, комп'ютерні віруси, поломки жорстких дисків, робочі помилки. Іноді стреси є причиною інфарктів. Стреси бувають емоційно позитивними і емоційно негативними, короткочасними і довгостроковими, гострими і хронічними, фізіологічними і психологічними (інформаційними й емоційними). Робота за комп'ютером є одним з факторів, що викликають стрес (стресором). Реакція організму на стрес являє собою запуск біохімічних процесів, які спрямовані на придушення екстремальній ситуації. Стресові ситуації і пов'язані з ними переживання викликають в організмі численні негативні зрушення.

Робота за комп'ютером і органи дихання. Захворювання органів дихання у даному контексті носять в основному алергічний характер. Це пояснюється тим, що за час довгої роботи комп'ютера корпус і плати останнього виділяють в повітря ряд шкідливих речовин, а так само комп'ютер створює навколо себе електростатичне поле, яке притягує пил, який осідає в легенях. Так же комп'ютер деіонізірует навколишнє середовище і зменшує вологість повітря. *Алергія* - це підвищена чутливість організму до різних подразників, що виявляється в специфічних реакціях при контакті з ними. Це викликає такі симптоми алергії як риніт, сльозоточивість, шкірний висип, анафілактичний шок. Комп'ютер є досить серйозним джерелом низки алергенів. Приміром, корпус монітора, нагріваючись до 50-55 °С починає

виділяти в повітря пари тріфенілфосфата. Крім монітора нагрівається і материнська плата, блок живлення, процесор, відеокарта, які так само можуть виділяти в навколишнє середовище шкідливі органічні та неорганічні речовини (фтор-, хлор-, фосфоровмісні). Крім того, в комп'ютері є дуже багато місць, де накопичується пил і бруд, розмножуються мікроби і грибки. Пил отримує від екрану монітора слабкий статичний заряд, якого вистачає, що б пил осідав на тілі користувача і в його дихальних шляхах. При алергії підвищується стомлюваність, посилюється дратівливість і знижується імунітет. Алергія провокує ряд захворювань: екзему, гемолітичну анемію, бронхіальну астму та ін. Найбільш тяжким проявом алергії є анафілактичний шок, який супроводжується утрудненням дихання, судомами, втратою свідомості, зниженням артеріального тиску і часто смерті.

7.2 Безпека в надзвичайних ситуаціях

7.2.1 Основні принципи і способи забезпечення життєдіяльності

Проблема захисту людей від небезпек у різних умовах їх існування виникла за сивої давнини. Уже тоді людина побоювалась різних невідомих процесів і явищ, загроз з боку представників біологічного світу. З розвитком людського суспільства з’являються додаткові небезпеки, причиною яких стала сама людина. Науково-технічний прогрес, у цілому збільшуючи безпеку життєдіяльності людини, підвищуючи комфортність життя, спричинив появу низки нових проблем.

По-перше, це надзвичайне зростання ризику ураження та загибелі людей у разі взаємодії зі складними технічними системами на виробництві, транспорті, у побуті.

По-друге, збільшується кількість аварій і катастроф, зумовлених зниженням надійності пристроїв, зроблених людиною, та помилками персоналу під час їх експлуатації, внаслідок яких створюються надзвичайні

ситуації, які призводять до загибелі людей, великих матеріальних втрат і забруднення природного середовища.

По-третє, зростає кількість стихійних лих, а також збройних конфліктів і терористичних дій, та розміри негативних наслідків від них.

По-четверте, ліквідація наслідків катастрофічних аварій, стихійних лих, збройних конфліктів потребує величезних зусиль і матеріальних витрат під час проведення аварійно-рятувальних і відновлювальних робіт, забезпечення життєдіяльності постраждалого населення у районах надзвичайних ситуацій. Усе це призводить до того, що вже зараз цивілізація несе величезні, іноді непоправні втрати.

До основних принципів забезпечення життєдіяльності відносяться:

1. Безперервне забезпечення фізіологічних процесів організму людини, що залежить від таких факторів: повітря; питна вода; продукти харчування; предмети середовища мешкання: житло, тепло; світло; електроенергія; предмети споживання (одяг, взуття та ін.).

2. Принципи взаємозв’язку і взаємозалежності людини з навколишнім середовищем. Життєдіяльність людини забезпечується навколишнім середовищем: енергоресурсами, корисними копалинами, продуктами харчування, ресурсами інформаційної сфери та іншими потоками навколишнього середовища. Характер взаємодії людини з навколишнім середовищем визначаються значенням показників потоків речовини, енергії та інформації. Гармонійна взаємодія людини і навколишнього середовища відбувається лише в умовах, коли потоки енергії, речовини та інформації знаходяться в допустимих межах і сприятливо сприймаються людиною і природним середовищем. Будь-яке перевищення допустимих рівнів зазначених потоків може супроводжуватись виникненням небезпек і їх впливом на життєдіяльність людини.

3. Принцип системності, який відображає універсальний закон діалектики про взаємний зв’язок явищ і подій. Згідно з цим принципом

необхідно розглядати явища із системних концепцій в їх взаємному зв'язку і цілісності. Принцип системності орієнтує на облік всіх елементів, що формують розглянутий результат, на повний облік обставин і чинників для забезпечення безпеки життєдіяльності. До елементів системи відносяться матеріальні об'єкти, а також відносини і зв'язки, що існують між ними.

4. Принцип захисту здоров'я, меж і умов життєдіяльності. Для реалізації цього принципу людство створило спеціальні інститути медичного забезпечення, оборони, екологічного захисту, моралі та ін. Окремі інститути як структурні частини життєдіяльності можуть створюватись для захисту людей і навколишнього середовища. До них можна віднести: систему охорони здоров'я, водних і повітряних ресурсів тощо.

5. Принцип запобігання і ліквідації негативних наслідків життєдіяльності людини супроводжується надзвичайними ситуаціями різного характеру і рівня. Тому необхідні спеціальні інститути як структурні елементи забезпечення життєдіяльності в особливих (надзвичайних) ситуаціях. До них можна зарахувати: Міністерство з надзвичайних ситуацій, комісії з питань техногенно-екологічної безпеки і надзвичайних ситуацій, штаби цивільної оборони.

6. Принцип деструкції полягає в тому, що система, яка веде до небезпечного результату, руйнується за рахунок виключення з неї однієї чи декількох елементів. Принцип деструкції органічно пов'язаний з розглянутим принципом системності та має настільки ж універсальне значення.

7. Принцип нормування полягає в регламентації умов, дотримання яких забезпечує заданий рівень безпеки. Необхідність нормування обумовлюється тим, що досягти абсолютну безпеку практично неможливо. Нормування має важливе методологічне значення. Норми є вихідними даними для розрахунку та організації заходів щодо забезпечення безпеки. При нормуванні враховуються психофізичні характеристики людини, а також технічні та економічні можливості.

8. Принцип несумісності полягає в просторовому і тимчасовому поділі об’єктів реального світу (речовин, матеріалів, обладнання, приміщень, людей), заснованому на обліку природи їх взаємодії з позицій безпеки. Такий поділ має на меті виключити виникнення небезпечних ситуацій, породжуваних взаємодією об’єктів. Цей принцип досить поширений у різних сферах техніки.

9. Принцип ергономічності полягає в тому, що для забезпечення безпеки враховуються антропометричні, психофізіологічні та психологічні властивості людини.

У загальному випадку забезпечення безпеки життєдіяльності може мати три виміри:

– за видом діяльності: нормативно-правовий, організаційно-технічний, виконавчий, науково-навчальний;

– за напрямом діяльності: охорона життя та здоров’я людини, захист територій і населення від надзвичайних ситуацій, охорона навколишнього середовища;

– за рівнями діяльності: загальнодержавний, регіональний, місцевий, об’єктовий.

Політика держави щодо вирішення проблем забезпечення безпеки життєдіяльності знаходить своє відображення в чинному законодавстві України. Ці питання прямо чи опосередковано регулюють понад 200 нормативно-правових актів різної юридичної сили. Основні правові засади забезпечення невід’ємних прав людини щодо забезпечення безпеки її життєдіяльності викладено в Конституції України, яка гарантує встановлення безпеки на рівні національної ідеї. Основними законами, які визначають правові та організаційні засади захисту громадян України від надзвичайних ситуацій і забезпечення безпеки їх життєдіяльності є закони України: «Про захист населення і територій від надзвичайних ситуацій»; «Про правові засади цивільного захисту»; «Про цивільну оборону України»; «Про

аварійнорятувальні служби»; «Про зону надзвичайної екологічної ситуації»; «Про правовий режим надзвичайного стану»; «Про Збройні Сили України» тощо.

7.2.2 Освітлення виробничих приміщень для роботи з ВДТ та локальній комп'ютерній мережі

Приміщення для роботи з ВДТ повинні мати природне та штучне освітлення відповідно до ДБН В.2.5-28-2006 (На заміну СНиП II-4-79).

Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і забезпечувати коефіцієнт природною освітленості (КПО) не нижче ніж 1,5%. Розраховується КПО за методикою, викладеною в ДБН В.2.5-28-2006.

За виробничої потреби дозволяється експлуатувати ЕОМ у приміщеннях без природного освітлення за узгодженням з органами державного нагляду за охороною праці та органами і установами санітарно-епідеміологічної служби.

Вікна приміщень з ВДТ повинні мати регулювальні пристрої для відкривання, а також жалюзі, штори, зовнішні козирки тощо.

Штучне освітлення приміщення з робочими місцями, обладнаними ВДТ ЕОМ загального та персонального користування, має бути обладнане системою загального рівномірного освітлення. У виробничих та адміністративно-громадських приміщеннях, де переважають роботи з документами, допускається вживати систему комбінованого освітлення (додатково до загального освітлення встановлюються світильники місцевого освітлення).

Загальне освітлення має бути виконане у вигляді суцільних або переривчатих ліній світильників, що розміщуються збоку від робочих місць (переважно зліва) паралельно лінії зору працівників. Допускається застосовувати світильники таких класів світлорозподілу:

– світильники прямого світла – П;

– переважно прямого світла – Н;

– переважно відбитого світла – В.

При розташуванні відеотерміналів ЕОМ за периметром приміщення лінії світильників штучного освітлення повинні розміщуватися локально над робочими місцями.

Для загального освітлення необхідно застосовувати світильники із розсіювачами та дзеркальними екранними сітками або віддзеркалювачами, укомплектовані високочастотними пускорегулювальними апаратами (ВЧ ПРА). Допускається застосовувати світильники без ВЧ ПРА тільки при використанні моделі з технічною назвою "Кососвет". Застосування світильників без розсіювачів та екранних сіток забороняється.

Як джерело світла при штучному освітленні повинні застосовуватися, як правило, люмінесцентні лампи типу ЛБ. При обладнанні відбивного освітлення у виробничих та адміністративно-громадських приміщеннях можуть застосовуватися металогалогенні лампи потужністю до 250 Вт. Допускається у світильниках місцевого освітлення застосовувати лампи розжарювання.

Яскравість світильників загального освітлення в зоні кутів випромінювання від 50º до 90º відносно вертикалі в подовжній і поперечній площинах повинна складати не більше 200 кд/м^2, а захисний кут світильників повинен бути не більшим за 40º.

Коефіцієнт запасу ($K_з$) відповідно до ДБН В.2.5-28-2006 для освітлювальної установки загального освітлення слід приймати рівним 1,4.

Коефіцієнт пульсації повинен не перевищувати 5 % і забезпечуватися застосуванням газорозрядних ламп у світильниках загального і місцевого освітлення.

За відсутності світильників з ВЧ ПРА лампи багатолампових світильників або розташовані поруч світильники загального освітлення необхідно підключати до різних фаз трифазної мережі.

Рівень освітленості на робочому столі в зоні розташування документів має бути в межах 300…500 лк. У разі неможливості забезпечити даний рівень освітленості системою загального освітлення допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрану та збільшення освітленості екрану більше ніж до 300 лк.

Світильники місцевого освітлення повинні мати напівпрозорий відбивач світла з захисним кутом не меншим за 40º.

Необхідно передбачити обмеження прямої блискості від джерела природного та штучного освітлення, при цьому яскравість поверхонь, що світяться (вікна, джерела штучного світла) і перебувають у полі зору, повинна бути не більшою за 200 кд/м^2.

Необхідно обмежувати відбиту блискість шляхом правильного вибору типів світильників та розміщенням робочих місць відносно джерел природного та штучного освітлення. При цьому яскравість відблисків на екрані відеотерміналу не повинна перевищувати 40 кд/м^2, яскравість стелі при застосуванні системи відбивного освітлення не повинна перевищувати 200 кд/м^2.

Необхідно обмежувати нерівномірність розподілу яскравості в полі зору осіб, що працюють з відеотерміналом, при цьому відношення значень яскравості робочих поверхонь не повинно перевищувати 3:1, а робочих поверхонь і навколишніх предметів (стіни, обладнання) – 5:1.

Необхідно використовувати систему вимикачів, що дозволяє регулювати інтенсивність штучного освітлення залежно від інтенсивності природного, а також дозволяє освітлювати тільки потрібні для роботи зони приміщення.

Для забезпечення нормованих значень освітлення в приміщеннях з відеотерміналами ЕОМ загального та персонального користування необхідно очищати віконне скло та світильники не рідше ніж 2 рази на рік, та своєчасно проводити заміну ламп, що перегоріли.

7.3 Висновки до сьомого розділу

Безпека життєдіяльності (БЖД) – це галузь знання та науково–практична діяльність, спрямована на вивчення загальних закономірностей виникнення небезпек, їхніх властивостей, наслідків їхнього впливу на організм людини, основ захисту здоров'я та життя людини і середовища її проживання від небезпек, а також на розробку і реалізацію відповідних засобів та заходів щодо створення і підтримки здорових та безпечних умов життя і діяльності людини як у повсякденних умовах побуту та виробництва, так і в умовах надзвичайних ситуацій.

8 ЕКОЛОГІЯ

8.1 Методика дослідження джерел забруднення промислових підприємств

Джерело забруднення атмосфери – це обширне поняття, яке можна широко інтерпретувати, особливо внаслідок діяльності людини, а саме:

– конкретна точка, в якій здійснюється викид шкідливих речовин у повітря (наприклад, димова труба або повітряний вихлоп), у тому значенні, що термін "джерело" застосовується для визначення кількості та типів забруднюючих речовин, для оцінки регіональних технічних проблем, таких як поширення забруднення і висота труби;

– технологічний підхід, тобто врахування технологічного процесу, обладнання (бойлери, печі, коксові батареї, преси, лаконаливні машини, пульверизаційні кабіни, автоматизовані лінії тощо), для яких ця концепція застосовується при встановленні меж викидів, а також оцінці рівня технічних засобів тощо;

– регіональний підхід – ряд джерел у конкретному регіоні, що належать до категорій 1) і 2): контрольовані однією організацією, наприклад, хімічною, металургійною або цементною корпорацією, – ця концепція застосовується для диференціації джерел за величиною, для комплексної оцінки їх впливу на навколишнє середовище тощо.

Оскільки класифікація джерел на технологічні та регіональні блоки є надзвичайно складною, доцільно застосовувати концепцію джерела як технологічного блоку.

На металургійних підприємствах, які є важливим джерелом забруднення атмосфери, проводяться численні операції на стадіях агломерації, в доменних печах, в електродугових печах, кисневих конверторах, в ливарних, коксових та інших виробничих об'єктах, які

роблять свій внесок у забруднення повітря. Цементні виробництва потребують близько 20 технологічних процесів (розмелювання, висушування, подрібнення, нагрівання в печах, охолодження в баштах, транспортування на стрічкових конвеєрах, транспортування готового продукту тощо), що супроводжуються забрудненням повітря, причому кожен з них має особливості і створює власні технічні проблеми. Найбільш складними технологічними блоками є хімічні підприємства. На одному заводі ряд виробничих процесів може спричинити викиди різних забруднюючих речовин, включно з газоподібними (наприклад, при виробництві азотної, сірчаної кислот, віскози та добрив, а також теплової енергії в котельнях).

Численні дослідження вчених-екологів засвідчують, що зі всієї кількості забруднюючих речовин, які викидаються в атмосферне повітря, близько 90 % становлять газоподібні речовини і близько 10 % – тверді та рідкі частинки.

В атмосферу всього потрапляє близько 3 х 10° т газоподібних, рідких і твердих забруднювальних речовин. Зараз на частку людської діяльності припадає близько 10 % від цієї кількості. З інтенсивним розвитком промисловості кількість шкідливих викидів в атмосферу може збільшитися в декілька разів.

В індустріально розвинених країнах, таких як США, Англія, Німеччина, Японія та ін., кількість викидів в атмосферу забруднюючих речовин у цей час становить від 350 до 1000 кг за рік на одну особу. В 2010 році річні викиди шкідливих речовин в атмосферу, за прогнозами вчених, можуть сягнути приблизно 10° т.

Дамо характеристику викидам забруднюючих речовин, які відносять лише до антропогенних джерел, зокрема, на промислових підприємствах.

Викиди шкідливих речовин в атмосферу можна поділити на чотири групи: тверді, рідкі, теплові та парогазоподібні.

Причини утворення твердих речовин (виробничий пил) залежать від типу виробничого процесу та його характеру:

– механічне оброблення різних речовин (буріння, розрівнювання, заповнення, подрібнення, розмелювання, полірування тощо);

– транспортування сипких матеріалів (навантажувально-розвантажувальні процеси, просіювання, змішування тощо).

Одним із значних джерел викидів твердих речовин в атмосферу є металургійна промисловість, зокрема виробництва сирого чавуну (агломерація і доменні печі), сталі (кисневі конвертори та тандем-печі або двополюсні печі), феросплавів, ливарні дільниці та вагранки, коксові установки або генератори.

Основним небезпечним виділенням з доменних печей є колошниковий газ і доменний шлак, в яких є значна кількість пилу. Гранулометричний склад і концентрація та хімічний склад пилу в доменних газах суттєво відрізняються й залежать від фізичних і хімічних властивостей застосованої сировини. Кількість пилу, що утворюється в доменних печах, становить від 20 до 300 кг/т сирого чавуну, або від 2 до 30 % його виробництва. Концентрація пилу змінюється від 10 до 20 мг/м3. Хімічний склад пилу в процесі агломерації: 50 % заліза, по 10 % оксидів кремнію, кальцію та алюмінію, приблизно по 2 % вуглецю, сірки та оксиду магнію. Найбільшим джерелом виділення пилу на металургійних підприємствах є електродугові печі. З джерел літератури відомо, що на 1 т виробленої сталі виділяється 5 – 9 кг пилу.

Виділення твердих і рідких забруднень переважно базується за аналогічними принципами. їх зазвичай об'єднують у групу забруднень у вигляді “частинок”.

Рідкі забруднення (туман, краплі) утворюються: а) при конденсації випарів; б) при розпилюванні або розтіканні рідин; в) у результаті хімічних або фотохімічних реакцій.

Пари можуть конденсуватися внаслідок охолодження в суміші з повітрям або іншим неконденсованим газом. Залежно від точки плавлення конденсованих речовин утворюються рідкі або інколи тверді частинки. Рідина знаходиться в рівновазі з парою при певній температурі й тиску. Якщо парціальний тиск пари в газі перевищує зрівноважуючий парціальний тиск насиченої пари при однаковій температурі, то вважають, що пара перенасичена. При досягненні критичного ступеня перенасичення починається конденсація. Пари речовин в газах конденсуються в основному на дрібнодисперсних пилових частинках унаслідок дії іонів, що знаходяться в атмосфері.

Теплові викиди трапляються під час спалювання, обпалювання, сушіння, плавлення, конденсування, карбонізації, газифікації, дистиляції тощо.

8.2 Статистична оцінка техногенних впливів

Розвиток людського суспільства завжди відбувався і відбувається в тісній взаємодії з природою. Взаємодіючи з природою, людина завжди прагнула поліпшити свій добробут, зробити життя більш комфортним і матеріально забезпеченим. Це обумовило збільшення виробництва необхідної продукції промисловості та сільського господарства і спричинило необмежене використання різноманітних природних ресурсів. Виробництво продукції, як відомо, пов'язане з утворенням відходів, які, потрапляючи в навколишнє природне середовище, забруднюють його.

Крім того, в процесі життєдіяльності людина цілеспрямовано перетворює природу, створюючи на місці природних систем техногенні об'єкти і території — міста і промислові комплекси, шляхи і лінії електропередач, водосховища і кар'єри.

Процес незворотного перетворення людиною частин біосфери на техногенні об'єкти і території дістав назву техногенезу, а частина біосфери, штучно перетворена в результаті життєдіяльності людини і заповнена її продуктами, називається технічною оболонкою біосфери (техносферою).

Основне завдання комплексної оцінки техногенного впливу:

– вивчення техногенних чинників забруднення довкілля;

– класифікація джерел забруднень;

– визначення джерел походження забруднень;

– всі проблемами людства, які породжують забруднення біосфери.

Техногенні чинники забруднення довкілля об'єднують у такі групи:

– атмосферні – хімічне, фізичне, механічне і теплове забруднення;

– водні – океани і моря, забруднення поверхневих і підземних вод;

– грунтові – хімічне, ерозійне забруднення, ущільнення, засолення, заболочення тощо;

– геологічні – негативні екзогенні процеси – зсуви, підтоплення, обвали, абразії берегів тощо;

– біотичні – деградації екосистем, збіднення біорізноманіття, мутації, зникнення лісів і пасовищ, біогенна акумуляція шкідливих речовин тощо; -комплексні – порушення природної структури ландшафтів, поява пустель, деградація земель.

Забруднення класифікують за галузевим принципом:

– промислові – хімічна промисловість, металургійна, видобувна тощо;

– транспортні – автотранспорт, авіаційний, морський тощо;

– енергетичні – теплові і атомні електростанції;

– сільськогосподарські – засоби захисту рослин, мінеральні та органічні добрива тощо;

– пов'язані з військовою діяльністю.

Вплив техносфери на стан атмосфери:

– найбільший вплив на стан атмосфери чинять теплоенергетика, металургійна промисловість, підприємства хімічної та будівельної індустрії, автотранспорт, що викидають у повітря пил, важкі метали, вуглеводні, оксиди карбону, бензапірен та інші речовини;

– найбільший вплив на хімічний склад атмосферного повітря чинить спалювання кам'яного вугілля;

– найпотужнішим негативним техногенним чинником є енергетика – підприємства чорної металургії утворюються пил та оксид сірки, хімічна і нафтохімічна промисловість продукують майже у два рази менше викидів при значно більшій різноманітності забруднюючих речовин; крім газоподібних речовин у повітря потрапляють рідкі і тверді частинки у вигляді аерозолів;

– серед усіх видів транспорту автомобільний посідає перше місце за кількістю і різноманітністю забруднюючих речовин, а також за кількістю незворотних змін ландшафтів та інших негативних впливів на довкілля. містах з розвинутою промисловістю внесок автотранспорту в забруднення довкілля досягає 80% усіх забруднень.

Проблеми, пов'язані з гідросферою, зумовлені нестачею прісної води для потреб людства, її' забрудненням, порушенням природних кругообігів та зменшенням продуктивності водних екосистем. Найбільшими забрудниками водних ресурсів є промисловість, комунальне і сільське господарства країни, яких в структурі забруднення водних ресурсів країни складають стосовно 60, 20 і 17%. Важливою проблемою країни є також забруднення підземних вод. підземні води забруднюючі речовини потрапляють зі звалищ побутових і промислових відходів, при будівництві метро, бурінні свердловин внаслідок виливів нафти і нафтопродуктів під час добування чи переробки, у разі протікання нафтопроводів тощо. Всі ці забруднювачі (пестициди, нітрати, важкі метали, вуглеводні) можуть потрапляти з питними водами і в організм людини, спричиняючи отруєння чи захворювання.

Будь-який вплив людини на природні екосистеми призводить до їх змін, які викликають позитивні чи негативні наслідки для економіки і для всього суспільства. При вирішенні сучасних екологічних проблем великого значення набуває комплексна оцінка регіональних екологічних проблем, яка базується на глибокому вивченні та врахуванні всіх природних і соціально-економічних умов і факторів регіонів. Суть такої оцінки полягає в дослідженні просторової структури історично складених природно-територіальних комплексів та проведенні на цій основі розділу території країни (районуванню) на природні зони (області), округи та райони.

Основне завдання комплексної оцінки в конкретних регіонах полягає у:

– виявленні комплексу несприятливих факторів, що складають необхідний вихідний матеріал для прогнозування можливих негативних наслідків господарської діяльності;

– визначенні характеру і масштабів наслідків;

– -виявленні причини на основі встановлення причинно-наслідкових зв'язків;

– розробці заходів, спрямованих на ліквідацію;

– попередженні і компенсації цих наслідків.

Основною метою комплексної оцінки території є встановлення суспільної значимості наслідків за існуючих масштабів господарського впливу на рівновагу екосистем[61].

Види негативного впливу на організм людини умовно можна об'єднати у дві групи: процеси прямого впливу і процеси непрямого впливу.

Процеси прямого впливу обумовлені безпосереднім контактом людини з техногенними об'єктами (механізмами, машинами) або робочими агентами цих об'єктів (високою температурою, токсичними речовинами, електричним струмом, електромагнітними полями чи іншими формами

енергетичного впливу, активними біологічними організмами, ін.), що можуть завдавати шкоди здоров'ю людини або навіть призводити до її загибелі.

Процеси непрямого впливу на організм людини пов'язані з погіршенням умов життя і діяльності людини (склад повітря, температура, вологість, ін.), які зумовлюють процеси метаболізму в організмі людини. Щоб зрозуміти природу цих факторів впливу, досить задуматися про особливості функціонування такої складної матеріально-енергетично-інформаційної системи, якою є людський організм. Зміна будь-якого з тисяч параметрів (хімічних, фізичних, механічних, біологічних), що до того ж дуже тісно взаємодіють між собою, може виявитися достатньою, щоб серйозно погіршити фізіологічні функції організму людини.

Погіршення якості їжі і питної води є однією з найбільш небезпечних форм непрямого впливу. Це пояснюється чутливістю організму до процесів інтоксикації продуктів, у першу чергу тих, що відповідають за стан метаболізму в організмі людини. Слід підкреслити взаємозв'язок ступеня впливу таких екодеструктивних факторів, як забруднення харчових продуктів і питної води, які, зрештою, визначають імунітет організму і його біологічну стійкість.

Серед основних факторів можна назвати: збалансованість і достатність харчування, можливості повноцінного відпочинку, здоровий спосіб життя та ін. Інтегральними оцінками впливу на організм людини є показники захворюваності і смертності населення.

Зниження інформаційної цінності природних систем, на відміну від попереднього виду впливу, діє не на організм людини, а на її особистісні характеристики. Повноцінне формування особистості людини може відбуватися тільки на тлі інформаційного контакту з природними системами. Інформаційне руйнування природних систем також негативно впливає на психологічний стан людини, а це збіднює резерви її природної життєвої

активності, що, у свою чергу, негативно позначається на формуванні соціальних позицій[61].

8.3 Висновки до восьмого розділу

В даному розділі розглянуто питання методики дослідження джерел забруднення промислових підприємств та статистичної оцінки техногенних впливів.

ВИСНОВКИ

В процесі роботи над дипломною роботою:

- описано чисельний метод сингулярно-спектрального аналізу для часових рядів;
- розглянуто алгоритм MRRR та його дерево представлення;
- проаналізовано можливі методи оптимізації програми;
- обґрунтовано вибір методу розпаралелення програми;
- розглянуто методи та технології паралельних обчислень;
- описано різні типи паралелізму і вибрано один із них – паралелізм на основі задач;
- проведено огляд бібліотек для реалізації паралелізму;
- здійснено порівняльний аналіз цих бібліотек;
- проведена реалізації паралельного методу сингулярного розкладу;
- розроблено адаптацію алгоритму SVD для паралельного виконання;
- модифіковано парадигму «розділяй і володарюй» під паралельні потреби;
- описано стратегію паралелізму для алгоритму MRRR;
- проведене порівняння ключових алгоритмів між собою і зроблено відповідний висновок
- здійснено обчислювальний експеримент;
- розраховано сингулярний розклад для матриць на базі як реальних даних газоспоживання, так і випадково згенерованих даних
- наведено результати використання паралельних методів.

В «Спеціальній частині» описано орієнтований ациклічний граф і його застосування.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Algorithm 880: A Testing Infrastructure for Symmetric Tridiagonals Eigensolver / J. Demmel, O. Marques, B. Parlett, C. Vömel. // ACM TOMS., 2008. – vol. 30, no. 1.– C. 1508–1526.

2. An Implementation of the Tile QR Factorization for a GPU and Multiple CPUs / Kurzak J., Nath R., Du P., Dongarr J. // PARA'10: State of the Art in Scientific and Parallel Computing / – Reykjavík: Springer, 2012. – C. 248–257.

3. Arbenz P. On the spectral decomposition of hermitian matrices modified by flow rankpertubations / P. Arbenz, G. H. Golub – SIAM J. Matrix Anal Appl., 1988. – vol. 9, no. 1. – C. 172-191.

4. Bashe, C. J. The Architecture of IBM's Early Computers [Electronic resource] / [C. J. Bashe, W. Buchholz, G. V. Hawkins та ін.] // IBM Journal of System Development, 1981 – Mode of access: URL: http://web.ece.ucdavis.edu/~vojin/CLASSES/EEC272/S2005/Papers/IBM-Architecture-Bashe_sep81.pdf. – с

5. Bentley Jon Writing Efficient Programs / Jon Bentley – Prentice Hall Ptr, 1982. – 170 с.– ISBN 978-0139702440.

6. Bientinesi P. A Parallel Eigensolver for Dense Symmetric Matrices Based on MRRR / P. Bientinesi, I. Dhillon, R. van der Geijn. // SIAM J. SCI. COMP., 2005. – vol. 21. – C. 43-66.

7. Chapman B. Using OpenMP – Portable Shared Memory Parallel Programming / Barbara Chapman, Gabriele Jost, Ruud van der Pas – The MIT Press, 2007. – 384 с. – ISBN 978-0262533027.

8. Chen D. Taming Hardware Event Samples for FDO Compilation // Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization / [Dehao Chen, Neil Vachharajani, Robert Hundt та ін.]., 2010 – C. 42-52.

9. Cilk: An Efficient Multithreaded Runtime System / [R. Blumofe, C. Joerg, B. Kuszmaul та ін.] // Journal of Parallel and Distributed Computing, 1996. – vol. 37, no. 1 – С. 55–69.

10. Cooper Keith D., Engineering a Compiler/ 2-nd Edition / Keith D. Cooper, Linda Torczon. – Morgan Kaufmann, 2011. – 404 с. – ISBN 1-55860-699-8.

11. Cuppen J. J. M. A divide and conquer method for the symmetric tridiagonal eigenproblem / J. J. M. Cuppen. // Numerische Mathematik, 1980. – vol. 36, no. 2. – С. 177-195.

12. Demmel J. W. Applied Numerical Linear Algebra / J. W. Demmel. – Philadelphia: SIAM, 1997. – 184 с.

13. Demmel James W. On the correctness of some bisection-like parallel eigenvalue algorithms in floating point arithmetic / James W. Demmel, Dhillon Inderj.it, Ren Huan // Electronic Trans. Num. Anal, 1995. – vol. 3. – С. 116-149

14. Dhillon I. Orthogonal Eigenvectors and Relative Gaps / I. Dhillon, B. Parlett –SIAM J. Matrix Anal, 2004. – vol. 25. – С. 858-899.

15. Dhillon I. Relative Robust Representations of Symmetric Tridiagonal Linear Algebra / I. Dhillon, B. Parlett // Linear Algebra and its Applications, 2000. – vol. 309, no. 1-3. – С. 121-151.

16. Dhillona Inderjit S. Multiple representations to compute orthogonal eigenvectors of symmetric tridiagonal matrices / Inderjit S. Dhillona, Beresford N. Parlettb // Linear Algebra and its Applications, 2004. – vol. 387. – С. 1-28.

17. Divide and Conquer Symmetric Tridiagonal Eigensolver for Multicore Architectures // Parallel and Distributed Processing Symposium / [G. Pichon, A. Haidar, M. Faverge та ін.]. – Hyderabad, 2015. – С. 51–60.

18. Dongarra J. A. Quark user’ guide: Queuing and runtime for kernels / J. Dongarra, J. Kurzak, A. YarKhan // Innovative Computing Laboratory University of Tennessee, Technical Report, 2011.

19. Francis J. G. F. The QR Transformation A Unitary Analogue to the LR Transformation – Part 1 / J. G. F. Francis // Computer Journal, 1961. – vol. 4, no. 3 – C. 256-271.

20. Frost R. Modular and Efficient Top-Down Parsing for Ambiguous Left-Recursive Grammars. // 10th International Workshop on Parsing Technologies / Richard Frost, Hafiz Rahmatullah, Paul Callaghan. – Prague , ACL-SIGPARSE, 2007. – C. 109-120.

21. Golub G. H. Some modified matrix eigenvalue problems / G. H. Golub // SIAM Review, 1973. – vol. 15, no. 2. – C. 318-334.

22. Gu Ming A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem / Ming Gu, Stanley C. Eisenstat // SIAM. J. Matrix Anal. Appl., 1995. – vol. 16, no. 1. – C.172-191.

23. Guand M. A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem / M. Guand, S. C. Eisenstat // SIAM J. Matrix Anal. Appl., 1995. – vol. 16, no. 1. – C. 172-191.

24. Hennessy John L Computer Organization and Design. The Hardware/Software Interface. 5th Edition / John L. Hennessy, David A. Patterson – Morgan Kaufmann Publishers, 2013. – ISBN 978-0124077263.

25. Hyde Randall The Art of Assembly Language. 2nd Edition / Randall Hyde – No Starch Press, 2010. – ISBN 978-1593272074.

26. Isard Michael Distributed Data-Parallel Programs from Sequential Building Blocks // European Conference on Computer Systems (EuroSys) / [Michael Isard, Mihai Budiu, Yuan Yu та ін.]. – Lisbon, Portugal, 2007. – vol. 41, no. 3. – C. 59-72. – ISBN 978-1-59593-636-3.

27. Knuth D. The Art of Computer Programming, Volume 1: Fundamental Algorithms / Donald Knuth - Addison-Wesley, 2011. – 672 с. – ISBN 978-0201896831.

28. Kublanovskaya V. N. On some algorithms for the solution of the complete eigenvalue problem / V. N. Kublanovskaya. // USSR Computational Mathematics and Mathematical Physics, 1961. – vol. 1, no. 3. – C. 555-572.

29. LAPACK Users' Guide. / [E. Anderson, Z. Bai, C. Bischof та ін.]. – Philadelphia,: SIAM, 2001. – 258 с.

30. Lessig C. An Implementation of the Tile QR Factorization for a GPU and Multiple CPUs // PPAM 2009: 8th Intern. Conf. on Parallel Processing and Applied Math. – Poland, Wroclaw, 2009. – C. 369–402.

31. Nakatsukasa Yuji A Divide-and-Conquer Algorithm for the Symmetric Tridiagonal Eigenproblem / Yuji Nakatsukasa, Nicholas J. Higham // SIAM. J. Sci. Comput., 2013 – vol. 35, no. 3. – C. A1325-A1349.

32. OpenMP Application Program Interface Version 3.0 [Electronic resource] – Mode of access: URL: http://www.openmp.org/mp-documents/spec30.pdf. – Last access: 10.05.2015. – Title from the screen.

33. OpenMP Application Program Interface, Version 4.0 [Electronic resource] – Mode of access: URL: http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf – Last access: 11.05.2015 – Title from the screen.

34. Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations / [T. Auckenthaler, V. Blum, H. Bungartz та ін.]. // Parallel Comput., 2011 – vol. 36, no. 12. – C. 783 –794.

35. Parallel Tiled QR Factorization for Multicore Architectures / A Buttari, J Langou, J Kurzak, J Dongarra // Concurrency and Computation: Practice and Experience, 2008. – vol. 35, no. 1. – C. 31573-1590.

36. Parlett B. An implementation of the DQDS Algorithm / B. Parlett, O. Marques // Linear Algebra and its Applications, 2000. – vol. 309, no. 1-3. – C. 217-259.

37. Perez J. A dependency-aware task-based programming environment for multi-core architectures / J. Perez, R. Badia, J. Labarta. //Cluster Computing, 2008

IEEE International Conference on, 2008. – C. 142-151. – ISBN 978-1-4244-2639-3.

38. Performance and accuracy of lapack's symmetric tridiagonal eigensolvers [Electronic resource] / J.Demmel, O. Marques, B. Parlett, C. Vomel // SIAM J. Scientific Computing, 2008 – vol. 30, no. 3. – C. 1508-1526. – Mode of access: URL: http://dblp.unitrier.e/db/journals/siamsc/siamsc30.html#DemmelMPV08. – Last access: 23.10.2015. – Title from the screen.

39. Performance and Accurancy of LAPACK's Symmetric Tridiagonal Eigensolvers / J. Demmel, O. Marques, B. Parlett, O. Voemel //SIAM J. SCI COMP., 2008. – vol. 30. – C. 1-28.

40. Pichon P. Divide and Conquer Symmetric Tridiagonal Eigensolver for Multicore Architectures / [G. Pichon, A. Haidar, M. Faverge та ін.]. // Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International – IEEE, 2015. – C. 51-60.

41. Plasma users' guide: Parallel linear algebra software for multicore architectures [Electronic resource] / [E. Agullo, J. Dongarra, B. Hadri та ін.]. // Technical report, University of Tennessee, Innovative Computing Laboratory, 2010. – Mode of access: URL: http://icl.cs.utk.edu/projectsfiles/plasma/pdf/users_guide20091104.pdf. – Last access: 03.05.2015. – Title from the screen.

42. Programming Matrix Algorithm-by-Blocks for Thread-Level Parallelism / [G. Quintana-Ort, E. S. Quintana-Ort, U. Jaume та ін.]. // ACM Trans. Math. Soft., 2009. – vol. 36, no. 3. – 28 с.

43. Reinders J. Intel Threading Building Blocks: Outfitting C++ for multi-core processor parallelism / James Reinders. – Sebastopol: O'Reilly Media, 2007. – 336 с. – ISBN 978-0596514808.

44. Reinders James Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism / James Reinders – Sebastopol: O'Reilly Media, 2007 – 336 с. – ISBN 978-0-596-51480-8.

45. ScaLAPACK USER's Guide / [L. S. Blackford, J. D. Crus, I. Duff та ін.] // Philadelphia: SIAM, 1997.

46. Sedgewick Robert Algorithms / Robert Sedgewick – Addison-Wesley Pub, 1988. – с. 657. – ISBN 978-0201066739.

47. StarPU: A unied platform for task scheduling on heterogeneous multicore architectures / C.Augonnet, S. Thibault, R. Namyst, P. Wacrenier // Euro-Par '09: In Proceedings of the 15th International Euro-Par Conference on Parallel Processing – Heidelberg: Springer-Verlag, 2009. – С. 863–874.

48. Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide / [Z. Bai, J. Demmel, J. Dongarra та ін.]. – Philadelphia: SIAM, 2000. – 440 с.

49. Tisseur F. Parallizing The devide and conquer algorithm for the symmetric tridiagonal eigenvalue problem on distributed memory architechtures / F. Tisseur, J. Dongarra // SIAM J. SCI. COMPUT, 1998. – vol. 20. – С. 2223-2236.

50. Wescott Bob The Every Computer Performance Book / Bob Wescott – CreateSpace, 2013. – 222 с. – ISBN 1482657759.

51. Wilkinson J. The Calculation of the Eigenvectors of Codiagonal Matrices / J. H. Wilkinson. // Computer Journal, 1958. – vol. 2, no. 2. – С. 90–96.

52. Willems P. On MR^3-type Algorithms for the Tridiagonal Symmetric Eigenproblem and Bidiagonal SVD / P. Willems – Disertation, University of Wuppertal, 2010.

53. Бобок С.А. Чрезвычайные ситуации: защита населения и территорий / С.А. Бобок, В.И. Юртушкин – М.: «Издательство ГНОМ и Д», 2000.

54. Закон України «Про оплату праці». Верховна Рада України. Закон від 24.03.1995 № 108/95-ВР, чинний, поточна редакція – Редакція від 01.01.2015. – К.: Відомості Верховної Ради України, 1995, № 17, ст. 121.

55. (Цушко) Максимець О. В. Дослідження методів реалізації паралельних алгоритмів для задачі сингулярно-спектрального розкладу на базі паралельних бібліотек : дипломна робота магістра / О.В. (Цушко) Максимець — Тернопіль: ТНТУ, 2015. – 103 с.

56. Назаревич О. Б. Інформаційна технологія моніторингу газоспоживання міста : автореф. дис. на здобуття наук. ступеня канд. техн. наук: 05.13.06 «Інформаційні технології» / О. Б. Назаревич. – Тернопіль, 2015. – 24 с.

57. Приймак М. Дослідження особливостей енергоспоживання в умовах ритміки методом гістограмного аналізу / М.В. Приймак, О.В. Мацюк, О.Б. Назаревич, Г.В. Шимчук // Міжнародна науково-технічна конференція "Фундаментальні та прикладні проблеми сучасних технологій" (присвячена 50-річчю заснування ТНТУ та 165- річчю з дня народження Івана Пулюя, 19-21 травня)., 2010 – Тернопіль: ТНТУ. – 301 с.

58. Основи охорони праці. В. Ц. Жидецький, В. С. Джигирей, О. В. Мельников — Вид. 2е, стериотипне. — Львів: Афіша, 2000. — 348 с.

59. Катренко П.А., Кіт Ю.В., Пістун І.П. Охорона праці. Курс лекцій. Практикум: Навчальний посібник.- Суми: ВТД “Університетська книга”, 2003. — 496с.

60. https://studfile.net/preview/5211197/page:3/

61. Тарасова В.В. «Екологічна статистика», 2008, - 392с.

ДОДАТКИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національна академія наук України
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Західний науковий центр НАН України
Університет імені П'єра і Марії Кюрі Сорбона Париж (Франція)
Вища школа промислової фізики і хімії міста Париж (Франція)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедімінаса (Литва)
Шяуляйська державна колегія (Литва)
Жешувський політехнічний університет ім. Лукачевича (Польща)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Національний університет біоресурсів і природокористування України
Наукове товариство імені Шевченка
Науковий парк «Інноваційно-інвестиційний кластер Тернопілля»
Асоціація випускників ТНТУ

Матеріали Міжнародної науково-технічної конференції

ФУНДАМЕНТАЛЬНІ ТА ПРИКЛАДНІ ПРОБЛЕМИ СУЧАСНИХ ТЕХНОЛОГІЙ

до 60 річчя з дня заснування
Тернопільського національного технічного університету
імені Івана Пулюя
та 175 річчя з дня народження Івана Пулюя

14 15 травня 2020 року

Тернопіль
2020

УДК 519.8
Р.М. Небесний, І.В. Свистун, Р.З. Золотий, канд техн. наук, доц.
Тернопільський державний технічний університет імені Івана Пулюя

ЗАСТОСУВАННЯ ОРІЄНТОВАНОГО АЦИКЛІЧНОГО ГРАФА

Nebesnyi R, Svistun, R. Zolotyi, Ph. D., Assoc. Prof.

APPLICATION OF ORIENTED ACYCLIC GRAPH

Кожен орієнтований ациклічний граф має топологічне впорядкування вершин таке, що початкова вершина кожного ребра проходить раніше в сортуванні, ніж кінцева. Загалом, це сортування не є унікальним; DAG має унікальне топологічне упорядкування тоді і тільки тоді, якщо він має орієнтований шлях, що містить всі вершини, в цьому випадку сортування таке ж, як у порядку, в якому вершини з'являються на шляху.

Комбінаторне завдання перелічення графа для підрахунку спрямованих ациклічних графів вивчав Робінсон в 1973 р.. Кількість DAG на позначеному вузлі n, де n = 0, 1, 2, 3, ..., (ці цифри можуть з'являтися в будь-якому порядку в топологічної впорядкованості DAG) становить 1, 1, 3, 25, 543, 29281, 3781503, ... Це також можна обчислити за допомогою рекурентного співвідношення

$$a_n = \sum_{k=1}^{n} (-1)^{k-1} \binom{n}{k} 2^{k(n-k)} a_{n-k} \qquad (1.1)$$

Ерік У. Вайнштайн припустив, а Маккей в 2004 р. довели [60], що ті ж числа будуть отримані в результаті обчислення бінарної матриці, для якої всі власні числа є додатними і дійсними.

Орієнтоване дерево це спрямований граф, утворений шляхом орієнтації ребра вільного дерева. Кожне орієнтоване дерево є DAG. Зокрема, це відноситься і до деревоподібних структур, сформованих орієнтацією всіх ребер в напрямку від кореня дерева. Multi-tree-structured граф являє собою орієнтований граф, в якому існує не більше одного орієнтованого шляху (в обидва напрямки) між будь-якими двома вузлами; еквівалентом є DAG, в якому для кожного вузла v, множина вузлів, досяжних з v утворює дерево.

Є ряд задач, які можна вирішити за допомогою DAG, оскільки він грає ключову роль в алгоритмі розв’язку. В роботі розглянуто саме такі алгоритми і тому ці випадки потребують детальнішого аналізу.

Алгоритмічна задача знаходження топологічного сортування може бути вирішена за лінійний час. Алгоритм Кана для топологічного сортування знаходить порядок вершин безпосередньо, шляхом збереження списку вершин, які не мають ребер, що з'єднують їх з вершинами, які ще не були перераховані, і повторно додає одну з таких вершини до кінця списку, який будується. Крім того, топологічний порядок може бути побудований за допомогою алгоритму пошуку в глибину. Також можна перевірити, який із даних орієнтованих графів є DAG за лінійний час декількома способами: або спробувати знайти топологічне сортування, а потім перевірити для кожного ребра чи є результат дійсним, або, як альтернатива, для деяких топологічних алгоритмів сортування, перевіривши, що алгоритм успішно сортує всі вершини не видаючи помилку умови.

Також складною задачею є побудова DAG. Будь-який неорієнтований граф може бути перетворений в DAG, якщо вибрати загальний порядок його вершин і орієнтацію кожного ребра з початкової кінцевої точки до більш пізньої кінцевої точки. Тим не менше, різні загальні порядки сортувань можуть привести до однієї ациклічної орієнтації. Кількість ациклічних орієнтацій рівна $|X(-1)|$, де X є хроматичним многочленом даного графа.

Будь-який орієнтований граф може бути перетворений в DAG видаленням набору зворотних вершин або набору зворотних дуг. Тим не менш, знаходження найменшого такого набору це NP-складна задача. Довільний орієнтований граф може бути перетворений в DAG, це називається стисненням, шляхом перетворення кожного з його сильно зв'язних компонентів в одну супер вершину. Коли граф вже є ациклічним, його найменша множина зворотних вершин і множина зворотних дуг порожня, і його стиснення є самим графом.

Транзитивне замикання даного DAG, з n вершинами і m ребрами, може бути побудоване за час O(mn) за допомогою або пошуку в ширину, або пошуку в глибину, щоб перевірити досяжність з кожної вершини. Крім того, воно може бути вирішене за час $O(n^{\omega})$, де $\omega < 2{,}373$ є показником для алгоритмів швидкого множення матриць; це теоретичне поліпшення в порівнянні з граничним O(mn) для щільних графів.

У всіх цих алгоритмах транзитивного замикання можна виділити пари вершин, яких можна досягнути шляхом довжиною два або більше з пар, які можуть бути пов'язані шляхом одиничної довжини. Транзитивне скорочення складається з ребер, що утворюють шлях одиничної довжини, які є єдиними шляхами, що з'єднують їх кінцеві точки. Таким чином, транзитивне скорочення може бути побудоване в тих же межах асимптотичного часу, що і транзитивне замикання.

Задача замикання приймає в якості вхідних даних орієнтований ациклічний граф з вагами на його вершинах і шукає мінімальну (максимальну) вагу закриття набір вершин з яких не виходять ребра. ЇЇ можна вирішити за поліноміальний час, використовуючи скорочення задачі про максимальний потік.

Деякі алгоритми спрощуються при використанні DAG замість загальних графів, який базується на принципі топологічного сортування. Наприклад, можна знайти найкоротший шлях і найдовший шлях від деякої вершини DAG за лінійний час шляхом обробки вершин в топологічному порядку, і обчислення довжини шляху для кожної вершини, який буде мінімальною або максимальною довжиною, отримується за допомогою будь-якого з його вхідних ребер. На відміну від цього, для довільних графів знаходження найкоротшого шляху може вимагати повільніших алгоритмів, таких як алгоритм Дейкстри або алгоритм Беллмана-Форда, а знаходження найдовшого шляху в довільному графі має NP-складність.

Представлення DAG часткового порядку часто використовується в задачах планування для систем завдань з обмеженим впорядкуванням. Наприклад, DAG можуть бути використані для опису залежностей між клітинами таблиці: якщо одна комірка обчислюється за формулою з участю значення другої комірки, потрібно намалювати ребро DAG з другої комірки до першої. Якщо вхідні значення для таблиці змінюються, всі інші значення таблиці можуть бути перераховані за допомогою однієї оцінки на клітку, шляхом топологічного сортування клітинки і переоцінки кожної клітинки згідно обрахованого порядку. Аналогічні проблеми впорядкування завдань виникають в make-файлах для компіляції програми, планування інструкцій для низькорівневої оптимізації програми і планування технічного оцінювання та аналізу програми для управління великими проектами. Залежні графи без циклічних залежностей формують орієнтовані ациклічні графи.

Література

1. Tisseur F. Parallizing The devide and conquer algorithm for the symmetric tridiagonal eigenvalue problem on distributed memory architechtures / F. Tisseur, J. Dongarra // SIAM J. SCI. COMPUT, 1998. Vol. 20. C. 2223 2236.

2. Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide / [Z. Bai, J. Demmel, J. Dongarra та ін.]. Philadelphia: SIAM, 2000. 440 с.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національна академія наук України
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Західний науковий центр НАН України
Університет імені П'єра і Марії Кюрі Сорбона Париж (Франція)
Вища школа промислової фізики і хімії міста Париж (Франція)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедімінаса (Литва)
Шяуляйська державна колегія (Литва)
Жешувський політехнічний університет ім. Лукачевича (Польща)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Національний університет біоресурсів і природокористування України
Наукове товариство імені Шевченка
Науковий парк «Інноваційно-інвестиційний кластер Тернопілля»
Асоціація випускників ТНТУ

Матеріали Міжнародної науково-технічної конференції

ФУНДАМЕНТАЛЬНІ ТА ПРИКЛАДНІ ПРОБЛЕМИ СУЧАСНИХ ТЕХНОЛОГІЙ

до 60 річчя з дня заснування
Тернопільського національного технічного університету
імені Івана Пулюя
та 175 річчя з дня народження Івана Пулюя

14 15 травня 2020 року

Тернопіль
2020

УДК 519.8
Р.М. Небесний, І.В. Свистун, О.С. Голотенко, канд техн. наук
Тернопільський державний технічний університет імені Івана Пулюя

ОРІЄНТОВАНИЙ АЦИКЛІЧНИЙ ГРАФ

R. Nebesnyy, I. Svistun, O. Golotenko, Ph. D.

ORIENTED ACYCLIC GRAPH

У математиці та інформатиці, орієнтований ациклічний граф (DAG), являє собою орієнтований граф без орієнтованих циклів. Тобто, він формується сукупністю вершин і орієнтованих ребер, кожне ребро з'єднує одну вершину з іншою, так що немає не можливо з деякої вершини V пройти деяку послідовність ребер і знов повернутися до V.

DAG можуть бути використані для моделювання багатьох різних видів інформації. Відношення досяжності в DAG утворює частковий порядок, а будь-яке кінцевий частковий порядок може бути представлений у вигляді DAG за допомогою відношення досяжності. Набір завдань, які повинні формувати послідовність, за умови, що деякі завдання повинні виконуватись раніше, ніж інші, можуть бути представлені у вигляді DAG з вершиною для кожного завдання і ребрами для кожного обмеження; алгоритми топологічного впорядкування можуть використовуватися для генерації валідної послідовності. Крім того, DAG може бути використаний як просторово-ефективне представлення набору послідовностей з підпослідовностями, які перекриваються. DAG також використовується для представлення системи подій або можливих подій і причинно-наслідкових зв'язків між ними. DAG також можуть бути використані для моделювання процесів, в яких потоки даних в рухаються через мережу процесорів, або станів сховища в системі керування версіями.

Відповідна концепція множини неорієнтованих графів називається лісом – неорієнтований граф без циклів. Вибір орієнтації для лісу виробляє особливий вид спрямованого ациклічного графа, що називається орієнтованим деревом. Крім того, кожен неорієнтований граф має ациклічну орієнтацію, розподілення напрямку для його ребер, що робить його орієнтованим ациклічним графом.

DAG має наступні математичні властивості: доступність; транзитивне замикання; транзитивне скорочення.

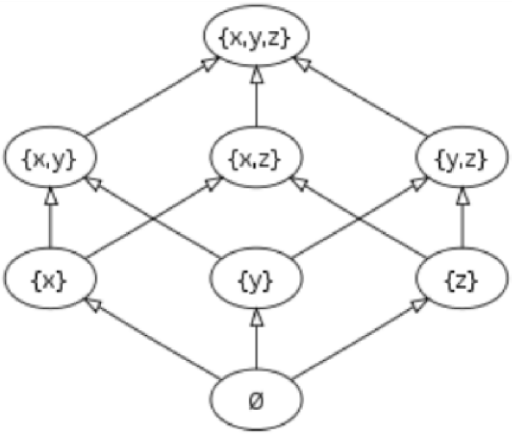


Рисунок 1 – Діаграма Гассе, яка представляє відношення часткового порядку серед множини підмножин елементів дерева

Кожен орієнтований ациклічний граф породжує частковий порядок ⊆ для його вершин, де u ⊆ v, коли існує орієнтований шлях з u в v в DAG. Тим не менше, багато різних DAG можуть призвести до цього ж відношення досяжності співвідношенням. Наприклад, DAG з двома ребрами a → b і b → c має те ж відношення досяжності, що й

граф з трьома ребрами a → b, b → c, a → c. Нехай G це DAG, його транзитивне скорочення це граф з найменшою кількістю ребер, що представляє те ж відношення досяжності, що і G, і його транзитивне замикання це граф з більшістю ребер, який представляє те ж відношення досяжності.

Транзитивне скорочення і транзитивне замикання однозначно визначені для DAG; на відміну від цього, для орієнтованого графа, що не є ациклічним, не може бути більше, ніж одного мінімального підграфа з тим же відношенням досяжності.

Транзитивне замикання G має ребро u → v для зв'язаної кожної пари u ≤ v різних елементів в відношенні досяжності G, і, отже, може розглядатися як орієнтоване представлення відношення досяжності ⊆ в термінах теорії графів: кожна частково впорядкована множина можуть бути перетворена в DAG в цьому випадку. Якщо DAG G являє собою часткове відношення ⊆, то транзитивне скорочення G є підграфом G з ребром u → v для кожної пари часткового порядку ⊆; транзитивне скорочення корисне в візуалізації часткового порядку, тому що вони мають менше ребер, ніж інші графи, що представляють той самий порядок, і тому спрощують креслення графа. Діаграма Гассе, зображена на рисунку 1 часткового порядку являє собою креслення транзитивного скорочення, в якому орієнтація кожного ребра показана шляхом розміщення вихідної вершини ребра на нижчу, ніж кінцева вершина ребра, позицію.

Отже, тип застосування спрямованих ациклічних графів виникає в стислому поданні безлічі послідовностей у вигляді шляхів в графі. Наприклад, орієнтований ациклічний граф слова являє собою структуру даних в інформатиці, утворену орієнтованим ациклічним графом з одним джерелом і з ребрами, поміченими літерами або символами; шлях від джерела до приймача в цьому графіку являє собою набір рядків, таких як англійські слова. Орієнтований ациклічний граф слова економить простір порівняно з префіксним деревом, дозволяючи шляхам розходитися і сходитися таким чином, що набір слів з тими ж ймовірними суфіксами може бути представлений в одному вузлі дерева.

Та ж ідея використання DAG для представлення сімейства шляхів, використана в бінарній діаграмі рішень, структури даних, які базуються на DAG використовуються для представлення двійкових функцій. У бінарній діаграмі рішень кожна вершин, яка не є приймачем, позначена ім'ям двійкової змінної, а кожен приймач і кожне ребро позначене 0 або 1. Значення функції для будь-якої підстановки змінної є значенням приймача, знайденим по шляху від однієї вершини джерела до кожної вершини не приймача по вихідному ребру, що має міткою значення змінної в початковій вершині. Так само, як спрямовані ациклічні графи слів можна розглядати як стислий вигляд префіксних дерев, бінарні діаграми рішень можна розглядати як стислі форми дерев рішень, які економлять простір, дозволяючи шляхам з'єднуватись, коли вони погоджуються з результатами всіх інших рішень.

Література.

1. Tisseur F. Parallizing The devide and conquer algorithm for the symmetric tridiagonal eigenvalue problem on distributed memory architechtures / F. Tisseur, J. Dongarra // SIAM J. SCI. COMPUT, 1998. – vol. 20. C. 2223-2236.
2. Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide / [Z. Bai, J. Demmel, J. Dongarra .]. Philadelphia: SIAM, 2000. 440 с.