Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Центр післядипломної освіти
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Методи розподілу обчислювальних ресурсів в інтелектуальній системі екологічного моніторингу
(тема)

Виконав:
студент 2 курсу, групи СШІздм-21-1
Чмуж В.С.
(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)

Освітня програма Системи штучного інтелекту
(повна назва освітньої програми)

Керівник проф. Удовенко С.Г.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ В.О. Філатов
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Центр Післядипломної освіти
(повна назва)

Кафедра Штучного інтелекту
(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системи штучного інтелекту (СШІ)
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«_____» _____ 20____ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Чмужу Володимиру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи розподілу обчислювальних ресурсів в інтелектуальній системі екологічного моніторингу

затверджена наказом університету від 31 березня 20 23 р. № 73 Стз.
2. Термін подання студентом роботи до екзаменаційної комісії 23 травня 20 23 р.
3. Вихідні дані до роботи науково-технічні публікації, дані інтернет-джерел та відомих наукових проектів щодо методів розподілу ресурсів, Python документація

4. Перелік питань, що потрібно опрацювати в роботі
1) Аналіз предметної галузі
2) Методи диспетчеризації обчислювальних ресурсів інтелектуальної системи екологічного моніторингу
3) Програмна реалізація завдань диспетчеризації ресурсів системи екологічного моніторингу

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри)_____

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	03.04.2023	Виконано
2	Аналіз предметної області	04.04.2023 -14.04.2023	Виконано
3	Формування постановки задачі	15.04.2023-18.04.2023	Виконано
4	Дослідження задачі диспетчеризації завдань	19.04.2023-23.04.2023	Виконано
5	Розроблення методів диспетчеризації ресурсів в	25.04.2023-29.04.2023	Виконано
6	Реалізація програми диспетчеризації	30.04.2023-09.05.2023	Виконано
7	Обробка результатів роботи програми	10.05.2023-15.05.2023	Виконано
8	Підготовка матеріалів для пояснювальної записки	16.05.2023- 18.05.2023	Виконано
9	Оформлення пояснювальної записки	19.05.2023-20.05.2023	Виконано
10	Попередній захист	21.05.2023	Виконано
11	Захист перед ЕК	23.05.2023	

Дата видачі завдання 3 квітня 2023 р.

Студент _____ _____
(підпис)

Керівник роботи _____ проф. Удовенко С.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 101 с., 21 рис., 2 дод., 25 джерел.

ВЕКТОРНА ОБРОБКА ДАНИХ, ДИСПЕТЧЕР, ІНТЕЛЕКТУАЛЬНА СИСТЕМА ЕКОЛОГІЧНОГО МОНІТОРИНГУ, КЛАСТЕРИЗАЦІЯ, ПАРАЛЕЛЬНІ ОБЧИСЛЕННЯ, РОЗПОДІЛЕНІ ОБЧИСЛЮВАЛЬНІ РЕСУРСИ.

Об'єкт дослідження – процеси диспетчеризації обчислювальних ресурсів в інтелектуальній системі екологічного моніторингу.

Предмет дослідження – методи розподілу обчислювальних ресурсів в інтелектуальній системі екологічного моніторингу.

Мета роботи – дослідження методів розподілу обчислювальних ресурсів в інтелектуальній системі екологічного моніторингу.

Методи дослідження – методи побудови систем екологічного моніторингу довкілля, методи розподілу завдань між обчислювальними вузлами інтелектуальних систем екологічного моніторингу, методи моделювання процесів моніторингу.

Досліджено методи диспетчеризації обчислювальних ресурсів в інтелектуальних системах екологічного моніторингу. Функціями диспетчера ресурсів є планування завдань системи, побудова топології розміщення і послідовності виконання завдань на обчислювальних вузлах.

Розглянуто методи диспетчеризації, що дозволяють реагувати на реконфігурацію розподіленого обчислювального середовища для вирішення різних класів завдань в системах екологічного моніторингу.

Докладно розглянуто питання розробки відповідних алгоритмів та можливості реалізації системи з використанням сучасних методів та компонентів.

ABSTRACT

Explanatory note: 101 p., 21 fig., 2 ann., 25 sources.

CLUSTERIZATION, DISPATCHER, DISTRIBUTED COMPUTING RESOURCES, INTELLIGENT ENVIRONMENTAL MONITORING SYSTEM, PARALLEL COMPUTING, VECTOR DATA PROCESSING.

The object of research is the dispatching processes of computing resources in the intelligent environmental monitoring system.

The subject of the research is the methods of distribution of computing resources in the intelligent system of environmental monitoring.

The purpose of the work is to study the methods of distribution of computing resources in the intelligent system of environmental monitoring.

Research methods – methods of building environmental monitoring systems, methods of distributing tasks between computing nodes of intelligent environmental monitoring systems, methods of modeling monitoring processes.

The methods of dispatching computing resources in intelligent environmental monitoring systems have been studied. The functions of the resource manager are the planning of system tasks, the construction of the placement topology and the sequence of execution of tasks on computing nodes.

Dispatching methods that allow responding to the reconfiguration of a distributed computing environment to solve various classes of tasks in environmental monitoring systems are considered.

The development of appropriate algorithms and the possibility of implementing the system using modern methods and components are considered in detail.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ

АОК – модуль автоматичного опитування екологічних даних;

ГІС – геоінформаційна система;

ЕН – екологічна небезпека;

ІСЕМ – інтелектуальна система екологічного моніторингу;

ПОЕІ – підсистема первинної обробки екологічної інформації;

РВС – розподілена обчислювальна система;

РС – робоча станція;

ЦФ – цільова функція;

ФАВ – програмний модуль фіксації аварійних викидів;

СОЕІ – модуль статистичної обробки екологічної інформації;

МЕО – моделювання екологічної обстановки;

RR – Round Robin – карусельна дисципліна обслуговування.

ВСТУП

Розробку та удосконалення методів екологічного моніторингу слід віднести до переліку фундаментальних наукових і інженерних досліджень.

Екологічний моніторинг в промислових зонах здійснюється з використанням комплексних систем спостережень за станом навколишнього середовища, що забезпечують оцінку і прогноз змін стану навколишнього середовища під впливом природних і антропогенних факторів [1], [2].

У Законі України «Про Основні засади (стратегію) державної екологічної політики України на період до 2030 року» зазначається необхідність розвитку систем екологічного моніторингу, що здійснюють регулярне спостереження за природним середовищем, природними ресурсами, рослинним і тваринним світом та дають можливість аналізувати процеси зміну стану екосистем, які відбуваються під впливом антропогенного навантаження [3].

У загальному вигляді процес екологічного моніторингу, що здійснюється у складі автоматизованих систем екологічного моніторингу (АСЕМ), відбувається за такою схемою: виявлення конкретних об'єктів моніторингу у навколишньому середовищі; вимірювання параметрів відповідними підсистемами АСЕМ (зазвичай, підсистемами збору та передачі інформації); обробка даних та формування узагальнених оцінок; прогнозування можливих змін екологічного стану в зонах моніторингу. Інформація про стан навколишнього середовища, отримана в АСЕМ, використовується для запобігання або усунення негативної екологічної ситуації, для оцінки несприятливих наслідків зміни стану навколишнього середовища, а також для розробки прогнозів та програм в області екологічного розвитку та охорони навколишнього середовища .

Підсистеми екологічного моніторингу розрізняються по об'єктах

спостереження. Оскільки компонентами навколишнього середовища є повітря, вода, мінерально-сировинні та енергетичні ресурси, біоресурси, то виділяють відповідні їм підсистеми моніторингу. Однак, підсистеми моніторингу не мають єдиної системи показників, єдиного районування територій, єдності в періодичності відстежування тощо, що унеможливлює прийняття адекватних заходів при управлінні розвитком та екологічним станом територій [4], [5]. Тому при прийнятті рішень важливо орієнтуватися не тільки на дані локальних систем моніторингу (гідрометеослужби, моніторингу ресурсів та ін.), а створювати на їх основі комплексні системи екологічного моніторингу.

Зазвичай на території промислових зон вже існують мережі спостережень, що належать різним службам, і які установчо-роз'єднані і не скоординовані в хронологічному, параметричному та інших аспектах. Тому завдання підготовки оцінок, прогнозів, критеріїв альтернатив вибору управлінських рішень на базі наявних в регіоні відомчих даних має, в загальному випадку, елементи суттєвої невизначеності. У зв'язку з цим, центральними проблемами організації екологічного моніторингу є еколого-господарське районування і вибір «інформативних показників» екологічного стану територій з перевіркою їх системної достатності.

Важливим є розвиток аналітичного підходу до вирішення завдань моніторингу екологічного стану навколишнього середовища з використанням методів обчислювального інтелекту. В сучасних системах екологічного моніторингу реального часу використовують принцип управління станом довкілля на основі аналізу потоку подій. Таке управління базується на реєстрації і обробці подій, що оцінюються в реальному часі за даними про поточний стан навколишнього середовища. На відміну від традиційних АСЕМ з фіксованою схемою виконання операцій збору та обробки екологічної інформації, в даному випадку здійснюється інтелектуальне управління функціонуванням системи моніторингу в залежності від особливостей вхідного потоку даних.

Інтелектуальне управління передбачає доцільність використання в складі АСЕМ таких елементів штучного інтелекту як методи логічного висновку, нечіткої логіки, еволюційної оптимізації, мультиагентних систем тощо [6], [7], [8].

Такий клас АСЕМ інколи відносять до інтелектуальних систем екологічного моніторингу (ІСЕМ).

Цим визначається актуальність теми цієї магістерської роботи, в якій розглянуті важливі питання організації обчислювального процесу в розподіленій ІСЕМ, що здійснює інтелектуальну підтримку прийняття рішень щодо підвищення рівня екологічної та техногенної безпеки в регіоні.

Відповідно до завдання на проектування в даній роботі необхідно розробити та дослідити методи і алгоритми розподілу обчислювальних ресурсів в ІСЕМ.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Загальна характеристика завдань екологічного моніторингу

Проблема захисту навколишнього середовища є надзвичайно актуальною для промислово розвинених країн. Інформація про екологічний стан природного середовища (особливо в промислових зонах) важлива для підтримки якості життя громадян та для здійснення безпечної господарської діяльності в зонах екологічного моніторингу. Відомо, що зміна екологічного стану довкілля відбуваються не лише під впливом природних процесів, але і в результаті промислової діяльності підприємств, що інколи може привести до антропогенних змін в природі.

Екологічна ситуація в промислових регіонах залежить від різних чинників, зокрема, від викидів, спричинених стаціонарними джерелами забруднення. Наприклад, більшість з великих підприємств застосовують технології, які супроводжуються виділенням до навколишнього середовища хімічних сполук, що відносяться до 1 і 2 класів небезпеки та негативно впливають на загальний екологічний стан довкілля. При цьому суттєвою проблемою, пов'язаною із забрудненням атмосферного повітря промисловими підприємствами, є близьке розташування житлової зони до основних промислових районів.

На сьогодні здійснюються регулярні спостереження за кліматичними змінами в природі. Поширеними є метеорологічні, фенологічні, сейсмологічні та інші види спостережень стану довкілля. З кожним роком розширюється коло таких спостережень, кількість вимірюваних параметрів. Підвищення складності проблеми екологічного моніторингу викликає необхідність збільшення станцій екологічного спостереження в складі АСЕМ.

Екологічний моніторинг має забезпечити адекватну оцінку екологічних умов середовища проживання громадян, біологічних об'єктів а

також оцінку стану функціонування екосистем, з метою створення умов для визначення коригувальних впливів, якщо навіть деякі екологічні показники ще не отримані в АСЕМ [9], [10], [11], [12].

Аналіз вітчизняних та зарубіжних публікацій щодо існуючих систем екологічного моніторингу дозволяє виділити відсутність єдиної концепції і розуміння цілей і завдань моніторингу. Крім того, роботи по створенню галузевих систем екологічного моніторингу не завжди скоординовані, що призводить до певних труднощів щодо побудови ефективних систем глобального рівня. Також слід відзначити недотримання принципів стандартизації та взаємозв'язку підходів до вирішення таких проблем: контролю за радіаційним впливом ядерно-енергетичних підприємств на навколишнє середовище; створенню приладового, метрологічного та методичного забезпечення робіт з розробки нових і експлуатації наявних систем контролю і моніторингу; врахування кліматичних змін, пов'язані як з використанням традиційних джерел енергії, так і з наслідками життєдіяльності людини.

Можливість отримувати безперервний потік екологічних та просторових даних (зокрема з використанням засобів ДЗЗ – дистанційного зондування Землі) дозволяє вживати превентивних заходів і уникати багатьох загроз, які пов'язані з аномаліями в навколишньому середовищі. Сьогодні активно розробляються засоби екологічного моніторингу з використанням геоінформаційних систем (ГІС ЕМ) різного функціонального призначення [13], [14], [15].

Для реалізації екологічного моніторингу важливе місце відводиться дистанційним методам, здатним безпосередньо представляти в ІСЕМ екологічні дані, пов’язані з рівнем екологічної безпеки в різних точках промислової зони.

Суттєві результати щодо оцінки антропогенних впливів отримуються шляхом комплексного використання даних, отриманих за допомогою ДЗЗ, авіації і наземних систем контролю. До інформації, одержуваної з

використанням ДЗЗ для цілей екологічного моніторингу, насамперед, відносять інформацію про стан лісних та сільськогосподарських угідь, про стан земної поверхні (про випадки порушення земної поверхні через антропогенну діяльність, про ерозійні процеси, про забруднення атмосфери, водойм та суші тощо).

Дистанційний контроль екологічного стану довкілля з застосуванням ДЗЗ можливий, зокрема, шляхом картування ґрунтів з подальшим аналізом властивостей ґрунтового покриву (вологості, засоленості, вмісту гумусу, наявності рослинності). Ґрунти можуть розпізнаватися за вимірюваннями прямих параметрів поверхні, а також шляхом визначення непрямих параметрів (за геологічними, геоботанічними та геоморфологічними показниками).

Спостереження за штучними та природними водними об'єктами дозволяють з великою точністю ідентифікувати про зміну їх стану (в тому числі і антропогенного характеру).

На сьогодні екологічний моніторинг навколишнього середовища і природних ресурсів здійснюється зазвичай за допомогою спеціалізованих систем спостережень і подальшої обробки отриманих даних. Комп'ютерна обробка реалістичних картографічних зображень ландшафтних об'єктів в геоінформаційних системах екологічного моніторингу (ГІСЕМ) передбачає в загальному випадку реалізацію етапів попередньої обробки, сегментації, розпізнавання і інтерпретації з використанням методів штучного інтелекту.

Окремий практичний інтерес представляють завдання розпізнавання об'єктів за супутниковими знімками і аерознімками. До основних труднощів такого розпізнавання відноситься зміна видимості об'єктів, що викликається різними чинниками (освітлення, орієнтація, наявність викривлених фрагментів зображення тощо). Аналіз і інтерпретація знімків є важливою частиною реалізації багатьох ГІС додатків (наприклад, побудова топографічних карт, кадастрових планів, локалізації районів забруднення, моніторинг зміни контурів окремих ділянок зображень під

впливом природних і антропогенних факторів).

За допомогою систем ДЗЗ якісно ідентифікуються антропогенні зміни в навколишньому середовищі (наприклад, лісові пожежі, забруднення атмосфери та земної поверхні). Системи екологічного контролю дозволяють також ідентифікувати шари пилу, хмари при пилових бурях; димові викиди тощо.

В даний час вже накопичено чималий досвід використання ГІС ЕМ та ІСЕМ для вивчення антропогенних змін в біосфері [16], [17]. В останні роки біологічна наука починає займати важливе місце в моніторингу стану навколишнього середовища. Важливим для моніторингу стану довкілля є той факт, що перелік і кількість викидів в навколишнє середовище забруднюючих речовин є надзвичайно великим (до 500 тисяч найменувань).

В промислових зонах екологічним спостереженням підлягають речовини, викид яких носить масовий характер і забруднення якими є особливо небезпечним. До таких речовин слід віднести, наприклад, діоксид сірки, монооксид вуглецю, нафтопродукти, поверхнево-активні речовини для природних вод; пестициди для ґрунтів тощо. Крім того, в зонах екологічного моніторингу мають контролюватися і токсичні речовини. Таким чином, екологічний моніторинг покликаний здійснювати стратегію і тактику не лише спостереження і контроль, а й оцінку та прогнозу стану навколишнього середовища.

Розвиток нових методів визначення токсичних та інших шкідливих речовин у навколишньому середовищі підняло на якісно новий рівень аналіз інтенсивності забруднення повітря, води і ґрунту, а також загальну оцінку якості навколишнього середовища. Заходи по захисту навколишнього середовища від викидів виробництв, вимагають великих економічних витрат (в деяких випадках воно можуть сягати до 50% від вартості основного виробництва). Тому до якості контролю, його надійності, точності в АСЕМ мають пред'являтися високі вимоги.

На сьогодні сформовано перелік пріоритетних забруднюючих речовин, які слід визначати в першу чергу. Більшість нормованих забруднюючих речовин для повітря мають гранично допустиму концентрацію (ГДК) в межах 0,005-0,1 мг / м3 (це, зокрема, оксид ванадію, неорганічні сполуки миш'яку, деякі органічні речовини тощо). Нормовані забруднюючі речовини для водойм мають ГДК в діапазоні 0,1-1 мг / л, а для токсичних речовин встановлена ГДК в межах 0,001-0,003 мг / л. Втім, незважаючи на зниження числа надзвичайних ситуацій на виробництвах, середньорічні концентрації небезпечних речовин в атмосфері ще й досі суттєво перевищують гранично допустимі значення.

При аналізі об'єктів екологічного моніторингу навколишнього середовища необхідно враховувати хімічні, фотохімічні і біохімічні перетворення досліджуваних речовин і можливість міграції забруднюючих речовин з однієї зони до іншої.

Згідно з завданням на кваліфікаційну роботу далі розглядатимемо особливості побудови інтелектуальних систем екологічного моніторингу.

1.2 Особливості побудови інтелектуальних систем екологічного моніторингу

До найбільш поширених типів АСЕМ слід віднести системи екологічної та техногенної безпеки регіонів. Ці системи будуються на базі розподілених обчислювальних комплексів, які об'єднують вимірювальні пристрої та контролери пунктів моніторингу, а також робочі станції центру моніторингу, які координуються з рівнем управління регіоном [18], [19].

Робочі станції здійснюють обробку, подання екологічної інформації, її архівування і аналіз. Крім того, вони також виконують розрахунки з прогнозування і спільно з контролерами здійснюють моніторинг стану всіх програмно-технічних засобів, що входять в обчислювальний комплекс.

Реалізацію функцій нижнього рівня системи управління екологічною

безпекою регіону забезпечують контролери [20]. Контролери, що входять до складу обчислювальної мережі, виконують функції введення інформації з аналогових і дискретних вхідних каналів зв'язку з об'єктом, первинної переробки цієї інформації (лінеаризацію, масштабування, приведення до фізичних одиниць, контроль на достовірність), виявлення загроз і реєстрацію подій. Використовуючи отриману з датчиків екологічну інформацію про об'єкт управління, контролери можуть виконувати також керуючі функції нижнього рівня.

Контролери різняться між собою за ступенем швидкодії, об'ємом оперативної пам'яті, кількістю каналів введення-виведення, інтерфейсів, характеристиками надійності та математичним забезпеченням.

Системи екологічної та техногенної безпеки промислових регіонів відносяться до класу систем критичного застосування з розподіленими обчислювальними ресурсами, тому при розробці таких систем особливу важливість набувають питання оптимізації обчислювального навантаження на різних рівнях паралельно працюючих елементів керуючої обчислювальної системи.

Зокрема, істотне підвищення продуктивності мережі в ICEM може бути досягнуто за рахунок оптимального балансування функцій пакета програм між робочими станціями і контролерами. Такий розподіл може дати економію часу процесорів робочих станцій або контролерів і суттєво зменшити навантаження на лініях зв'язку мережі ICEM.

Сучасні системи екологічного моніторингу в промислових регіонах будуються за допомогою спеціальних програмних пакетів, зокрема, інтегрованих систем конфігурації програм SCADA (Supervisory Control and Data Acquisition) [21]. Ці пакети мають високий ступінь інтеграції засобів розробки і виконання, дружній інтерфейс, підтримують різноманітні засоби зв'язку з об'єктами екологічного контролю, що дозволяє створювати ефективні і надійні системи моніторингу. На відміну від спеціалізованих систем нижнього рівня, які поставляються в комплекті з фірмовою

апаратурою збору даних і управління (програмованими контролерами), ці системи відкриті для зв'язку з сучасними контролерами різних виробників. На робочих станціях реалізуються обробка, визначення спектру можливих загроз, а також прийняття тактичних рішень на основі інформації, отриманої від рівня збору екологічних даних, що реалізується контролерами в пунктах моніторингу.

Рішення завдання закріплення програм та їх функцій за робочими станціями або за контролерами залежить від характеристик робочих станцій і контролерів, а також від топології і швидкості роботи мережевих засобів, що забезпечують обмін інформацією між контролерами і робочими станціями. Оптимальне рішення цього завдання може істотно збільшити продуктивність системи в цілому і зробити її більш стійкою до відмов обладнання.

На рисунку 1.1 наведено загальну схему зв'язків між рівнями екологічного контролю в промисловому регіоні.

Рисунок 1.1 – Схема зв'язків між рівнями екологічного моніторингу в промисловому регіоні

Програмне забезпечення ICEM в промисловому регіоні може бути побудовано побудовано на основі SCADA-систем [21].

Архітектура і склад пакету прикладних програм (ППП), що застосовуються в деяких відомих системах екологічного моніторингу промислових регіонів, наведено на рисунку 1.2.

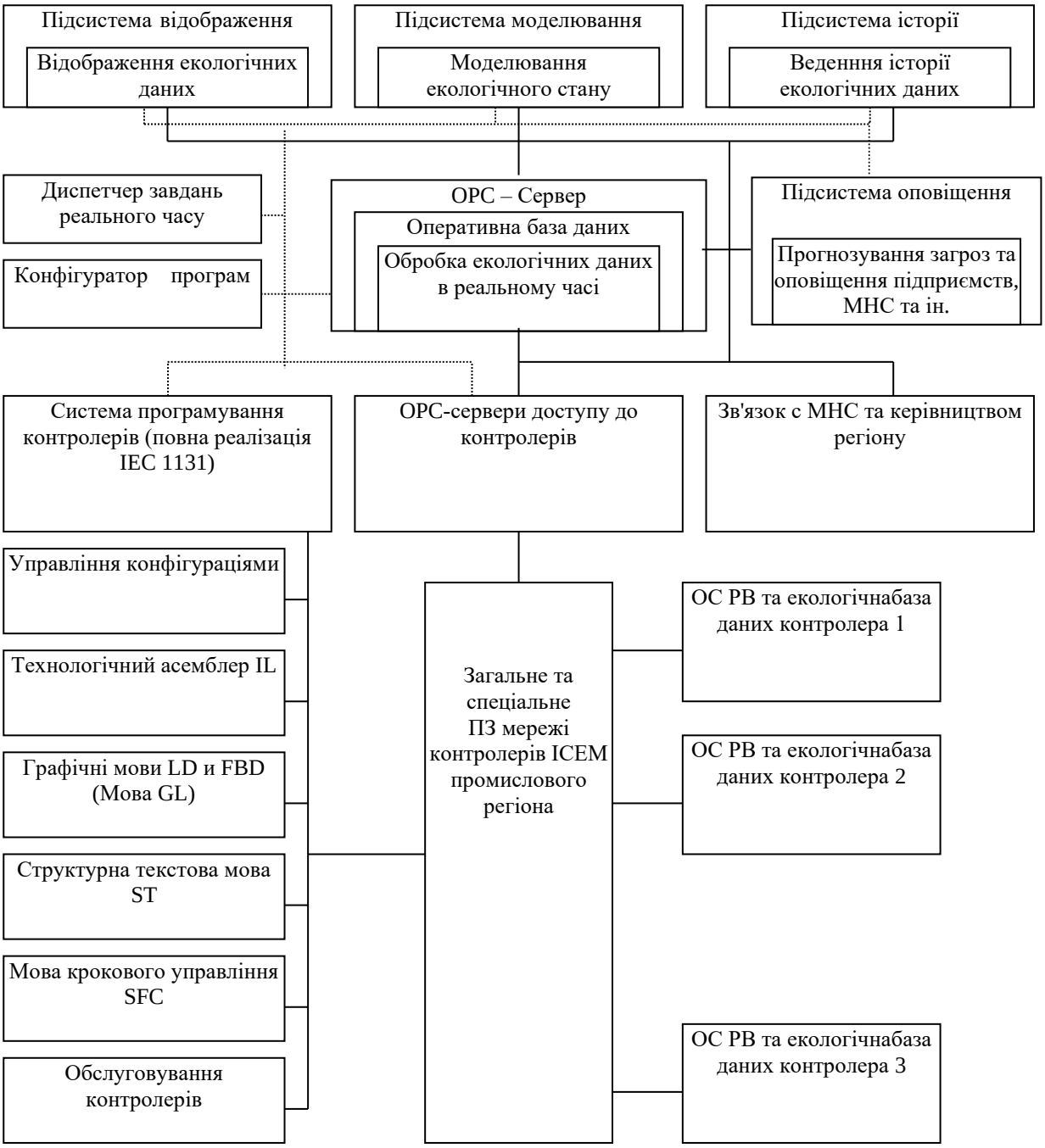


Рисунок 1.2 – ППП ICEM промислових регіонів

Основними функціями, які реалізує в загальному випадку SCADA – система на робочої станції центру моніторингу, є:

– прийом і обробка в реальному масштабі часу екологічних даних, що надходять з контролерів пунктів екологічного моніторингу;

– прогнозування загроз і оповіщення в разі потреби служб безпеки підприємств, служб МНС, керівників підприємств, міських управлінських служб;

– ведення історії екологічних даних для їх подальшого аналізу і виявлення тенденцій в екологічній обстановці регіону;

– моделювання розвитку екологічної обстановки в регіоні з метою виявлення і попередження можливих загроз у майбутньому;

– оперативне відображення на дисплеях поточного рівня забруднень навколишнього середовища на окремих ділянках і в усьому регіоні в цілому

Важливою проблемою розподілу обчислювальних ресурсів в ICEM є вибір завдань для виконання конкретними процесорами [8], [9]. При виборі завдань потрібно враховувати пріоритетність їх виконання та можливу необхідність паралельної реалізації декількох завдань.

1.3 Розподіл навантаження між комп'ютерними вузлами ICEM

Оптимальний розподіл програм між вузлами обчислювальної мережі ICEM, або балансування обчислювального навантаження, може підвищити продуктивність програмно-технічного комплексу в цілому за рахунок рівномірно розподіленого по всіх вузлах мережі навантаження. Такий розподіл зменшує навантаження на лінії зв'язку мережевої обчислювальної системи. Як було зазначено вище, мережа системи екологічного моніторингу складається з робочих станцій і контролерів. Рішення завдання закріплення програм та їх функцій за робочими станціями або за контролерами залежить від характеристик контролерів, а також від

топології і швидкості роботи мережевих засобів, що забезпечують обмін інформацією між контролерами і робочими станціями [22].

При проектуванні розподіленої обчислювальної мережі ICEM існує можливість обгрунтованого призначення операцій обробки для процесорів, розподілу прикладних програм, що реалізують ці операції, і масивів даних, які обробляються цими програмами, в запам'ятовуючих пристроях чи інших вузлах мережі. Якщо пов'язані операції виконуються в процесорах різних комп'ютерів, мережа передачі даних завантажується передачею відповідних масивів даних. При цьому кожен з масивів може зберігатися на комп'ютері, де формуються дані для нього, на комп'ютері, програмні модулі якого його використовують або, нарешті, на деякому іншому комп'ютері.

Доцільність того чи іншого рішення залежить, зокрема, від співвідношення частот поновлення масиву програмою, що його формує, і частот використання цього масиву іншими програмами, а також обмежень, що накладаються потужністю пам'яті використовуваних комп'ютерів.

Завдання розміщення є актуальним і в ході функціонування обчислювального комплексу, наприклад, при введенні в систему нових програмних модулів. Необхідність в перерозподілі програмних модулів і масивів даних може виникнути також при виявленні несправності окремих пристроїв обчислювальної системи. У цих випадках правильне рішення задачі розміщення програм і даних може збільшити час безперебійної роботи системи. Останнім часом пропонуються рішення задачі перерозподілу програм і даних між комп'ютерами обчислювальної мережі, що зводяться до лінеаризації булевих нелінійних моделей, до використання спеціально розроблених методів просторового динамічного програмування або ж рекурсивних чи покрокових алгоритмів обчислювального інтелекту, які застосовують на кожному кроці різні модифікації методу гілок і меж.

В цілому, відомі методи є ресурсоємними і вимагають для своєї

реалізації значних витрат процесорного часу, що утруднює їх використання в пакетах екологічних програм, що працюють в реальному масштабі часу.

В роботі [23] запропонований варіант рішення задачі статичного розподілу програм по вузлах обчислювального комплексу шляхом знаходження мінімального розрізу на графі, але в застосуванні до програм операційної системи, що розподіляються між центральними і периферійними процесорами обчислювальної системи. В силу особливостей функціонування програм операційної системи (робота в захищеному режимі, час роботи близько декількох мікросекунд, пряме управління центральними процесорами з використанням спеціальних шин) пропонований метод не може бути використаний для керуючих обчислювальних систем. До того ж, рішення задачі не доведено до працездатного алгоритму, який можна було б без додаткових теоретичних досліджень перенести на завдання розподілу програм у керуючих системах.

Таким чином, важливим є завдання розподілу навантаження між вузлами екологічної мережі, тобто розподілу програм моніторингу між робочими станціями і контролерами паралельного обчислювального комплексу.

1.4 Планування завдань розподілу обчислювальних ресурсів ICEM

При розробці пакетів екологічних програм для обчислювальних систем велике значення мають питання вибору дисципліни диспетчеризації (планування) завдань в комп'ютерних вузлах мережі ICEM. Оптимальна реалізація дисципліни диспетчеризації може істотно підвищити продуктивність обчислювального комплексу.

Диспетчеризація задач, тобто вибір завдань на виконання процесорами робочої станції є важливою проблемою. Доводиться

враховувати, що одночасна робота декількох процесорів з загальними ресурсами системи, такими, як лінії зв'язку, таблиці бази даних (БД), може привести до істотного зниження продуктивності ICEM.

При виборі дисципліни обслуговування завдань необхідно задовольняти таким вимогам:

– обслуговувати завдання вищого пріоритету в найкоротший час;

– обслуговувати завдання нижчого пріоритету в прийнятні для системи терміни;

– повніше завантажувати робочу станцію корисною роботою (від того, яка буде прийнята дисципліна обслуговування, залежить частота перемикання процесора з виконання однієї програми на іншу, а значить, і сумарна втрата часу на ці перемикання);

– зменшити середній час реакції робочої станції на зовнішні події;

– забезпечити відносну простоту реалізації обраної дисципліни диспетчеризації.

Перші дві вимоги є взаємовиключними, так як надання пільгових умов терміновим завданням здійснюється за рахунок завдань більш низьких пріоритетів. І навпаки, прагнення зменшити середній час обслуговування завдань низьких пріоритетів неминуче пов'язане (за інших рівних умов) з необхідністю скорочення переліку завдань, що мають більш високий пріоритет. У зв'язку з цим при виборі дисципліни диспетчеризації виникає задача знаходження компромісного рішення, що задовольняє в тій чи іншій мірі зазначеним вимогам.

Одна з проблем, яка виникає при виборі підходящої дисципліни обслуговування – це гарантія обслуговування. Потрібно гарантувати виконання завдань реального часу до терміну їх завершення. У деяких дисциплінах, наприклад, в дисципліні абсолютних пріоритетів, фонові процеси виходять обділеними багатьма ресурсами і, перш за все, процесорним часом. Виникає реальна дискримінація фонових завдань, в результаті чого вони досить тривалий час можуть не отримувати

процесорний час. Деякі процеси і завдання взагалі можуть бути не виконані до заданого терміну. Більш жорсткою вимогою до системи, ніж просто гарантоване завершення процесу, є його гарантоване завершення до зазначеного моменту часу або за вказаний інтервал часу.

Існує велика кількість методів диспетчеризації (планування і призначення) завдань, відповідно до яких формується їхня черга для виконання процесорами [24]. Розрізняють три основні класи дисциплін диспетчеризації: пріоритетні, безпріорітетні і комбіновані.

При реалізації пріоритетних дисциплін обслуговування окремим завданням надається переважне право перейти в стан виконання. При диспетчеризації відповідно до присвоєним кожному завданню пріоритетам все процесори виконують *M* найбільш пріоритетних завдань. У разі відносних пріоритетів поява нового завдання в списку готових до виконання не викликає переривання жодного з виконуваних завдань. Завдання виконуються до кінця або ж до переходу в стан очікування, а процесор, що звільнився, призначається для старшої за пріоритетом задачі зі списку готових до виконання. При абсолютних пріоритетах в разі появи нового завдання в списку готових до виконання аналізується його пріоритет.

Якщо пріоритет нового завдання виявляється більш високим, ніж пріоритети завдань, що виконуються всіма або хоча б деякими процесорами, процесор, який виконував найменш пріоритетне серед виконуваних завдання, переключається на нову задачу. Всі інші ситуації обробляються так само, як і в разі відносних пріоритетів. Як у випадку відносних пріоритетів, так і в разі абсолютних пріоритетів завдання з однаковими пріоритетами обслуговуються в порядку їх надходження.

У багатьох операційних системах реального часу використовуються методи диспетчеризації на основі абсолютних пріоритетів [25]. Це дозволяє скоротити час реакції системи, проте вимагає детального аналізу всієї системи для правильного присвоєння відповідних пріоритетів всім

виконуваним завданням з тим, щоб гарантувати їх обслуговування.

Для робочих станцій ICEM дисципліна диспетчеризації задач з абсолютними пріоритетами не може бути застосована, так як склад вирішуваних завдань в кожному базовому циклі може змінюватися, а, значить, обчислити заздалегідь їх пріоритети не можна. Диспетчеризація з динамічними пріоритетами вимагає додаткових витрат на обчислення значень пріоритетів виконуваних завдань. З огляду на те, що кожне із завдань верхнього рівня, що вирішуються на робочої станції, характеризується цілим набором конфліктних критеріїв, зведення яких до одного скалярного критерія (з динамічним пріоритетом) в кожному базовому циклі зажадає значних витрат процесорного часу, динамічні пріоритети для диспетчеризації набору завдань робочої станції також використовувати недоцільно.

При безпріоритетному обслуговуванні використовується так звана карусельна дисципліна диспетчеризації (Round Robin – RR) і пріоритетні методи обслуговування. Дисципліна обслуговування RR припускає, що кожна задача одержує процесорний час порціями або квантами часу (time slice) q. Після закінчення кванта часу q задача знімається з процесора і він передається наступній задачі. Знята задача ставиться в кінець черги задач, готових до виконання. Для оптимальної роботи системи необхідно правильно вибрати алгоритм, за яким кванти часу виділяються задачам.

Безпріоритетні дисципліни диспетчеризації оптимізовані для використання в системах пакетної обробки завдань і мультітермінальних системах і, природно, в чистому вигляді не можуть бути використані для обчислювальних систем екологічного моніторингу.

Останнім часом в операційних системах загального призначення в планувальниках завдань почала використовуватися комбінована дисципліна диспетчеризації, яка асоціює переваги пріоритетних і безпріорітетних алгоритмів диспетчеризації.

Операційні системи сімейства Windows останнім часом все частіше

використовуються на робочих станціях обчислювальних мереж в ІСЕМ. Щоб гарантувати виконання завдань верхнього рівня управління технологічним процесом в заданий час, ці завдання мають виконуватися з пріоритетами реального масштабу часу, тобто з абсолютними пріоритетами. У той же час не можна безпосередньо привласнити завданням робочої станції абсолютні пріоритети, так як склад вирішуваних завдань в кожному базовому циклі роботи робочої станції може змінюватися. Крім того, діапазон абсолютних пріоритетів (від 16 до 30) ОС Windows недостатній для диспетчеризації всіх завдань реального часу робочої станції керуючої обчислювальної системи.

Поряд з розглянутими трьома групами основних дисциплін диспетчеризації набув поширення підхід до побудови дисципліни диспетчеризації обчислювальних процесів за векторним критерієм, але без вимоги гарантованого часу обслуговування для кожної з задач, що є обов'язковою вимогою для диспетчеризації задач реального масштабу часу керуючих обчислювальних систем.

Поряд з розпаралелюванням обчислювального процесу на кілька одночасно функціонуючих процесів і потоків, ефективним засобом підвищення продуктивності є також векторизація обчислень на процесорах робочих станцій з глибокою конвеєризацією обчислювальних операцій або з векторним механізмом обробки, що включає в себе кілька паралельно працюючих процесорних елементів [26].

Для векторного процесора в одній команді в якості кожного операнда використовується не одне єдине значення, а відразу масив (вектор) значень. Нехай, наприклад, $A1$, $A2$ і P – три одновимірних масиви з однаковим числом елементів, причому $P = A1 + A2$.

Векторний процесор по одній команді виконує попарні складання елементів масивів $A1$ і $A2$ і присвоює отримані значення відповідних елементів масиву P. Кожен операнд при цьому може зберігатися в окремому векторному регістрі.

На противагу цьому, звичайному послідовному процесору довелося б багато разів виконувати операцію додавання елементів двох масивів, а також кількох команд, які організовують повторне виконання операцій додавання. Апаратна реалізація такої однієї векторної команди, зрозуміло, буде досить складною. Очевидно, що при вирішенні великого завдання за рахунок паралельної векторної реалізації її алгоритму можна досягти високої швидкості обчислень. Векторній обробці притаманні і інші важливі особливості. Кількість команд, необхідних для вирішення однієї і тієї ж програми, що використовує векторні операції, менше, ніж при виконанні на звичайному скалярному процесорі. Зменшення потоку команд дозволяє знизити навантаження на шини передачі даних, в першу чергу, між процесором і оперативною пам'яттю комп'ютера, що, в свою чергу, також може підвищити його продуктивність. Більш того, при відповідній організації оперативної пам'яті дані в процесор для векторної обробки можуть передаватися в кожному такті його роботи, що також може дати значний виграш в продуктивності комп'ютера. Одним із способів реалізації векторної обробки є конвеєрна обробка даних. При конвеєрній обробці даних складні операції розбиваються на безліч більш простих підопераций, які можуть виконуватися одночасно з підопераціями інших операцій. При русі об'єктів по конвеєру на різних його ділянках виконуються різні операції, а при досягненні кожною операцією кінця конвеєра вона виявиться повністю обробленою. Для ефективної роботи конвеєра мають виконуватися наступні умови:

– система виконує повторювані операції;
– ці операції можуть бути розділені на незалежні підоперації;
– трудомісткість підопераций приблизно однакова.

Кількість підопераций називають глибиною конвеєра. Важливою умовою нормальної роботи конвеєра є відсутність конфліктів, тобто дані, що подаються в конвеєр, повинні бути незалежними. У тому випадку, коли черговий операнд залежить від результату попередньої операції,

виникають такі періоди роботи конвеєра, коли він порожній. При скалярній обробці це істотно знижує продуктивність конвеєрних систем.

В конвеєрах команд також можуть виникати простої, джерелом яких є залежність між командами. Такі ситуації виникають при наявності в програмах не распаралелених циклів і умовних розгалужень. Конвеєрізація ефективна тільки тоді, коли завантаження конвеєра близьке до повного, а швидкість подачі нових операндів відповідає максимальній продуктивності конвеєра. Якщо відбувається затримка, паралельно буде виконуватися менше операцій і сумарна продуктивність знизиться. Векторні операції забезпечують ідеальну можливість повного завантаження обчислювального конвеєра. Одним з найбільш використовуваних способів збільшення завантаження конвеєра є векторне розпаралелювання програм компіляторами. Існують наступні основні види розпаралелювання циклів, які використовуються сучасними компіляторами:

– розворот циклів, що полягає в багаторазовому дублюванні циклу. Причина – мікропроцесори з конвеєрною архітектурою (а до них відносяться всі сучасні процесори) погано пристосовані для роботи з циклами. Для досягнення найвищої продуктивності процесору необхідна досить протяжна ділянка «траси виконання», вільна від розгалужень;

– фальцювання циклів (fold loops). Зовні дуже схоже на розворот, але переслідує зовсім інші цілі, а саме – збільшення кількості потоків даних на одну ітерацію. Чим вище щільність (strength) потоків, тим вище паралелізм обробки даних, а, значить, і швидкість виконання циклу. На відміну від розвороту, виконується практично всіма компіляторами мови C++. Відмінність від розвороту – цикл не розгортається повністю, а тільки збільшується кількість операцій в одній ітерації циклу;

– безпосередня векторизація. В сучасних процесорах реалізується підтримка векторних команд. Так як в стандарті мови C++ векторні операції відсутні, то всі дії з векторизації циклів здійснює компілятор. Він

повинен проаналізувати вихідний код і визначити, які операції саме можуть виконуватися паралельно.

Компілятор не може распаралелити цикл, як правило, з таких причин:

– в тілі циклу зустрічається умовний оператор;
– в тілі циклу зустрічається оператор безумовного переходу;
– в тілі циклу зустрічається оператор переривання циклу;
– в тілі циклу зустрічається виклик функції;
– в тілі циклу зустрічається оператор введення / виведення.

У цих випадках, як правило, цикл не є векторною операцією і, отже, не може бути векторизованим. Саме для розпаралелювання таких (а також складніших) циклів потрібні засоби, що зводять їх до більш простих циклів, які розпізнаються компиляторами, і зводяться до програмно організованих векторних операцій, що підвищує завантаження конвеєра центрального процесора робочої станції керуючої обчислювальної системи. Одним з таких засобів для програмного векторного розпаралелювання обробки даних є шаблон класів valarray з бібліотеки STL мови програмування C ++ . Але слід відзначити, що цей шаблон не є універсальним і здатний обробляти елементи векторів тільки по безгілковим послідовностям операцій, зведення задач до яких вимагає дуже специфічного програмування. До того ж, навіть якщо вдається звести задачу до використання методів цього шаблону, заповнення конвеєра переривається на ділянках переходу від однієї розмірності векторів до іншої. В даний час відсутні універсальні засоби програмної векторної обробки даних, що дозволяють ефективно завантажувати конвеєр процесора робочої станції.

У зв'язку з цим доцільно розробити алгоритм диспетчеризації задач за векторним критерієм для задач реального масштабу часу робочих станцій обчислювальних систем.

1.5 Постановка задач дослідження

Проведений аналіз дозволяє сформулювати мету і завдання кваліфікаційну роботи. Відповідно до завдання в даній роботі необхідно розробити та дослідити алгоритми і програми диспетчеризації обчислювальних ресурсів для інтелектуальної системи екологічного моніторингу.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– аналіз існуючих методів розподілу обчислювальних ресурсів в системах екологічного моніторингу;

– дослідження методу диспетчеризації вузлів мережі ІСЕМ, що здійснює оптимальний розподіл завдань між робочими станціями і контролерами системи екологічного моніторингу;

– дослідження методів пріоритетного планування завдань ІСЕМ з метою мінімізації часу виконання завдань і збільшення пропускної спроможності системи;

– розробка методу формування мережі пунктів спостереження з застосуванням графової моделі кластеризації;

– дослідження програмних засобів векторної обробки даних екологічного моніторингу, що ефективно завантажують конвеєр процесора ІСЕМ за рахунок зменшення його простою через розгалуження обчислень за умовами;

– програмна реалізація та моделювання задач розподілу обчислювальних ресурсів інтелектуальної системи екологічного моніторингу.

2 МЕТОДИ ДИСПЕТЧЕРИЗАЦІЇ ОБЧИСЛЮВАЛЬНИХ РЕСУРСІВ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ЕКОЛОГІЧНОГО МОНІТОРИНГУ

2.1 Опис загальної структури мережевого комплексу ICEM

Керуючу обчислювальну систему екологічної та техногенної безпеки регіону можна представити як мережевий комплекс, який об'єднує вимірювальні пристрої та контролери пунктів моніторингу, робочі станції центру моніторингу між собою, а також з рівнем управління регіоном [8].

Робочі станції займаються обробкою, поданням екологічної інформації, її архівуванням і аналізом. Вони також виконують статистичні та оптимізаційні розрахунки і ведуть моніторинг стану всіх програмно-технічних засобів, що входять в обчислювальний комплекс.

Робочі станції, що використовуються в системах управління та інформаційних системах, зазвичай містять: шасі, генмонтажну плату, системну плата, дискові накопичувачі, контролери введення – виведення, пристрої введення, контролери зв'язку, пристрої відображення, мережеве обладнання, джерело живлення, пристрій безперебійного живлення, стіл , тумби, шафу. Як приклад розглянемо конфигуратор робочої станції ПС 5110, що входить до складу ПТК МСКУ М [21] Виконання можуть відрізнятися складом обладнання та його компоновкою в конструктивах. У ПС 5110 можуть входити від 1 до 3 системних блоків. Системний блок – основний пристрій робочої станції ПС 5110. Системний блок містить в своєму складі блок процесорний, який управляє функціонуванням робочої станції ПС 5110.

Системний блок забезпечує можливість підключення до нього інших пристроїв, що становлять робочу станцію ПС 5110, а також ліній зв'язку з іншими обчислювальними системами. Системні блоки компонуються на основі шасі промислового комп'ютера, в яке встановлюється джерело

живлення, генмонтажна плата, блок процесорний, контролери та дискові накопичувачі. Генмонтажна плата системного блоку може мати до 14 слотів системних інтерфейсів PCI і ISA в різних комбінаціях. Максимальна кількість встановлюваних в системному блоці дискових накопичувачів дорівнює 5. Блок процесорний має в своєму складі основні системні пристрої: процесор, набір системних мікросхем (chip set), слоти для установки модулів оперативної пам'яті, інтерфейси дискових накопичувачів (EIDE, FDD), інтерфейси пристроїв введення - виведення, watch-dog timer, DiskOnChip тощо. У разі необхідності, блок процесорний може мати інтерфейс Ultra 160 SCSI (тільки для підключення HDD) і видеоконтроллер з шиною AGP (відеопам'ять до 32 MB). Шасі являє собою сталевий високоміцний корпус з алюмінієвою передньою панеллю і двома вентиляторами зі змінними фільтрами. У шасі встановлюється PICMG – генмонтажна плата на 14 слотів з інтерфейсом PCI / ISA і PS / 2 – джерело живлення. Генмонтажні плати – чотирьохшарові з внутрішніми шарами ланцюгів живлення і землі (у вигляді екранів), а також з інтерфейсами PCI rev. 2.1 і ISA IEEE P996. В платах присутня індикація наявності номіналів живлення, а також забезпечена підтримка джерел живлення AT і ATX. Блок процесорний робочої станції базується на процесорі з архітектурою Intel з глибиною конвеєра до 25 ступенів. У робочу станцію центру моніторингу за допомогою телефонних каналів зв'язку надходить з контролерів пунктів моніторингу після первинної обробки екологічна інформація, яка служить основою для подання реального стану екологічного стану регіону на різного роду мнемосхемах, реєстрації історії екологічного моніторингу, статистичних розрахунків, а також прогнозних розрахунків і розрахунків виникаючих загроз. Крім того, робочі станції спільно з контролерами забезпечують моніторинг стану всього комплексу програмно-технічних засобів. Реалізацію функцій нижнього рівня системи управління екологічною безпекою регіону забезпечують контролери. Контролери, що входять до складу обчислювальної мережі, виконують

функції введення інформації з аналогових і дискретних вхідних каналів зв'язку з об'єктом, первинної переробки цієї інформації (лінеарізацію, масштабування, приведення до відповідних фізичних одиниць, контроль на достовірність) і виявлення загроз і реєстрацію подій. Використовуючи отриману з датчиків екологічну інформацію про об'єкт управління, контролери можуть виконувати також керуючі функції нижнього рівня.

Контролери різняться між собою за ступенем швидкодії, об'єму оперативної пам'яті, кількості каналів введення-виведення, інтерфейсів, характеристиками надійності, наявного математичного забезпечення.

Сучасні контролери діляться на 2 основних типи: незмінні і модульні. Незмінні контролери поставляються як самостійні пристрої з процесором, джерелом живлення і фіксованою кількістю каналів дискретного і аналогового вводу / виводу. Незмінні контролери можуть мати окремі взаємопов'язані компоненти для розширення, є значно меншими за розмірами, вони дешевше і простіше в установці. Модульні ж контролери мають більшу гнучкість, мають широкі можливості по вводу / виводу, обсягам процесорної пам'яті, вхідним напруженням, типу і кількості каналів зв'язку тощо. Сучасні ПЛК обробляють дискретні і аналогові сигнали, що надходять від датчиків, кінцевих вимикачів, пультів оператора і інших пристроїв введення відповідно до заданих алгоритмів. Вони формують дискретні і аналогові вихідні сигнали для виконавчих механізмів, таких, наприклад, як соленоїди, приводи клапанів, засувки тощо. У багатьох випадках технологічний процес не потребує великої обчислювальної потужності системи управління. Ядром апаратури управління цілком може бути процесор, що підтримує адресацію до пам'яті, що є достатньою для розміщення до 20000 рядків керуючої програми і зберігання великого обсягу даних. Контролери на базі 32-розрядних мікропроцесорів, так само як і робочі станції, забезпечують розпаралелювання обчислень за рахунок конвеєризації обробки алгоритмів збору даних і управління. Кількість, функції та місця розташування

контролерів в пунктах збору екологічних даних ІСЕМ, які суттєво впливають на загальний процес розподілу завдань в ІСЕМ, мають бути визначені в рамках окремого завдання побудови ІСЕМ.

2.2 Розподіл навантаження між контролерами і робочою станцією системи екологічного моніторингу

Так як між робочою станцією центру моніторингу і віддаленими на великі відстані закритими необслуговуваними контролерами пунктів екологічного контролю використовуються повільні комутовані телефонні лінії зв'язку, була поставлена задача зменшення часу взаємодії та обсягу інформації, що передається по лініях зв'язку, а також мінімізації навантаження на контролери з метою підвищення надійності їх функціонування. Розглянемо пропонований метод оптимального розподілу програмних модулів між робочою станцією і контролером. Для пари «робоча станція – контролер» завдання ставилося і вирішувалося наступним чином.

Припустимо, що є проста обчислювальна мережа, що складається з однієї робочої станції, що включає в себе один або кілька центральних процесорів, що працюють на загальне поле оперативної пам'яті, і один контролер, який має свій процесор, свою локальну пам'ять і доступ до робочої станції по лінії зв'язку. Процесори робочої станції можуть обмінюватися інформацією між собою зі швидкістю виконання ними основних операцій. Відомий також склад програмних модулів, їх взаємозв'язки один з одним, а також витрати часу процесора на роботу кожного модуля і на кожен тип взаємодії модулів за умови, що ці модулі розміщені у вузлах мережі різних типів. Потрібно так розподілити модулі комплексу програм між робочою станцією і контролером, щоб витрати часу на виконання програм контролером і на взаємодію програм контролера і робочої станції були мінімальними. У поставленому завданні

потрібно визначити, які програмні модулі необхідно винести в робочу станцію, а які залишити для виконання в контролері. Для вирішення завдання розподілу програмних модулів між робочою станцією і контролером доцільно застосувати алгоритм пошуку максимального потоку (мінімального розрізу в мережі), заснований на побудові графа взаємозв'язків між програмними модулями. Наведемо основні ітерації цього алгоритму.

Крок 1. Комплекс програм представимо у вигляді неорієнтованого зв'язного графа $G = (U, V)$ з множиною вузлів $V = \{v_1, v_2, \ldots v_n\}$, кожен з яких відповідає окремій програмі, і множиною ребер $U = \{u_1, u_2, \ldots u_m\}$, що відповідають зв'язкам між програмами. Кожному ребру $[v_i, v_j]$ відповідає вага $c(i,j)$, що дорівнює витратам часу контролера на взаємодії між відповідними програмами за умови, що одна з них виконується робочою станцією, інша – контролером.

Крок 2. До графу додамо два додаткових вузла: *К* та *РС*, що формально представляють контролер і робочу станцію відповідно.

Всі вузли графа, що представляють програми, з'єднаємо ребрами зі знову введеним вузлом *РС*, і кожному з них пріпішем вагу, рівну витратам часу контролера на роботу відповідної програми. Вузлам відповідних програм, які повинні обов'язково виконуватися робочою станцією, поставимо у відповідність ваги, що дорівнюють нескінченості: $c(i, PC) = \infty$.

Крок 3. З вузлом *К* (контролер) з'єднаємо тільки ті вузли графа, які представляють програми, що використовують спеціальне обладнання контролера або ж виконують специфічні функції, які повинні завжди виконуватися контролером. Вагу таких ребер приймаємо рівною нескінченності: $c(K,j) = \infty$.

Крок 4. Отриманий в результаті розширення граф взаємозв'язків між програмами будемо розглядати як транспортну мережу з джерелом *К* і стоком *РС*. Таким чином, початкове завдання можна звести до задачі знаходження мінімального розрізу мережі.

Зв'язки комплексу програм можна представити у вигляді квадратної матриці, в якій одиницями позначаються зв'язки між двома програмними модулями. У рядку *К* (контролер) одиницями позначаються програми, для виконання яких потрібне спеціальне обладнання контролера. Приклад такої матриці для 7 модулів представлений у таблиці 2.1.

Таблиця 2.1 – Зв'язки між програмними модулями

0	1	2	3	4	5	6	7	РС	К
1	0	0	0	1	0	0	0	1	0
2	0	0	0	0	1	0	0	1	0
3	0	0	0	0	1	0	0	1	0
4	0	0	0	0	0	1	1	1	0
5	0	0	0	1	0	0	0	1	0
6	0	0	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	1	0
РС	0	0	0	0	0	0	0	0	0
К	1	1	1	0	0	0	0	0	0

Наведену матрицю можна вважати матрицею інціденцій для графа, що підлягає побудові. Сам граф для семи програмних модулів буде виглядати так, як показано на рисунку 2.1.

Після рішення задачі знаходження мінімального розрізу в такому графі програмні модулі між контролером і робочою станцією розподіляються таким чином, щоб сума часів виконання програм контролером і обміну між контролером і робочою станцією геоінформаційної системи була мінімальною.

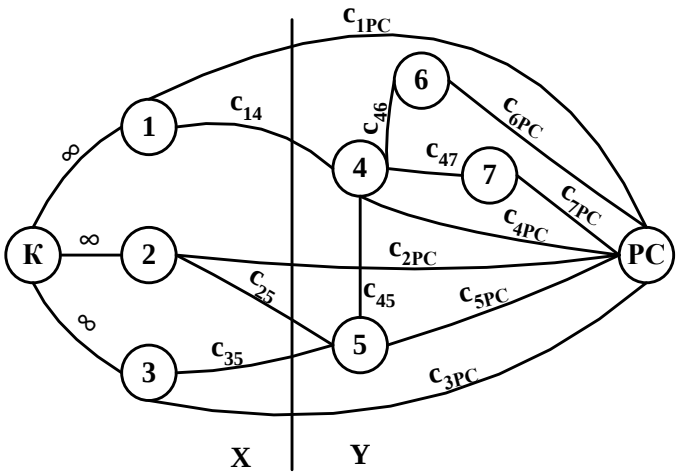


Рисунок 2.1 – Граф взаємозв'язків між програмами для мінімізації часу виконання програм контролером

Дійсно, будь-який розріз модифікованого графа взаємозв'язків програм, що розділяє вузли N_s та N_t , включає в себе ребра двох типів: ребра, що задають взаємозв'язок між програмами, і ребра, що визначають витрати на виконання програм контролером.

Якщо програми, представлені вузлами множини *Y*, виконуються робочою станцією, а програми, представлені вузлами множини *X*, виконуються контролером, тоді в розріз входитимуть тільки ті ребра, які представляють взаємозв'язки між програмами, що відносяться до робочої станції і програмами, які належать до контролера, а також ребра, що представляють витрати часу контролера на виконання всіх програм системи ($t = t_1 + t_2 + \ldots + t$).

У розріз не входить жодне ребро, що задає зв'язок між програмами, розташованими в одному процесорі, і жодне ребро, яке вказує витрати часу на виконання програми, що стосується робочої станції.

Так як сумарні витрати часу контролера геоінформаційної системи екологічного монніторингу на роботу програм, розподілених між робочою станцією і контролером, є сумою витрат на роботу всіх програм, що відносяться до контролера, і витрат на взаємодію цих програм з програмами з робочої станції, то сума ваг ребер розрізу (X, Y) як раз і буде

дорівнювати цим сумарним витратам.

Відповідно до викладеного вище алгоритму розподілу обчислювальних ресурсів були проведені реальні розрахунки для програмного забезпечення системи екологічної та техногенної безпеки регіону, перелік програмних модулів якого представлений у таблиці 2.2.

Таблиця 2.2 – Склад основних модулів системи екологічного моніторингу регіона

№ модуля	Позн.	Призначення програмного модуля	Закріплення за робочою станцією (РС) або контролером (К)
1	ВВАІ	Підсистема ввода – виведення аналогової информации	К
2	ВВДІ	Підсистема ввода – вывода дискретної інформації	К
3	ПОЕІ	Підсистема первичної обробки екологічної інформації, що надходить з аналогових та дискретних датчиків	Ні
4	ФАВ	Програмний модуль фіксації аварійних викидів	Ні
5	ОУ	Програмна підсистема виявлення екологічних та техногенних загроз	Ні
6	УГЗ	Програмна підсистема управління газоаналізаторами	Ні
7	АОК	Модуль автоматичного опитування екологічних даних з контролерів пунктів моніторингу	Ні
8	АОРС	Адаптер рівня автоматизації інтерфейсу ОРС	Ні
9	ОРС	Сервер ОРС, що забезпечує зв'язок між основними оброблюваючими програмами та підсистемами нижнього рівня	Ні
10	ОЕІ	Обробка екологічної інформації в реальному часі	Ні
11	СОЕІ	Модуль статистичної обробки екологічної інформації	Ні
12	ВЕД	Програмний модуль відображення екологічних даних	РС
13	МЕС	Програмна підсистема моделювання екологічного стану регіона	РС
14	ІЕД	Програмний модуль ведення історії екологічних даних	РС

У пакеті в якості мережевої технології взаємодії програм на робочих станціях і в контролерах після аналізу результатів тестування технологій CORBA, DCOM, і .NET Remoting будемо використовувати мережеву технологію DCOM. Балансування навантаження на обчислювальні вузли системи екологічної та техногенної безпеки регіону доцільно виконувати методом мінімізації часу виконання програм контролером [27].

Для розподілу програмних модулів пакета між контролером і робочою станцією з метою мінімізації часу виконання програм контролером побудуємо таблицю зв'язків між модулями у вигляді квадратної матриці і доповнимо її зв'язками кожного програмного модуля з вузлом РС (табл. 2.3).

Таблиця 2.3 – Матриця зв'язків програмних модулів пакета с вузлом РС

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	К	РС
1	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	1
2	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
6	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	1	1	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1
10	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
К	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
РС	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Відповідно до отриманої матриці побудуємо граф зв'язків, в якому ваги ребер рівні часу виконання програмних модулів контролером (рисунок 2.2).

Для наочності вузли графа на рисунку 2.2 позначені не номерами модулів, а їх абревіатурами; крім того, ребра взаємозв'язків між модулями представлені суцільними лініями, а ребра, що з'єднують програмні модулі з вузлом РС, – пунктирними лініями.

Крім ваг ребер, що представляють взаємозв'язки програмних модулів, ребрах графа, що з'єднують програмні модулі з вузлом РС, призначимо ваги, рівні середнім часам виконання цих модулів контролером. Як і для взаємозв'язків, ці часи (в мілісекундах) наведемо для одного базового циклу роботи системи управління, що дорівнює 1 секунді. При цьому часи виконання модулів розрахуємо для роботи програмних модулів в системі управління, що має 128 аналогових і 64 дискретних входів, а також 16 аналогових і 32 дискретних виходів.

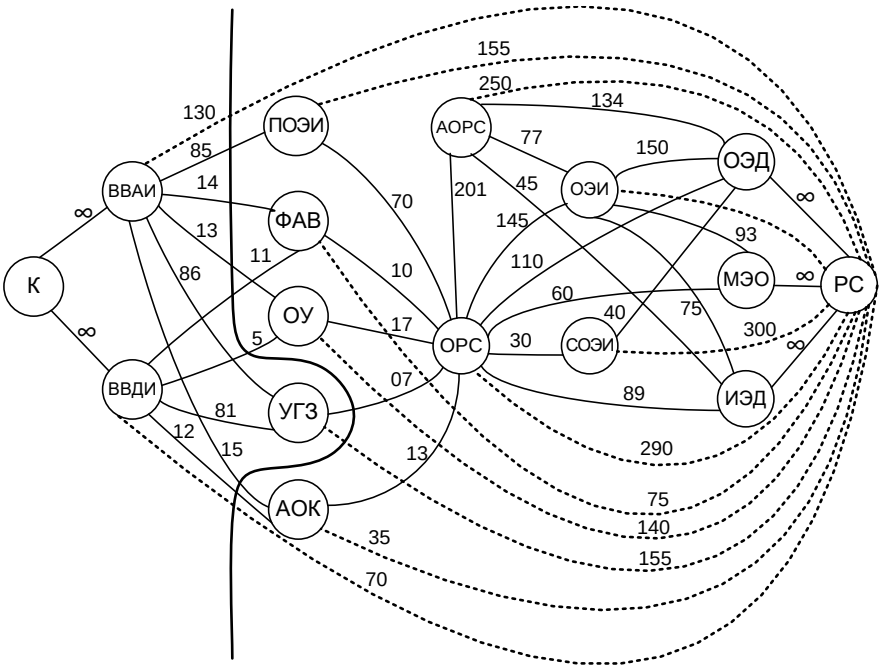


Рисунок 2.2 – Граф зв'язків програмних модулів з вузлом РС

Якщо наведений на рисунку 2.1 граф розглядати як мережу з витоком *К* та стоком *РС*, і відповідно до методу мінімізації взаємодій знайти максимальний потік (мінімальний розріз цієї мережі), то отримаємо розріз, представлений лінією, що перетинає граф зверху донизу. Таким чином, розглянутий метод залишає в контролері лише заздалегідь закріплені за ним підсистеми введення - виведення аналогової і дискретної інформації, а також пов'язану з цими підсистемами програмну підсистему управління газоаналізаторами. Підсумовування ваг всіх дуг розрізу показало, що максимальне завантаження контролера в одному базовому циклі роботи системи управління склала 517 мілісекунд, тобто контролер буде завантажений приблизно наполовину. Це означає, що при необхідності можна практично в два рази збільшити кількість вхідних і вихідних каналів, що обслуговуються контролером. Фактичне ж час виконання програм контролером при цьому в середньому зменшилася на 26% (рисунок 2.3).

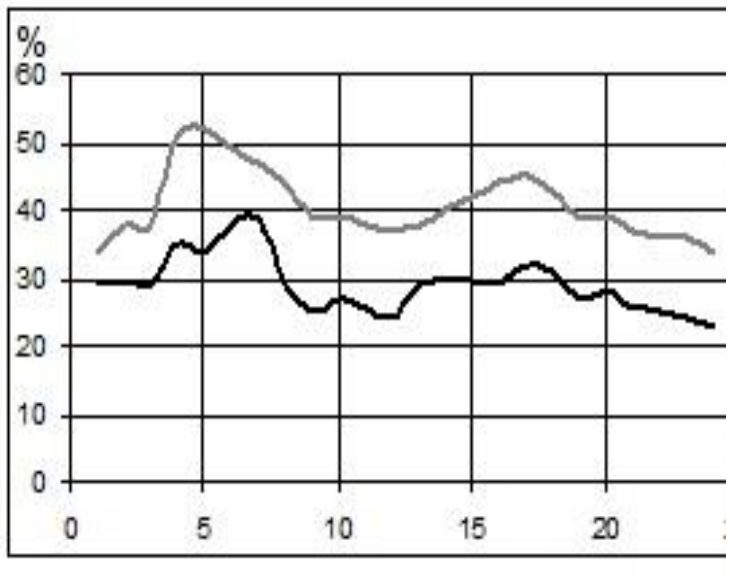


Рисунок 2.3 – Зміна часу виконання програм контролером при використанні базового (верхній графік) і запропонованого (нижній графік) методів

При цьому потрібно враховувати, що завантаження робочої станції після перерозподілу модулів збільшиться. Отримані результати по завантаженню робочої станції до і після перерозподілу програмних модулів представлені на графіку (рис. 2.4).

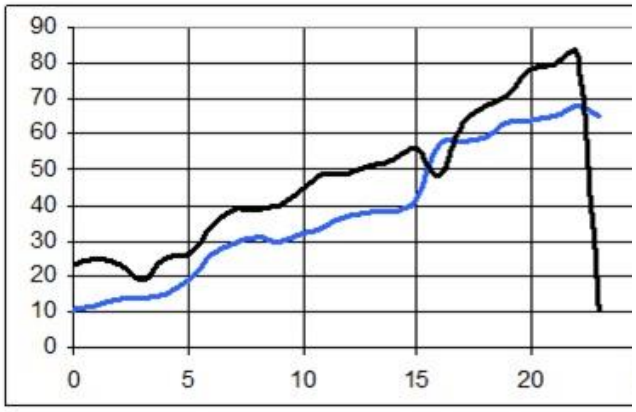


Рисунок 2.4 – Ступінь завантаження процесора робочої станції при використанні запропонованого методу розподілу модулів

Як можна побачити з представлених результатів, ступінь завантаження робочої станції після перерозподілу програмних модулів збільшилася менше, ніж на 26%, через більшу потужність процесора PC в порівнянні з процесором контролера. Відсоток завантаження робочої станції при ручному розподілі програмних модулів в середньому виріс лише на 11%.

2.3 Оптимальне планування пріоритетної реалізації завдань ICEM

Важливою відмінною рисою управління обчислювальним процесом в системах реального масштабу часу, до яких належить система екологічного і техногенного моніторингу, є те, що для всіх підлягаючих виконанню завдань відомі, як правило, всі основні характеристики, а також системні ресурси, які використовуються під час їх роботи. Можна підвищити ефективну продуктивність цієї системи, якщо для планування

робіт (завдань) використовувати алгоритми, що забезпечують порівняно з найпростішими алгоритмами планування більш повне завантаження процесора і мінімізують втрати продуктивності системи через колізії, що виникають при одночасній роботі із загальними системними ресурсами.

Більшість дисциплін планування вимагають лінійного упорядкування пріоритетів всіх завдань, які підлягають виконанню, що для складних систем реального масштабу часу може виявитися недоцільним, так як для встановлення пріоритету завдань потрібно враховувати кілька непорівняних між собою показників. У той же час можливість паралельного виконання в ОС Windows кількох завдань дозволяє перейти від їх лінійного упорядкування за одним скалярним критерієм до структурного, тобто до диспетчеризації завдань відразу по групі критеріїв. При цьому черговість встановлюється не між окремими завданнями, а між групами рівноцінних завдань. Такий підхід не дає ніяких переваг, якщо для впорядкування черги завдань є тільки один числовий показник, але відкриває нові можливості, якщо таких показників декілька.

На робочій станції центру моніторингу вирішуються десятки завдань верхнього рівня, таких як обробка екологічних даних в реальному часі, моделювання екологічної обстановки в регіоні, реєстрація значень екологічних параметрів, виявлення і прогнозування техногенних загроз, оптимізаційні задачі, диспетчерські завдання.

Період виконання таких завдань різний (деякі з них виконуються раз в день, раз на місяць, раз в квартал; інші виконуються кожну годину, кожну хвилину, кожну секунду). При цьому потрібно враховувати, що система економічного і техногенного моніторингу – це система реального часу, де для кожного завдання задано, протягом якого інтервалу часу вона повинна бути виконана. Тому небажано використовувати динамічний перерахунок пріоритетів, так як на нього витрачається чимало часу, а також зменшується ймовірність виконання всіх завдань у визначені терміни.

Варіант безпосереднього використання диспетчера задач, вбудованого в ОС, що поставляється разом з комп'ютером, може виявитися також неприйнятним, так як такий диспетчер не враховує особливості роботи з комплексами задач систем управління. Наприклад, в Windows диспетчер задач постійно перераховує пріоритети інтерактивних процесів (пріоритети 0 – 15), так як пріоритет процесу повинен підвищуватися при взаємодії з користувачем і зменшуватися, якщо користувач з процесом не взаємодіє. В системі реального часу пріоритети завдань потрібно перераховувати тільки в разі зміни набору завдань.

У диспетчерів завдань ОС Windows для вибору завдання на виконання використовуються багаторівневі черги зі зворотним зв'язком. Цей метод використовується також в ОС сімейства Linux, де задані пріоритети завдань від 0 до 31 і встановлені 32 черги завдань з різними пріоритетами відповідно. Першими виконуються завдання з черги з найвищим пріоритетом, що може привести до постійного відкладання виконання фонових завдань, якщо в більш високопріоритетні черги будуть регулярно надходити нові завдання. В системі реального часу це неприпустимо, тому що є жорсткі тимчасові рамки на терміни запуску і завершення виконання кожного завдання.

Ще одна відмінність способів управління завданнями в системі реального часу і в ОС загального призначення полягає в тому, що в системі реального часу абсолютні пріоритети завдань не змінюються в залежності від подій, що відбуваються в системі. Тільки якщо склад завдань змінився, потрібно перераховувати їх відносні пріоритети.

В ОС сімейства Windows пріоритет підвищується при виході потоку зі стану очікування, при користувацькому введенні для вікна потоку, а також якщо потік не виконувався тривалий час. Пріоритет зменшується на 1, якщо потік використовував свій квант часу повністю. Диспетчер завдань ОС здійснює перепланування потоків, коли відбувається будь-яка з цих подій [14]. В системі управління перепланування повинно відбуватися,

коли змінюється склад завдань – це відбувається значно рідше, ніж перераховані вище події.

Безпріоритетні дисципліни обслуговування в системах управління використовувати важко, так як для задач реального часу зазвичай задається максимальний час відгуку. При виборі завдання на виконання його треба враховувати в першу чергу. У зв'язку з цим найпростіші безпріорітетні дисципліни обслуговування (такі, як SJN, SRT, FCFS) не слід розглядати як основу диспетчеризації задач в системах управління. У сучасних ОС повсюдно використовуються пріоритетні модифікації RR – кругового алгоритму диспетчеризації. Для кругового виконання використовуються кілька черг завдань з різними пріоритетами (багаторівневі черги зі зворотним зв'язком).

Для виконання завдань в режимі квантування часу вибирається непорожня черга з найвищим пріоритетом. Величина кванта часу q повинна бути обрана з урахуванням того, що більшість завдань мають бути виконані за один квант часу, щоб не витрачати ресурси на повернення до задачі ще раз. Величину кванта часу вибирають, як правило, в межах 0.02 – 0.05 сек, оскільки цей час більше часу виконання більшості службових завдань програмного комплексу.

У сучасних ОС використовується багатозадачність з витісненням, так як завжди існують першорядні за важливістю завдання, яким потрібно дати можливість перервати виконання завдань з меншим пріоритетом. Якби використовувалася багатозадачність без витіснення, то завдання повинні були б самі віддавати управління процесору через невеликий проміжок часу. Однак немає впевненості, що всі завдання написані коректно і будуть вчасно повертати управління, тому у всіх сучасних системах реального часу використовуються саме дисципліни диспетчеризації з витісненням. Крім того, багатозадачність без витіснення не підходить при роботі в режимі реального масштабу часу, так як оперативність виконання багатьох завдань стає залежною від одного,

найбільш тривалого за часом виконання завдання.

Очевидно, що в методі планування завдань системи екологічного і техногенного моніторингу потрібно по можливості зменшити час реакції робочої станції на події на об'єкті і операторські запити, збільшити її продуктивність. Щоб збільшити продуктивність, тобто збільшити кількість завдань, що виконуються в одиницю часу робочою станцією, слід зменшити час простою завдань в очікуванні звільнення будь-якого ресурсу іншим завданням (файлу, таблиці БД). Для цього потрібно не запускати одночасно завдання, що використовують одні і ті ж ресурси. При цьому спочатку слід виконувати завдання, що вимагають менше часу для їх виконання і менше часу зайнятості каналів зв'язку. Відзначимо, що користувач не в змозі врахувати подібні обмеження при ручному призначенні пріоритетів обчислювальним процесам в системі управління.

Таким чином, для ефективного розподілу ресурсів в системі екологічного і техногенного моніторингу треба розробити метод планування виконання завдань, що максимізує завантаження робочої станції і збільшує її ефективну продуктивність за рахунок того, що при плануванні завдань на виконання повинні враховуватися їх комплексні (структурні) характеристики, а, значить, завдання, що одночасно виконуються, будуть втрачати менше часу в чергах очікування спільно використовуваних ресурсів.

Пропонований метод планування задач за структурним критерієм полягає в наступному. Показниками, що характеризують завдання, можуть бути: період повторення завдання, обсяг оперативної пам'яті для виконання завдання, час його вирішення, інтенсивність роботи з таблицями баз даних, час зайнятості каналів зв'язку для отримання значень екологічних параметрів від OPC - серверів тощо. Може бути встановлений порядок вирішення завдань по кожному з показників, але інформація, необхідна для порівняння цих показників між собою, як правило, відсутня. Одна задача може вимагати великого часу для роботи з таблицями баз

даних, малого часу зайнятості каналів зв'язку і середнього часу рішення, інша – малого часу для роботи з таблицями баз даних, великого часу зайнятості каналів зв'язку і значного часу рішення.

Розглянемо алгоритм порівняння завдань, що характеризуються декількома числовими властивостями. Для кожного завдання можна занести значення всіх його властивостей в спеціальну структуру [26], яка називається блоком управління завданням (*tcb*). При цьому список всіх завдань, які обробляються алгоритмом планування, буде представлятися масивом структур розмірності *N* (*tcn* [*N*]), де *N* – число всіх завдань.

Будемо вважати, що завдання *tcn* [*i*] строго краще завдання *tcn* [*k*], якщо завдання *tcn* [*i*] перевершує завдання *tcn* [*k*] хоча б по одній властивості (*tcn* [*i*] *.j> tcn* [k] *.j*), а по всіх інших не гірше неї. Наприклад, нехай завдання *tcn* [1] характеризується наступними властивостями: число звернень до таблиць бази даних історії – 14, час рішення – 9 секунд, завдання *tcn* [2] – 14 і 6 секунд відповідно. І нехай «кращим» вважається те завдання, у якому число звернень до таблиць баз даних і час вирішення менше. Тоді завдання *tcn* [2] буде строго краще завдання *tcn* [1], так як за другою властивості воно краще, а значення першоъ властивості у обох завдань однакові. Будемо вважати, що завдання *tcn* [*i*] і *tcn* [*k*] непорівнянні між собою, якщо завдання *tcn* [*i*] перевершує завдання *tcn* [*k*] за значеннями одних властивостей, а завдання *tcn* [*k*] перевершує завдання *tcn* [*i*] за значеннями інших. Наприклад, *tcn* [1]= {14, 8} і *tcn* [2] = {16, 7} непорівнянні між собою. Незалежно від того, чи вважається «кращим» більше значення властивості або менше, порівнювати ці завдання без додаткової інформації не можна. Відсутність вимоги лінійного упорядкування завдань в списку дозволяє об'єднати деякі незрівнянні і еквівалентні завдання в одну групу і цій групі привласнити номер (ранг), що визначає порядок виконання: чим менше номер, тим вище ранг групи завдань. Під час призначення пріоритетів для кожної характеристики завдання створюється своя черга, в кожній з яких всі завдання, що

вимагають виконання, ранжуються в порядку погіршення одного параметра. У кожній черзі заявки ранжуються в порядку зменшення важливості, тобто першими вважаються заявки з найкращим значенням характеристики. Пріоритети завданням присвоюються таким чином:

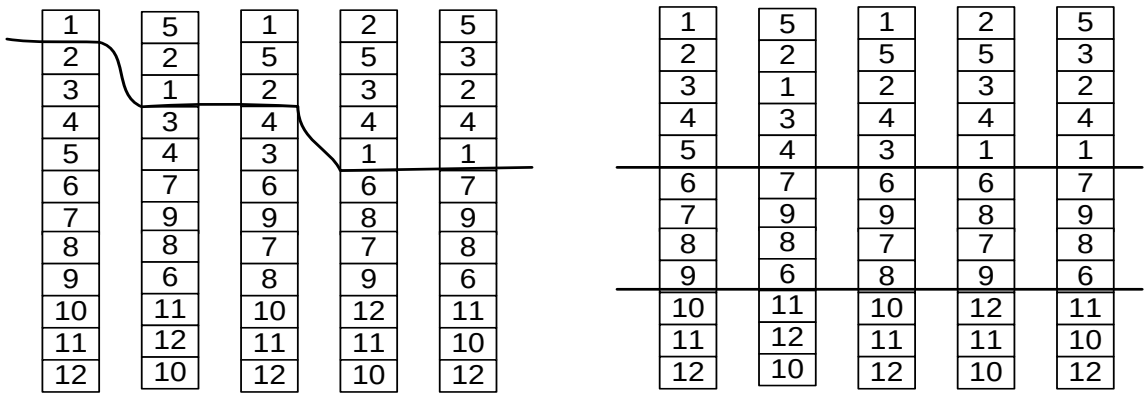
– крок 1. Вибирається перше завдання з першої черги;

– крок 2. Виконується пошук цього ж завдання в наступній черзі. Складається список завдань, які знаходяться вище за нього в черзі. Всі ці завдання краще нього за другою характеристикою, але гірше за першою. Якщо вважати, що всі характеристики однаково важливі, то встановити першість між такими завданнями неможливо, а значить, вони відносяться до однієї групи пріоритетів;

– крок 3. Виконується пошук в наступній черзі кожного з обраних раніше завдань. Запам'ятовуються всі завдання, що знаходяться вище останнього. Далі виконується п. 2 для кожного нового завдання зі списку;

– крок 4. П. 2, 3 повторюються до тих пір, поки знаходяться нові завдання в чергах за такими характеристиками;

– крок 5. Якщо завдання різних рангів представити у вигляді стовпців (рис. 2.5), тоді якщо в черговому стовпці розширився набір завдань конкретного рангу, треба повернутися в усі попередні стовпці і знайти гірше становище кожного завдання з цього набору.



Початковий варіант Остаточний варіант

Рисунок 2.5 – Приклад призначення пріоритетів завданням

На наведеному рисунку значення характеристик нормалізовані відповідно до шкали 1-12 для спрощення.

Таким чином, в запропонованому методі фактично використовується розвинений варіант дисципліни обслуговування SJN (Shortest Job Next – наступним виконується найкоротше завдання), так як для визначення рангів використовуються не тільки часи виконання, а й інші характеристики завдань. Процедура планування за структурним критерієм з урахуванням несумісності задач за використовуваними ресурсами була реалізована за допомогою наступного простого алгоритму, запропонованого в роботі [14].

З метою запобігання одночасного запуску на виконання несумісних між собою завдань, для всіх, хто знаходиться на виконанні і готових до виконання завдань підтримується симетрична булева матриця, кожен елемент якої визначається наступним чином: 1, якщо *i*-е завдання сумісне за використовуваними ресурсами з *j*-м завданням; *tcn* [*i*] *.r* [*j*] = 0 – в іншому випадку. Процедура вибору завдань на виконання полягає в наступному. Перший раз для виконання вибираються завдання з найвищим рангом. Далі аналізуються рядки матриці сумісності завдань, помічені номерами виконуваних завдань. Відзначаються всі стовпці матриці, в яких у виділених рядках знаходяться одиничні значення. Номери цих стовпців визначають завдання, сумісні з виконуваними завданнями. Серед них і вибираються завдання з найбільш високим рангом для виконання. При завершенні виконання будь-якої задачі з матриці сумісності завдань видаляються відповідні рядок і стовпець, і для вільних процесорів вибираються завдання відповідно до відкоригованої матриці. Вибрані задачі запускаються на виконання за допомогою API-функцій базової операційної системи (ОС). Далі вже ними керує вбудований планувальник операційної системи і завдання працюють відповідно до дисципліни диспетчеризації цієї ОС.

У цьому методі при виборі завдань враховуються ресурси, використовувані завданням. На виконання спочатку запускаються завдання, що використовують мінімум ресурсів. При цьому метод дозволяє уникати одночасного запуску на виконання завдань, що використовують одні і ті ж ресурси. Цей метод призначений для використання в зовнішньому щодо ядра операційної системи планувальнику (диспетчері) завдань.

Розглянемо деякі відомі методи планування завдань в розподілених обчислювальних системах і порівняємо їх з запропонованим вище методом.

Одним з відомих сучасних методів планування є метод, який використовує багаторівневі черги зі зворотним зв'язком. Цей метод використовують диспетчери завдань в ОС Windows і Linux. Його використання призводить до того, що першими виконуються процеси, які можуть виконуватися за час менше виділеного кванта часу. Якщо квант часу використовується повністю, завдання переходить в чергу нижчого рівня. Система запам'ятовує, чи використовувало завдання минулого разу повністю свій квант часу. Якщо так, то завдання переходить в чергу нижчого рангу. Наступного разу, коли завдання буде додано до системи, воно знову почне виконуватися з тієї черги, в якій закінчувалося виконання минулого разу. Завданням з верхньої черги надається квант часу в першу чергу, але завданням з нижніх черг в деяких модифікаціях методу квант часу надається більший. У всіх модифікаціях методу першими виконуються завдання, які швидше звільняють процесор. У розробленому методі планування також присвоюються максимальні пріоритети завданням, що мінімально використовують ресурси. Таким чином, підвищується число виконаних в одиницю часу завдань.

У системах черг із зворотними зв'язками відкладається виконання завдань, яким не вистачило одного кванту часу для виконання. Завдання з'являється вперше в черзі з пріоритетом свого базового потоку і

спускається в нижні черги, поступаючись місцем іншим завданням.

Така організація може привести до нескінченного відкладання процесів з нижньої черги. Це трапиться, якщо в систему постійно надходить велика кількість процесів, які потребують обробки, або є кілька процесів, що інтенсивно використовують функції введення - виведення.

Перевагою розробленого методу планування є те, що пріоритети завдань перераховуються досить рідко (тільки коли змінюється склад завдань). При додаванні нових завдань інформація про них записується в матрицю пріоритетів всіх завдань; розраховується, до якої групи пріоритетів вони відносяться. Інформація про сумісність нових завдань по ресурсах з іншими додається також в матрицю сумісності всіх завдань. Після закінчення виконання завдання зникає зі списку завдань, готових до виконання.

В ОС Windows пріоритет потоку підвищується, якщо потік був фоновим, і до нього з'явився інтерактивний запит. Розроблений планувальник завдань не витрачає час на перерахунок пріоритетів, а просто вибирає сумісні по ресурсам завдання з найвищим пріоритетом і запускає їх на виконання.

При цьому пріоритети потоків змінюються щодо пріоритету базового потоку для того, щоб інтерактивні процеси виконувалися першими (швидкість їх виконання контролює користувач і, до того ж, вони швидко звільняють процесор). Завдання ж фонові, які активно і довго займають процесор, працюють з більш низьким пріоритетом. Це дуже гнучка система, що швидко реагує на зміну в поведінці потоків. Втім, слід відзначити, що в системі диспетчеризації завдань це не завжди потрібно. Потоки реального часу вважаються більш високопріоритетними, а потоки з динамічними пріоритетами можуть і почекати. Не міняються динамічно пріоритети потоків (завдань), що дозволяє економити час на розрахунки пріоритетів.

Крім того, в системі диспетчеризації ОС Windows збільшується

пріоритет потоку, який вийшов зі стану очікування і перейшов в стан готовності. Це використовується для зменшення часу реакції інтерактивних додатків на дії користувача. В системі управління не так важливі інтерактивні процеси, а важливішими є процеси, що керують роботою об'єкта управління, тобто вони повинні мати найвищий пріоритет. Тому в розробленій дисципліни планування підвищення пріоритету потоку, що відповідає на запит користувача, не використовується.

В ОС Windows є базовий пріоритет у кожного потоку, який може мати 6 значень: низький – 4, нижче середнього – 6, середній – 8, вище середнього – 10, високий – 13, інтерактивних потоків (за термінологією Windows реального часу) – 24. Його можна змінити за допомогою панелі диспетчера задач. У розробленій системі пріоритет потоку не змінюється в залежності від того, чи став потік інтерактивним або є робочим потоком. Це зроблено тому, що в системі управління не можна зменшувати час відгуку на запит оператора за рахунок завдань, що реагують на зміни в екологічній обстановці.

При реалізації методу планування завдань за структурним критерієм потрібно було забезпечити гарантії обслуговування завданням з тим, щоб завдання реального часу виконувалися до заданого терміну. При цьому треба враховувати, що на верхньому рівні системи управління є завдання з періодом виконання в одну або кілька секунд, хвилин, годин, діб, тижнів, місяців. Щоб визначити значення властивостей завдань для планування розглянутим вище способом, на стадії налагодження системи потрібно декілька раз запускати кожну задачу з метою визначення максимальних значень використовуваних властивостей. Зокрема, перерахунок призначених для користувача пріоритетів завдань проводився при додаванні в систему нових завдань або при зменшенні кількості завдань верхнього рівня, так як пріоритети у завдань проставляються відносно один одного. Відповідно, спочатку завдання запускаються з пріоритетами, призначеними емпірично, далі пріоритети перераховуються. Коли

завдання запускається вперше, емпіричний пріоритет – це єдиний параметр, що використовується для визначення робочого пріоритету. Після декількох запусків визначаються максимальні значення всіх характеристик, і перерахунок пріоритетів виконується вже з урахуванням декількох параметрів. Також в процесі обчислення пріоритетів завдань визначається, які завдання використовували одні й ті ж ресурси, що відзначається в матриці їх сумісності. Розглянутий метод планування задач збільшує середні значення завантаженості РС, а, відповідно, і її пропускну здатність.

2.4 Формування мережі пунктів екологічного контролю в ІСЕМ

У загальному випадку формування мережі пунктів екологічного контролю в ІСЕМ, де мають бути розташовані засоби бору (датчики) та засоби обробки (контролери) первинних екологічних даних полягає у вирішенні таких завдань:

– виявлення джерел екологічної небезпеки (зокрема, викидів шкідливих речовин) в зоні екологічного моніторингу;

– визначення кількості та місць розташування пунктів екологічного контролю (і, відповідно, контролерів обчислювальної мережі ІСЕМ), для моніторингу екологічних параметрів поблизу виявлених джерел екологічної небезпеки (ЕН);

– формування топології обчислювальної мережі, що поєднує робочу станцію та контролери ІСЕМ.

Розглянемо підхід до ідентифікації джерел екологічної небезпеки (на прикладі викидів шкідливих речовин), що передбачає виконання таких основних етапів:

– виділення з множини потенційних джерел ЕН підмножини можливих джерел ЕН з урахуванням апріорної інформації (зокрема, даних метеослужб);

– виявлення джерела викидів з використанням методів кластеризації (угрупування) екологічних даних (ЕД).

Нехай промислова зона, де має здійснюватися екологічний моніторинг (ЕМ), зазвичай має такі характерні особливості:

– наявність підприємств з виробничими технологіями, повязаними з використанням різних типів шкідливих та забруднюючих речовин;

– наявність різних груп забруднень, що утруднює використання єдиної моделі екологічного стану;

– взаємозв'язок шкідливих речовин (ШР) та їх проявів, що значно ускладнює знаходження вирішальних правил.

Зазвичай в зоні екологічного моніторингу промислового району присутні декілька джерел забруднення атмосфери з відомою потужністю. Ці джерела створюють поле концентрації забруднюючих речовин, параметри якого залежать від метеоумов. Наявність таких джерел створює на території моніторингу область підвищених концентрацій, які реєструються у певних точках. За цими даними необхідно відновити поле концентрації забруднення на території даного регіону, а потім ідентифікувати джерело забруднення.

Поширеною є ситуація, коли в певних точках зони ЕМ відсутні необхідні датчики первинних даних, які мали б забезпечувати отримання необхідної інформації. В цьому разі доцільно використовувати інтелектуальні інформаційні технології (ІІТ), які дозволяють на основі обробки нечіткої вхідної інформації отримати необхідні дані для вирішення завдань ЕМ. Розробка алгоритмів кластеризації екологічних даних для цілей виявлення ШР передбачає необхідність необхідність поділу загальної множини подій на кластери з подальшим виділенням в кожному кластері головного (центрального) об'єкта. Таке завдання може бути вирішене за допомогою побудови графа взаємозв'язків об'єктів кластеризації (із зазначенням відстаней між ними) з подальшим пошуком

на цьому графі множин вершин, які відповідають властивостям кластерів. Для рішення цього завдання введемо деякі позначення та припущення.

Нехай заданий *n*-вершинний дводольний граф G(*H,P,E*) (рис. 2.6), в якому вершини першої долі $H=\{h_1,....h_i,...h_n\}$ є множиною забруднюючих речовин; $i=1,...,n$ – індекс, який нумерує всі забруднюючі речовини; вершини другої долі $P=\{p_1,...p_j,...,p_m\}$ створюють множину джерел викидів (передумов), $j=1...,m$ – індекс, який нумерує всі можливі джерела викидів; $E=\{e\}$ – множина ребер.

Графу *G* відповідає бінарне нечітке відношення P_G, що складається з усіх пар виду $<h_i,p_j>$, для кожної з яких визначено деяке дійсне число з інтервалу [0,1], яке дорівнює значенню функції належності $\mu_G(e_k)$ для дуги $e_k \in E$, що відповідає цій парі вершин.

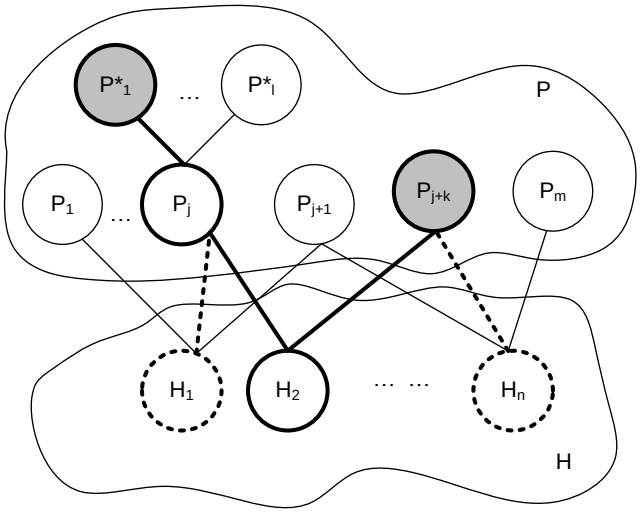


Рисунок 2.6 – Графова модель кластеризації джерел ЕН

Припустимо, що експертами визначена вартість помилки C_{ou}, яка залежить від обраного напрямку пошуку (за найбільш небезпечним видом ЕН та за найбільш ймовірним джерелом ЕН тощо). Позначимо через k_y коефіцієнт впевненості, що визначається відношенням кількості контрольованих джерел ЕН на виділеній множині до загальної кількості можливих джерел ЕН, викликаних появою *i*-ї небезпечної речовини.

Розглянемо задачу покриття двудольного графу G(H,P,E) підграфом G'(H',P',E'), що містить лише множину вершин, виділених в результаті виявлених викидів. Допустиме рішення має бути впорядкованою множиною, що характеризує всі можливі джерела ЕН при заданому пороговому значенні стійкості зв'язку (ступеня належності $\mu_G(<h_i,p_i>)$), кожна компонента якого об'єднана в групу $g^i=(\{h_i\},P^i,E^i)$, $h_i \in H'$, $P^i \, \delta \, P'$, $E^i \, \delta \, E'$ з центром в деякій вершині h_i з першої долі і множиною P^i вершин з другої долі P.

На множині допустимих рішень графа G визначена цільова функція F, що є узагальненим критерієм оптимальності [18]:

$$F_{\Sigma}(\omega, R(x)) = \sum_{i=1}^{2} \omega_i R_i(x) \to min, \qquad (2.1)$$

де $R(x) = \{ R_1(x), R_2(x) \}$ – вектор локальних критеріїв ризику; $\omega = \{ \omega_1, \omega_2 \}$ – вагові коефіцієнти відносної важливості локальних критеріїв:

$$0 < \omega_i < 1 \quad \text{та} \quad \sum_{i=1}^{2} \omega_i = 1. \qquad (2.2)$$

Вагові коефіцієнти для окремих критеріїв призначаються на основі експертних оцінок. Завдання другого рівня полягає в тому, щоб знайти таке рішення, при якому виявляються малими як ризик R_1, так і ризик R_2.

Згідно з розглянутими критеріїями це означає вибір такої послідовності передумов, щоб на множині допустимих рішень графа G' можна було отримати максимальну величину коефіцієнта впевненості пошуку (k_y) при мінімальній ціні помилки (C_{ou}):

$$k_{\partial} \to max \qquad (2.3)$$
$$C\partial = \sum C(p_j) \to min.$$

Метод кластеризації інформації в цьому випадку передбачає реалізацію трьох етапів отримання рішення. На першому етапі для кожного зареєстрованого відхилення змінної проводиться процедура попередньої кластеризації. Для цього використовуються основні властивості нечітких відносин, зокрема узагальнення поняття α-рівня, під яким розуміється відношення $P\alpha=\{(<h_i,p_j>|\mu_X<h_i,p_j>)\geq\alpha\}$, $(\forall <h_i,p_j>\in P_G)$, де $\alpha \in [0,1]$.

Необхідно підкреслити деякі особливості графової моделі, а саме – дворівневу структуру однієї з долей графа, яка обумовлена такими об'єктивними факторами: множина забруднюючих речовин P може містити як прості (мають лише одну вершину) так і складові P^*, $P^* \subseteq P$ передумови, існування яких обумовлено більш широким складом можливих викидів, при цьому сукупність складових передумов описується як бінарні нечіткі відносини $H \times P^*$: $P_1=\{<h_i, p_l^*>, \mu(<h_i, p_l^*>)\}$; $P^* \times P$: $P_2=\{<p_l^*, p_j>, \mu(<p_l^*, p_j>)\}$.

Таким чином, кожному кортежу нечіткого відношення $<h_i,p_j>\in P_G$ буде відповідати дуга графа $e_k=<h_i,p_j>$, що починається у вершині h_i, закінчується у вершині p_j та має значення функції належності $\mu_P(<h_i,p_i>)$ (рис. 2.7).

Потім для кожного зареєстрованого відхилення змінної формуються нечіткі матриці, які декомпозуются на відношення еквівалентності, що представляють собою відповідні α-рівні. Для всіх можливих α-рівнів формується система класів, які включають можливі джерела викидів. Дана процедура передбачає увазі розподіл вихідної множини рішень на класи за принципом належності α-рівня. Результати виконання алгоритму попередньої кластеризації можуть бути використані як вихідні дані для

вирішення завдання оптимізації або ж як попередні результати ідентифікації.

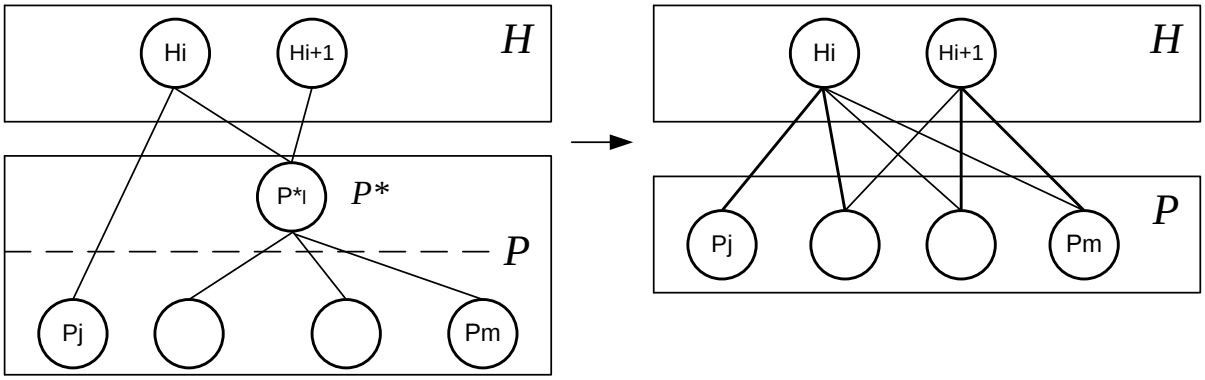


Рисунок 2.7 – Схема задання нечітких відношень

На другому етапі здійснюється упорядкування вершин графа, причому цей етап виконується лише при наявності двох і більше зареєстрованих відхилень змінних (забруднюючих речовин).

Послідовність залежності реєстрованих забруднюючих речовин від можливих джерел, що їх викликають (передумов і висновків), може бути отримана відповідною перенумерацією вершин двудольного графа таким чином, щоб зв'язкові вершини розташовувалися найближче одна до одної (це еквівалентно мінімізації сумарної довжини ребер у відповідному представленні графа).

Процедура упорядкування вершин нечіткого двудольного графа передбачає виконання перенумерації рядків і стовпців його матриці за такої умови: чим більше значення елемента, тим ближче він повинен бути до головної діагоналі.

Формально ця мета є завданням мінімізації функції $(m + 1)$ $(n + 1)$ змінних:

$$J(h_0,...,h_n,p_0,...,p_m)=\sum_{i=0}^{n}\sum_{j=0}^{m}\mu_{ij}(p_j-h_i)^2\rightarrow\min_{p_j,h_i}, \quad (2.4)$$

де μ_{ij} – елемент матриці нечіткого графа, що визначає ступінь зв'язності i-ї вершини з j-ю;

p_j і h_i – номера (координати) рядків і стовпців, що упорядковуються.

Знявши обмеження на цілочисельність координат p_j і h_i, дозволимо рядкам і стовпцях розташовуватися довільним чином на площині. У цьому випадку критерію (2.2) відповідає тривіальне рішення: $J_{min} = 0$ при $p_j = h_i$ $\forall i,j$.

Умовна оптимізація (2.4) може бути здійснена ітераційно (наприклад, з використанням методу проекції градієнта).

Упорядкована матриця дозволяє кожному зареєстрованому порушенню зіставити список можливих типів ЕН в послідовності віддалення від головної діагоналі. З її допомогою можна також визначати групові вклади найбільш близьких пар «висновок-передумова», формуючи уздовж діагоналі блоки заданих розмірів.

На третьому етапі вирішується завдання багатокритеріальної оптимізації з остаточним ранжуванням локальних оцінок в послідовності зменшення їх важливості відповідно до обраних критеріїв пошуку. Перед вирішенням цього завдання проводиться нормалізація локальних критеріїв. Для цього приймається додаткова шкала вимірювання $[\beta,\varphi]$=[1,2]. Вектори нормованих величин коефіцієнта впевненості пошуку і нормованих величин вартості помилки формуються з локальних значень і визначаються за такими формулами:

$$K_N^{(Pi)}=\frac{k_y^{(Pi)}-\min k_y}{\max k_y-\min k_y}(\varphi-\beta)+\beta, \quad (2.5)$$

$$C_N^{(Pi)} = \frac{C^{(Pi)} - \min C_{ou}}{\max C_{ou} - \min C_{ou}} (\beta - \varphi) + \varphi, \quad (2.6)$$

де $k_y^{(Pi)}$ – значення коефіцієнта впевненості пошуку;

$\max k_y, \min k_y$ – відповідно максимальне і мінімальне значення коефіцієнта впевненості пошуку;

$C^{(Pi)}$ – значення ціни помилки;

$\max C_{ou}, \min C_{ou}$ – відповідно максимальне і мінімальне значення ціни помилки.

Після нормування локальних показників визначення найбільш критичних вершин зводиться до вирішення однокритерійного завдання оптимізації виду:

$$F = \max(\omega_1 K_N + \omega_2 C_N). \quad (2.7)$$

Це завдання може бути вирішене за допомогою методу згортання векторного критерію.

Для кожного зареєстрованого відхилення (рис. 2.8), відповідно до послідовності їх виявлення, виконуються наступні операції.

Розглянутий нечіткий граф *G*'(*H*',*P*',*E*') представляється у вигляді списку наступним чином: *G*'(H27)={(<29,27>|0,2), (<28,27>|0,6), (<7,27>|0,3), (<7/1,7>|0,3), (<7/2,7>|0,3), (<5,27>|0,6), (<30,5>|0,5)}.

Формується матриця досяжності *G*, виходячи з правил побудови шляху в нечіткому кінцевому графі.

На множині причин порушень для даної змінної (H27) будується нечітка матриця перевірок *M*. Для визначення множини, що визначає вектор-рядок матриці перевірок, визначається перетин нечіткої множини вихідної матриці по стовпцю і по рядку.

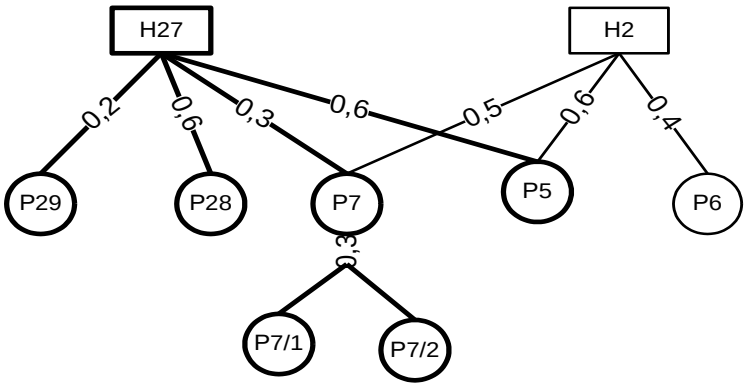


Рисунок 2.8 – Граф, що містить вершини з зареєстрованими відхиленнями

З огляду на основні властивості бінарних нечітких відносин, нечітка матриця перевірок *H* приводиться до множини, що складається з 0 і 1 та є α-рівнем нечіткої множини матриці *H*, а далі записується у вигляді чітких матриць *M*.

Для всіх можливих α-рівнів будується система класів, що описують можливі джерела викидів. Система вкладених класів для розглянутого прикладу представлена в таблиці 2.4.

Таблиця 2.4 – Система вкладених класів для розглянутого прикладу

Порушення	Значення α-рівня	Кількість класів	Опис класів
H_{27}	0,2	5	$\{H_{27,}P_{7/1,}P_7\}$, $\{H_{27,}P_{7/2,}P_7\}$, $\{H_{27,}P_{28}\}$, $\{H_{27,}P_{29}\}$, $\{H_{27,}P_5\}$
	0,3	3	$\{H_{27,}P_{28}\}$, $\{H_{27,}P_{29}\}$, $\{H_{27,}P_5\}$
	0,5	2	$\{H_{27,}P_{28}\}$, $\{H_{27,}P_5\}$
	0,6	1	$\{H_{27,}P_{28}\}$, $\{H_{27,}P_5\}$
H_2	0,4	3	$\{H_{2,}P_5\}$, $\{H_{2,}P_6\}$, $\{H_{2,}P_7\}$
	0,5	2	$\{H_{2,}P_5\}$, $\{H_{2,}P_7\}$
	0,6	1	$\{H_{2,}P_5\}$

Результати виконання алгоритму кластеризації можуть бути використані як вихідні дані для вирішення завдання оптимізації або як

попередні результатів діагностування забруднення. Отримані класи можуть інтерпретуватися користувачем (оператором) як зони можливих порушень, які потребують особливої уваги.

Для наведеного прикладу при α = 0,5 з розгляду випадають відразу дві мультідуги системи, що скорочує кількість вершин-класів з 5 до 2.

Далі матриці окремих висновків і передумов об'єднуються в підсумкову матрицю. Ітеративний алгоритм упорядкування по зв'язності вершин двудольного графа G=(*H*,*P*,*E*) полягає в почергової фіксації однієї з груп уздовж координатної осі і переміщення кожної вершини іншої групи в геометричний центр координат пов'язаних з нею вершин протилежної групи. Упорядкування вершин двудольного графа еквівалентно приведенню його матриці суміжності до матриці, де ненульові елементи зосереджуються біля головної діагоналі.

Для оперативного групування інформації уточняються значення мінімального рівня стійкості зв'язку (α - рівень) (за замовчуванням в системі приймаються значення α = 0,5). Згідно з пороговим значенням α - рівня з множини результатів алгоритму попередньої кластеризації вибирається відповідна чітка матриця перевірок M_α, де видаляються нульові стовпці і рядки. Решта стовпців матриці інтерпретуються як визначальні елементи можливих джерел виникнення порушень. Для всіх елементів залишившихся стовпців матриці M_α обчислюється нормоване значення ціни помилки. Ціна помилки ідентифікації вибирається, виходячи з обраного напрямку пошуку: по найбільш небезпечним викидам або по найбільш частим порушенням (в розглянутому прикладі вибрано напрямок пошуку по найбільш небезпечним викидам).

Для всіх елементів стовпців матриці *Hα*, що залишилися в процесі пошуку, обчислюються поточне і нормоване значення коефіцієнта впевненості пошуку на множині $D \cup V$. Поточне значення цього коефіцієнта обчислюється як відношення загальної кількості можливих порушень на множині *D* до кількості контрольованих порушень на виділеній множині *V*.

Результати обчислень наведені в таблиці 2.5 , де ЦФ – позначення цільової функції. Аналіз отриманих результатів кластеризації дозволяє зробити такий висновок: представлення нечіткої матриці перевірок *M* у вигляді відношення подібності дає можливість організувати процедуру пошуку відхилень в нечіткому графі G * (*V, E*) по частинах, уникаючи повного комбінаторного перебору.

Таблиця 2.5 – Результати кластеризації

Пара-метр	Ціна похибки ідентифікації		Коефіцієнт впевненості пошуку		ЦФ
	Експерт. величина	Нормована величина	Фактичне значення	Нормоване. значеннння	
$\boldsymbol{P_i}$	$\boldsymbol{C^{(Pi)}}$	$C_N^{(Pi)}$	$k_д^{(Pi)}$	$K_N^{(Pi)}$	**F**
P_5	6	1,6	0,032	2	1,76
P_7	7	2	0,016	1	1,6
P_{28}	9	1	0,016	1	1

Однак при цьому можливі випадки, коли реальне джерело викидів буде проігноровано, на що вказує розрахункове значення коефіцієнта впевненості пошуку. Крім того, необхідно відзначити, що пропонований підхід використовує обмежену кількість елементів, що входять в множини передумов (можливих джерел) і висновків (проявів порушень). Якість аналізу сукупності ознак прямо пов'язана з повнотою моделі, яка має бути правдоподібною, але в той же час простою та компактною.

Розглянемо завдання побудови мережі станцій спостереження в ІСЕМ, що полягають у визначенні пунктів та місць розташування в зоні екологічного моніторингу.

До найважливіших сфер застосування ІСЕМ слід зарахувати контроль екологічного стану промислових регіонів. Екологічна безпека

таких регіонів пов'язана з необхідністю зниження концентрації шкідливих викидів у довкілля. Існуючий класифікатор надзвичайних ситуацій (НС) дозволяє формалізувати метод визначення можливої шкоди та може використовуватися для машинної обробки інформації екологічного контролю в ІСЕМ. При цьому виникає необхідність створення мережі станцій моніторингу даних щодо якості атмосферного повітря. Як вихідні дані для створення мережі моніторингу необхідні: топографічна карта території; просторовий розподіл промислових підприємств; дані про викиди із джерел забруднення; метеодані, з якими може бути пов'язане розсіювання викидів. Для моніторингу викидів необхідно визначити простір, для якого можливе перевищення гранично допустимих концентрацій (ГДК) домішок, а також поле ймовірності такого перевищення. Моделювання розсіювання від стаціонарних джерел дозволяє визначити масив розподілу максимальних концентрацій. Сортування елементів такого масиву за зменшенням ймовірності дозволяє оптимально вибрати місця розташування постів контролю. При виборі вузлів координатної сітки їх розміщення необхідно враховувати сумарну ймовірність виявлення викидів. Величина найбільшої концентрації кожної домішки $C_м$ (мг/м) у приземному шарі атмосфери немає перевищувати величини її ГДК в атмосферному повітрі: $C_м \pounds ГДК$. При визначенні простору, на якому можливе перевищення ГДК, на топографічну карту регіону наноситься сітка з кроком 0,1 км. Результати розрахунків концентрації небезпечних хімічних речовин (НХР) відображаються на карті розсіювання НХР для групи джерел.

Для кожного джерела забруднень розраховується зона впливу з радіусом, що дорівнює найбільшому із значень $X1$ і $X2$ ($X1$ відповідає зоні зі значенням $C_м$, рівним 5%, а $X2$ – відстань від джерела, починаючи з якого концентрація домішок становить 5% ГДК). Зона впливу уточнюється з урахуванням троянди вітрів. Існують два основні методи побудови карт зон можливого перевищення ГДК: у першому методі використовується

система трикутників, де вершини трикутників знаходяться в точках вимірювання, в іншому – регулярна сітка, де точки розміщуються у її вузлах. В усіх вузлах визначається параметр, що характеризує швидкість зміни концентрації ОХВ. Виділення регіоналізованих змінних та облік троянди вітрів дозволяють застосувати метод крайгінгу, пов'язаний із поняттям напівдисперсії. Потрібні параметри крайгінгу визначаються з умови мінімуму напівдисперсії відхилення побудованої функції від відомих (напівдисперсія є міра ступеня залежності між пробами заданої бази). Напівдисперсія може бути обчислена для відстаней, кратних заданій величині:

$$\gamma_h = \frac{1}{2n} \sum_{i=1}^{n-h} (X_i - X_{i+h})^2 , \qquad (2.8)$$

де X_i – значення змінної X у точці i;

h – інтервальний обрій.

У міру збільшення відстані h порівнювані точки слабше пов'язані одна з одною, що призводить до великих значень напівдисперсії. На певній відстані напівдисперсія не зростає, а квадрати різниць відповідають дисперсії відносного середнього значення. Відстань, де напівдисперсія наближається до дисперсії, визначає окіл зв'язкових точок. У вузлах сітки залишаються точки та їх околи, що є потенційними місцями для розташування станцій мережі моніторингу. Для розрахунку напівдисперсії при великій кількості точок і визначення околів, що враховуються, був розроблений програмний модуль «НХР-ГДК». Як вхідні дані цього модуля необхідно задати: розмір карти (ширину і довжину), крок по осях; координати точок відбору проб повітря та значення приземних концентрацій у цих точках; координати підприємств; координати потенційних місць розміщення постів спостереження. Вихідними даними модуля НХР-ГДК є околи точок з координатами стаціонарних постів. З

урахуванням швидкості та напрямку вітру розраховується ймовірність виявлення викиду для відомих метеоумов та оцінюється ймовірність виявлення викиду загалом для даної ICEM.

В таблиці 2.3 наведено результати оцінки ймовірності виявлення викиду системою спостереження для 4 джерел, що складається з 3, 4 або 5 постів спостереження, отримані із застосуванням запропонованого підходу.

Імовірність виявлення викиду системою спостереження обчислюється як добуток ймовірності певної швидкості та напрямки вітру та ймовірності наявності викиду джерелом забруднення.

Таблиця 2.6 – Оцінка ймовірності виявлення викидів системою моніторингу

Джерело	Кількість постів спостереження		
	3	4	5
A	23,362%	82,432%	98,758%
B	25,745%	65,179%	85,975%
C	67,891%	89,357%	99,134%
D	76,211%	100%	100%
E	58,620%	98,785%	100%

Сума ймовірностей всіх значень швидкості і напряму вітру є ймовірністю виявлення викиду системою спостереження загалом. Враховуючи високу вартість організації постів спостереження, необхідно вибрати таку їх кількість, яка забезпечить мінімально заданий рівень ймовірності виявлення викидів.

Сортування підсумкового масиву за спаданням ймовірності дозволяє оптимально вибрати місця розташування стаціонарних постів ICEM, починаючи з місць, де ймовірність виявлення викидів максимальна.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ЗАВДАНЬ ДИСПЕТЧЕРИЗАЦІЇ РЕСУРСІВ СИСТЕМИ ЕКОЛОГІЧНОГО МОНІТОРИНГУ

3.1 Диспетчер завдань системи екологічного моніторингу

Диспетчер завдань реального часу робочої станції, в основу роботи якого покладено метод планування за структурним критерієм, був розроблений на мові С ++. Він працює як звичайний процес ОС Windows з пріоритетом 16 і запускається на виконання з періодом повторення, що дорівнює одній секунді. Величина періоду повторення роботи диспетчера може змінюватися користувачем, виходячи з особливостей конкретної обчислювальної системи.

Вікно управління диспетчера задач викликається на екран з головного меню пакета.

Диспетчер забезпечує функціонування в багатозадачному режимі до 32 завдань згідно з важливістю завдання, часом початкового запуску і / або періодом повторення завдання, використовуваними ресурсами та іншими характеристиками, що встановлюються індивідуально для кожного завдання після натиснення на значок відповідного завдання.

Програма функціонування завдання є EXE-модулем, написаним на одній з мов програмування для ОС Windows. Вона може використовувати практично всі можливості ОС Windows і середовища програмування, використаного для розробки програми. В програмі можуть бути задіяні практично всі дані системи екологічного моніторингу, включаючи первинні дані, що надходять з контролерів. У завданнях, що функціонують під управлінням диспетчера, може використовуватися подієве управління виконанням програм, в тому числі відпрацювання тимчасових подій з точністю до міллісекунд.

У кожному періоді повторення диспетчер перевіряє, чи змінився набір завдань в порівнянні з попереднім періодом. Якщо набір завдань не

змінився (тобто мають виконуватися тільки завдання з періодом повторення в 1 секунду, як це було і в попередньому періоді), ранги завдань є відомі, і вони запускаються планувальником на виконання з пріоритетами реального часу 17, 18,… відповідно до їх рангів, визначених в попередній період.

Якщо ж в поточному періоді набір завдань змінився (додалися одне або кілька завдань з періодами повторення більшими, ніж 1 сек, або ж, навпаки, не потрібно виконувати частину завдань, що виконувалися в попередньому періоді), включається в роботу алгоритм визначення рангів завдань відповідно до їх структурних характеристик. Після перерахунку рангів завдань згідно з розглянутим вище методом вони, як і в попередньому випадку, запускаються на виконання з пріоритетами реального часу 17, 18, … відповідно до їх нових рангів.

Реалізація процедури диспетчеризації полягає в наступному: будуються матриця пріоритетів і матриця сумісності для всіх завдань системи, незалежно від заданих часу / дати їх запуску. Це робиться після установки системи, а також кожного разу, коли оператор змінює склад завдань. Створюються поточні матриці пріоритетів і сумісності, куди копіюється інформація про завдання, готові до виконання. Після цього завдання ранжуються за пріоритетами та вибирається група завдань з найвищим пріоритетом, серед них з матриці сумісності вибираються завдання для виконання, які не використовують одні і ті ж ресурси одночасно. Вибрані задачі запускаються на виконання в режимі поділу часу. Виконані завдання викреслюються з поточної матриці пріоритетів і поточної матриці сумісності. Якщо з'являються нові завдання, час виконання яких настав, інформація про них заноситься в поточні матрицю пріоритетів і матрицю сумісності. Коли оператор додає нові завдання в систему, після декількох їх виконань стають відомі їхні характеристики і розраховується їх пріоритет щодо інших завдань.

Спочатку в матрицю пріоритетів заноситься пріоритет, вказаний для

завдання оператором при додаванні його в систему. Після того, як буде виконано кілька разів, його пріоритет розраховується заново з урахуванням інших характеристик (таких, як час виконання, число звернень до файлів бази даних тощо). Для кожного показника відомо, яке його значення вважається найкращим.

Таким чином, при додаванні нових завдань в систему планувальник запам'ятовує значущі характеристики нових завдань, вибирає максимальні значення характеристик після декількох виконань завдання та додає дані про них в матрицю пріоритетів і матрицю сумісності завдань. Диспетчер завдань реального часу робочої станції пакета, який використовує розроблений метод, працює паралельно з диспетчером завдань ОС Windows. Роль такого додаткового диспетчера задач полягає у визначенні пріоритетів завдань (17, 18, ...) відповідно до їх рангів, запуску завдань на виконання; далі управління завданням здійснює вбудований планувальник ОС. Зокрема, він виділяє пам'ять для потоків, перериває виконання потоків після закінчення виділеного кванта часу, перемикає контексти потоків та зберігає в стеці призупинених потоків стан регістрів процесора.

Додатковий диспетчер запускає тільки завдання з програмного комплексу задач верхнього рівня системи екологічного і техногенного моніторингу. Управління сервісами ОС здійснює вбудований диспетчер ОС. Додатковий диспетчер далі перевіряє, чи виконалися повністю за відведений квант часу завдання з найвищим пріоритетом; якщо виконалися всі завдання, він запускає на виконання всі завдання наступного пріоритету (нижчого). Як приклад, який демонструє переваги методу планування завдань по структурному критерію, розглянемо один з періодів роботи диспетчера задач робочої станції при обслуговуванні завдань, перерахованих в таблиці 3.1.

Для простоти викладу в цій таблиці наведено в повному обсязі лише основні завдання робочої станції.

Таблиця 3.1 – Завдання робочої станції центру екологічного моніторингу

№ завдання	Позначення	Призначення завдання	Період виконання завдання
1	ОЕІ	Завдання обробки екологічної інформації в реальному часі	1 сек
2	ФАВ	Завдання фіксації аварійних викидів	1 сек
3	ІЕД	Завдання ведення історії екологічних даних	2 сек
4	ВЕД	Завдання відображення екологічних даних	1 сек
5	СОЕІ	Завдання статистичної обробки екологічної інформації	30 сек
6	КПО	Завдання періодичного контроля працездатності обладнання	1 хв.
7	КВК	Завдання періодичного контроля вимірювальних каналів	30 хв.
8	РПО	Завдання розрахунку продуктивності обладнання з метою оптимізації його роботи	15 хв.
9	СМЕО	Завдання спрощеного моделювання екологічної обстановки	1 година
10	ДМЕО	Завдання детального моделювання екологічної обстановки	8 годин

Завдання в таблиці перераховані за зменшенням їх важливості, тобто номер завдання збігається з призначеним для користувача пріоритетом; при цьому номер 1 відповідає найбільш пріоритетному завданню.

Кожне завдання характеризується наступними властивостями:

– призначеним для користувача пріоритетом завдання (ППЗ);

– припустимим часом рішення завдання в мілісекундах (ПЧР);

– кількістю звернень до записів таблиць бази історії екологічних даних (КЗІ);

– кількістю екологічних параметрів, для читання або запису значень яких завдання звертається до сервера OPC (КЕП).

Значення перерахованих властивостей завдань наведені в таблиці 3.2.

Таблиця 3.2 – Властивості завдань PC ICEM

	ОЕІ	ФАВ	ІЕД	ВЕД	СОЕІ	КПО	КВК	РПО	СМЕО	ДМЕО
ППЗ	1	2	3	4	5	6	7	8	9	10
ПЧР	150	500	700	750	900	1000	1400	1200	1600	1800
КЗІ	0	53	140	0	106	152	34	40	124	150
КЕП	130	48	62	134	63	64	142	10	12	23

Побудуємо булеву матрицю з елементами *tcn* [*i*] *.p* [*j*] (табл. 3.3). Для цього значення властивостей завдання *tcn* [*i*] будемо порівнювати зі значеннями відповідних властивостей завдання *tcn* [*j*]. Якщо виявиться, що завдання *tcn* [*i*] за значенням всіх властивостей перевершує завдання *tcn* [*j*], тоді в *j*-му стовпці матриці ставимо 1, інакше – 0.

Таблиця 3.3 – Булева матриця для визначення рангів завдань

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	1	0	0	1	0	0	0
2	0	0	1	0	1	1	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0
4	0	0	0	0	0	0	1	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0	0	0	0

В даному випадку завдання вважається кращим, якщо значення цієї властивості для завдання виявляється меншим або рівним, ніж це ж значення для іншої задачі. Слід зазначити, що в планувальнику завдань є можливість для кожної властивості, що характеризує завдання, задавати, що вважати краще: оператор порівняння більше (>), оператор більше або дорівнює (> =), оператор менше (<) або, нарешті, оператор менше або дорівнює (<=).

У цій матриці нульові стовпці відповідають завданням 1, 2 і 8 (завдання 1-го рангу). Далі викреслюємо перераховані стовпці і відповідні їм рядки з матриці та отримуємо матрицю, наведену в таблиці 3.4.

Таблиця 3.4 – Булева матриця 2-го рівня для визначення рангів завдань

	3	4	5	6	7	9	10
3	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0
5	0	0	0	1	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1
10	0	0	0	0	0	0	0

Відповідно до нульових стовпців матриці 2-го рівня (табл. 3.4) приписуємо ранг, рівний 2, завданням 3, 4, 5 і 9, після чого будуємо матрицю 3-го рівня (табл. 3.5).

В отриманій матриці всі стовпці нульові, тому приписуємо залишившимся завданням 6, 7 і 10 ранг, що дорівнює 3.

Таблиця 3.5 – Булева матриця 3-го рівня для визначення рангів завдань

	6	7	10
6	0	0	0
7	0	0	0
10	0	0	0

Аналіз отриманих результатів показує, що значення першої властивості (призначений для користувача пріоритет ППЗ) для завдань є набором монотонно зростаючих величин, а значення другої властивості також зростають практично в тому ж порядку (порушена монотонність тільки для задач 7 і 8). При цьому диспетчер відніс до груп різних рангів кілька завдань (два або три) в порядку зростання значень їх перших двох властивостей і додав в кожну з груп по одному завданню з кінця списку (табл. 3.6). У таблиці, крім розподілу завдань по рангах, для кожної групи завдань одного рангу задано сумарне значення кожної з властивостей ППЗ, ПВР, КЗІ та КТП. З таблиці 3.5 випливає, що при підсумовуванні абсолютних значень властивостей завдань ці суми для кожної з властивостей монотонно зростають при переході від групи завдань одного рангу до наступного, а між різними властивостями ці суми не можна порівнювати, так як вони представляють різні характеристики завдань в непорівнянних одиницях виміру.

Таблиця 3.6 – Розподіл завдань за рангами

	Ранг1	Ранг 2	Ранг 3
	1, 2, 8	3, 4, 5, 9	6, 7, 10
ППЗ	11	21	23
ПЧР	1850	3950	4200
КЗІ	93	370	336
КЕП	188	271	229

Оскільки в процедурі розбиття задач на ранги враховувалися тільки відношення значень властивостей завдань між собою (краще, якщо менше або дорівнює), а не їх абсолютні величини, то значення властивостей можна замінити їх номерами, починаючи з 1, так, щоб дотримувалися встановлені раніше відносини між завданнями (табл. 3.7).

Таблиця 3.7 – Нормалізовані властивості завдань

	ОЕІ	ФАВ	ІЕД	ВЕД	СОЕІ	КРО	КВК	РПО	СМЕО	ДМЕОО
ППЗ	1	2	3	4	5	6	7	8	9	10
ПЧР	1	2	3	4	5	6	8	7	9	10
КЗІ	1	5	8	2	6	10	3	4	7	9
КЕП	8	4	5	9	6	7	10	1	2	3

Якщо по цій таблиці побудувати булеву матрицю для визначення рангів завдань, вона виявиться такою ж, як і таблиця 3.2, а, значить, і розбиття задач на ранги буде тим же. У відповідності з отриманими значеннями таблиці 3.7, на рисунку 3.1 кольоровими областями показана діаграма зміни значень властивостей від завдання до завдання; при цьому номери завдань перераховані по осі Х.

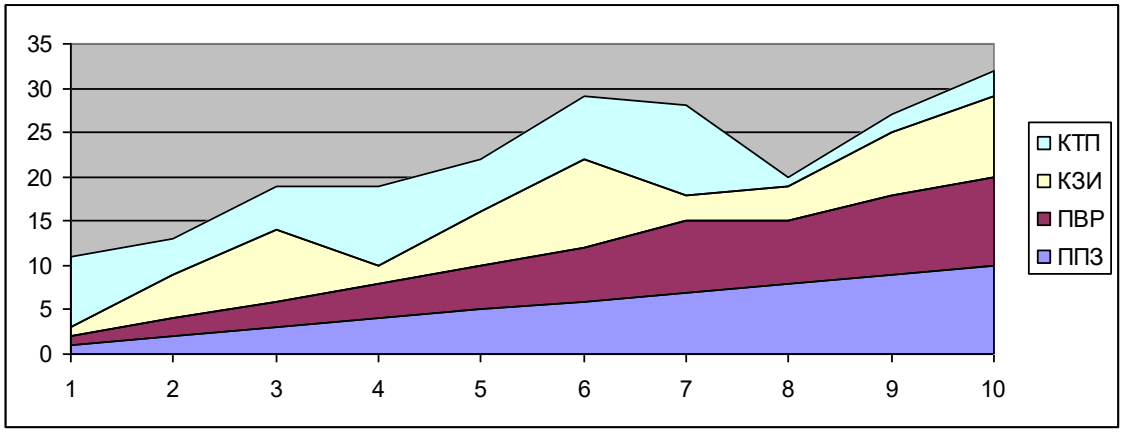


Рисунок 3.1 – Діаграма зміни властивостей завдань

Згідно з діаграмою на рисунку 3.1, якщо розглядати кожну властивість завдання як використовуваний нею той чи інший ресурс, і якщо виконувати ці завдання за зменшенням їх користувальницьких пріоритетів, використання ресурсів може виявитися досить нерівномірним.

Далі побудуємо таблицю (табл. 3.8) і діаграму (рис. 3.2) зміни тих же властивостей, але вже не для окремих завдань, а для груп завдань.

Таблиця 3.8 – Сумарні значення узагальнених властивостей груп завдань з однаковими рангами

	Ранг 1	Ранг 2	Ранг 3
	1, 2, 8	3, 4, 5, 9	6, 7, 10
ППЗ	11	21	23
ПВР	10	21	24
КЗІ	10	23	22
КТП	13	22	20

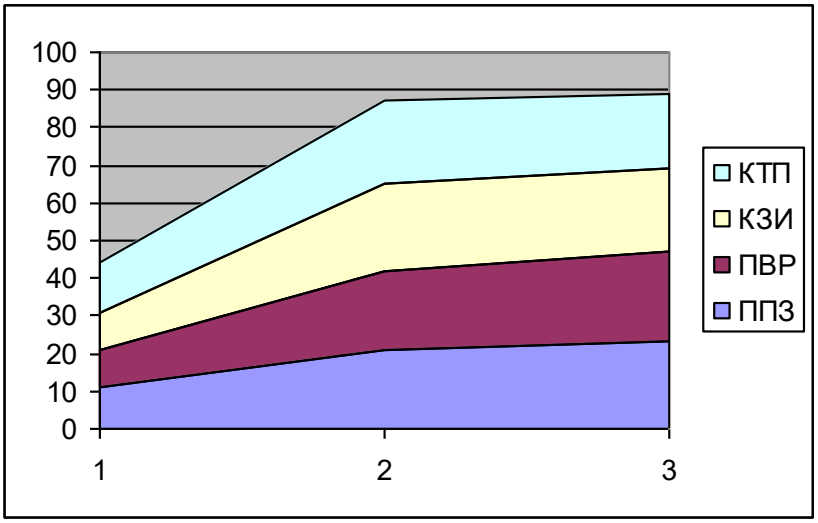


Рисунок 3.2 – Діаграма зміни властивостей груп завдань

Як видно з таблиці і діаграми, диспетчер, який використовує метод планування задач за структурним критерієм, розбив завдання на три групи

так, що кожна з груп використовує приблизно рівну кількість кожного з ресурсів, представлених узагальненими властивостями. А так як в якості критерію «краще» для завдань використовувалася операція «<=», в першу групу завдань диспетчер відніс завдання з меншим споживанням кожного з ресурсів, в другу і третю – з великим споживанням.

У нашому випадку, так як завдання 8, 9 і 10 будуть виконуватися в різних групах як фонові по відношенню до більш пріоритетних завдань {1, 2}, {3, 4, 5} і {6, 7} відповідно, і так як обчислювальні ресурси системи при виконанні кожної групи завдань будуть збалансовані, на виконання всіх завдань робочою станцією буде витрачено на 9 - 11% часу менше, ніж якби завдання виконувалися відповідно до призначених для користувача пріоритетів.

Таким чином, можна зробити висновок, що метод диспетчеризації завдань за структурним критерієм може успішно використовуватися в системах нежорсткого реального масштабу часу, якщо допускається балансування використовуваних ними обчислювальних ресурсів за рахунок можливого порушення послідовності виконання завдань, заданої призначеними для користувача пріоритетами.

3.2 Планування завантаження конвеєра за допомогою векторів розгалуження

В даний час найбільш ефективним програмним засобом максимального завантаження конвеєра центрального процесора при обробці масивів даних є використання шаблону класів valarray з бібліотеки STL мови програмування C ++. До операцій над об'єктами класів valarray зводяться задачі лінійної алгебри, завдання статистичної обробки даних, завдання розв'язання систем звичайних диференціальних рівнянь і рівнянь в часткових похідних. Головною перевагою використання valarray є ефективність проведення однакових операцій відразу над всіма

елементами векторів, що є наслідком агресивної оптимізації, яку компілятори виконують при використанні операцій над цими типами даних.

У комп'ютерній системі екологічної та техногенної безпеки регіону також обробляються великі масиви однотипних даних. Але на відміну від перерахованих вище завдань, обробка цих даних проводиться по гілковим алгоритмам залежно від умов, що виникають на попередніх кроках обробки і впливають на обробку кожного елемента даних екологічних масивів окремо. Це робить практично неможливим програмування алгоритмів верхнього рівня системи екологічної та техногенної безпеки регіону з використанням шаблонів valarray, а значить, не використовується можливість істотного прискорення швидкості обробки екологічних даних за рахунок повного завантаження конвеєра центрального процесора і, як наслідок, зменшення часу виконання завдань системою і збільшення надійності її функціонування.

Тому для системи екологічної та техногенної безпеки регіону доцільно розробити програмні засоби, що забезпечують збалансовану роботу конвеєра центрального процесора за рахунок векторизації з розгалуженням за умовами і циклічну обробку елементів великих масивів даних систем управління або інформаційних систем. Розроблені засоби векторного програмування вбудовуються в сучасні системи підготовки програм, подібні Microsoft Visual Studio, і орієнтовані на вбудовані в компілятори цих систем засоби векторного розпаралелювання. Одними з найважливіших вимог, які висуваються до розробників програм керуючих та інформаційних обчислювальних систем, є надійність функціонування системи, мінімальні тимчасові характеристики по обробці масивів даних фіксованої розмірності і пов'язані з цим вимоги як можна меншого часу відгуку системи на зовнішні події, включаючи час реакції системи на різні запити обслуговуючого персоналу. Прикладами такої обробки масивів можуть служити: отримання значень екологічних параметрів на основі

первинної переробки знятих з датчиків безпосередніх значень (кілька сотень в системі), моделювання екологічної обстановки та прогнозування ризиків на основі поточних значень екологічних параметрів, відображення екологічних даних на мнемосхемах, отримання та подання на екранах графіків розвитку (тенденцій) ключових екологічних параметрів [9]. Основною особливістю перерахованих вище алгоритмів обробки масивів даних в екологічних системах є наявність в них умовних операторів і циклів, контрольованих результатами виконання операцій над окремими елементами цих масивів або ж таблицями рішень, що задаються на етапі проектування програмного забезпечення екологічної системи. При наявності в системі програмування ефективних засобів векторної обробки з можливістю її розгалуження в залежності від динамічно виникаючих умов, виконання по стовпцям або порядкове виконання будь-якого алгоритму з двомірним масивом даних може бути зроблено тільки за один прохід програми цього алгоритму. Така організація обчислень може дати істотну економію часу обробки за рахунок того, що в алгоритмах всі допоміжні інструкції будуть виконуватися тільки один раз, а не повторюватися для кожного стовпця окремо. Крім того, використання векторних обчислень з можливістю розгалужень в інформаційних системах і системах управління дасть економію часу за рахунок агресивної оптимізації обчислень компілятором, особливо при використанні для оптимізації обчислень векторних розширювачів процесора з архітектурою MMX, SSE, SSE2, SSE3 або SSE4. Розглянемо докладніше програмні засоби умовних векторних обчислень, які використовуються в системі екологічного моніторингу.

Шаблон valarray з використанням допоміжних механізмів, таких як зрізи, узагальнені зрізи, маски або непрямі масиви, дозволяє виконувати операції над підмножинами елементів векторів [27]. Але використання цих механізмів утруднено через їх нетрадиційний синтаксис і призводить до громіздких алгоритмічних і програмних конструкцій, які практично не

можна використовувати для регулярної векторної обробки за алгоритмами з рогалуженням. Спосіб обробки даних в шаблоні valarray не дозволяє з його допомогою ефективно обробляти масиви, тому що будь-коли обробка даних може бути перервана на кілька тактів командою умовного переходу. Щоб удосконалити використовуваний в ньому підхід, потрібно позбутися цього недоліку, тобто мінімізувати число умовних переходів при обробці масивів. Для подолання перелічених недоліків шаблону valarray був розроблений інший шаблон класів – vcarray, що включає в себе практично всі можливості шаблону valarray, але забезпечує спільно з іншим керуючим класом можливості розгалуження векторної обробки.

Зовні оголошення шаблону vcarray виглядає практично так само, як і оголошення стандартного бібліотечного шаблону valarray (додаток А).

Крім одномісних і двомісних операторів, які є членами шаблону класів vcarray, є також повний набір тримісних операторів, які не є членами шаблону класів, і визначені в ранзі функцій - помічників шаблону vcarray. Це все логічні, арифметичні, тригонометричні операції, а також кілька функцій, що забезпечують зручне для користувача створення масиву даних.

Хоча синтаксис одномісних, двомісних та тримісних операторів шаблону vcarray практично збігається з синтаксисом цих же операторів для valarray, їх семантика і реалізація істотно відрізняються. Щоб забезпечити вибіркову обробку елементів векторів в залежності від динамічно генерованих векторних умов, всі арифметичні, логічні операції та операції порівняння мають виконуватися тільки для тих елементів векторів, індекси яких перераховані в поточному векторі розгалуження. Поточний вектор розгалуження формується спеціальними функціями, що згадуються нижче. Ці функції додають в спеціальний масив індекси елементів, відповідних заданій умові (тобто вектор розгалуження), і в кінці дописується кількість відповідаючих умовам елементів. Поточний вектор розгалуження завжди розміщується на вершині спеціального керуючого масиву, організованого

за стековим принципом і призначеного для зберігання векторів розгалуження (в загальному випадку, різної розмірності) (рис. 3.3).

На мові програмування С ++ масив для управління розгалуженнями разом з покажчиком на верхній вектор розгалуження представляється об'єктом спеціальної структури vccontrol, яка зберігає, власне, масив індексів і покажчик на останній елемент масиву.

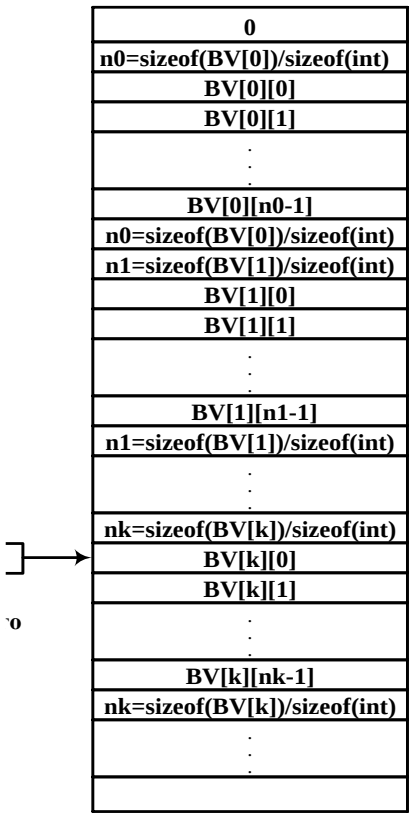


Рисунок 3.3 – Схема управління розгалуженнями при обробці векторів

В області знаходження шаблону vcarray і структури, призначеної для управління ходом векторної обробки, можна визначити ряд керуючих функцій. Перша з цих функцій встановлює поточну розмірність векторної обробки, тобто розмір всіх векторів vcarray, що беруть участь в обробці незалежно від типу їх елементів:

```
        //вказати розмір vcarray, що приймають участь у
векторній обробці
      void SetVcarraySize (size_t nSize);
```

Команди векторної обробки в реалізації на мові С ++ представляються у вигляді макросів, які керують створенням і видаленням поточного вектора розгалуження.

Перша інструкція записує в керуючий масив індекси елементів, що задовольняють заданій умові:

```
#define VcarrayIf(BoolArray) if(AddNewVector(BoolArray) == true){
```

BoolArray – логічна умова; функція AddNewVector () створює новий вектор розгалуження з номерами елементів, що задовольняють відповідній умові.

Друга інструкція записує в керуючий масив індекси елементів, що навпаки, не відповідають останній умові:

```
#define VcarrayElse  } if(OppositeVectorInPlaceOfCurrent() > 0){
```

У новий вектор розгалуження входять всі елементи, які не ввійшли у попередній:

```
BV[k] = BV[k-1] – BV[k]
```

Функція OppositeVectorInPlaceOfCurrent () повертає розмір створеного вектора розгалуження. Таким чином, після макросу VcarrayElse оброблятися будуть елементи вектора, які не відповідали умові в VcarrayIf (BoolArray) (якщо такі є). Завершуватися інструкція VcarrayIf () повинна інструкцією EndVcarrayIf:

```
#define EndVcarrayIf } DeleteCurrentVector();
```

DeleteCurrentVector () видаляє вектор розгалуження, створений VcarrayIf (BoolArray) і VcarrayElse. VcarrayElse можна не використовувати в цій конструкції.

Наступними командами є команди циклічної обробки даних. Це дві команди: з перевіркою умови перед виконанням циклу і після кожної ітерації циклу. Ці команди замінюють поточний вектор розгалуження при кожній ітерації.

Перший варіант циклу, з передумовою:

```
#define VcarrayWhile(BoolArray) CopyVectorInVccontrol();\
while(NewVectorInPlaceOfCurrent(BoolArray) > 0){
```

Функція CopyVectorInVccontrol () створює копію поточного вектора індексів, а NewVectorInPlaceOfCurrent (BoolArray**)** замінює його вектором, в якому індекси елементів відповідають заданій логічній умові (BoolArray). Функція NewVectorInPlaceOfCurrent () повертає число елементів створеного вектора, якщо воно більше нуля – цикл виконується знову.

Завершувати цикл VcarrayWhile (BoolArray) треба командою EndVcarrayWhile:

```
#define EndVcarrayWhile } DeleteCurrentVector();
```

Вона завершує цикл і видаляє вектор, який використовувася в циклі.

Визначається ще один цикл, з перевіркою умови в кінці кожної ітерації VcarrayDo - VcarrayDoWhile (булевий вираз):

```
#define VcarrayDo   CopyVectorInVccontrol(); do {
```

Функція CopyVectorInVccontrol (), як і в попередньому випадку, створює копію поточного вектора індексів:

```
#define VcarrayDoWhile(BoolArray)
}while(NewVectorInPlaceOfCurrent(BoolArray));\
DeleteCurrentVector();
```

Командою VcarrayDoWhile (BoolArray) цикл закінчується. Вона складається з перевірки умови і створення вектора розгалуження з елементів, відповідаючих умові. Після циклу функція DeleteCurrentVector () видаляє використовуваний вектор індексів, а далі робота буде проводитися з попереднім вектором.

3.3 Оцінка ефективності реалізованих засобів векторної обробки даних екологічного моніторингу

Шаблон vcarray побудований так, що всі арифметичні операції проводяться з елементами, що входять в поточний вектор розгалуження. Спочатку це вектор розгалуження, куди входять всі елементи послідовності. Умовні оператори – VcarrayIf (vcarray <bool>), VcarrayElse, оператори повторення – VcarrayWhile (vcarray <bool>) – EndVcarrayWhile,

VcarrayDo – VcarrayDoWhile (vcarray <bool>) формують новий вектор розгалуження, і всередині них обробляються лише елементи послідовності, що задовольняють умові (що входять в vcarray <bool>). Команда EndVcarrayIf, а також команди завершення циклів видаляють останній вектор індексів, і відбувається повернення до попереднього вектору індексів, з яким працювали до циклу (умовного оператора). Таким чином можна будувати обробку будь-якого ступеня вкладеності – кожний цикл і умовний оператор будуть додавати в спеціальний стек новий вектор індексів, а по його завершенні цей вектор індексів буде віддалятися. На відміну від цього, стандартний шаблон valarray не дозволяє робити вкладену вибіркову обробку.

Розроблені засоби векторної обробки даних дають можливість суттєво збільшити швидкість обробки екологічних даних за рахунок наступних правил програмування, реалізація яких була забезпечена в системі екологічної та техногенної безпеки регіону.

Для шаблону vcarray можливості векторних операторів присвоювання з арифметичними і логічними векторними виразами, віднесені до одного індексу елементів оброблюваних векторів, відповідають їх скалярним аналогам – оператору присвоювання і виразами мови С ++.

Для шаблону vcarray забезпечується функціонально повний набір одномісних, двомісних та тримісних арифметичних, логічних операторів і операторів порівняння таких же, як і для шаблону valarray. З цього випливає, що за допомогою шаблону vcarray для всіх елементів векторів з однаковими індексами, що беруть участь в операціях, реалізуються оператори присвоювання з виразами з правого боку оператора "=", що включають будь-які операції і, при необхідності, необмежене число можливо вкладених один в одний підвиразів, обмежених круглими дужками, точно так же, як і при їх скалярної обробці.

Для шаблону vcarray розгалуження векторної обробки за

індивідуальними умовами для кожного з індексів елементів оброблюваних векторів виконується точно так же, як якщо б елементи векторів оброблялися скалярним алгоритмами з використанням оператора "if () {} else {}".

Простий блок розгалуження, представлений графічно, в векторному варіанті обробки з використанням шаблону vcarray програмується послідовністю інструкцій, представленої на цьому ж рисунку в центрі. Виконання тієї ж обробки над елементами тих же векторів в скалярному випадку виконується простим циклом з оператором if всередині циклу (на рисунку показано праворуч). Так як VcarrayIf формує вектор індексів, куди входять всі елементи послідовності, що задовольняють умові, операції проводяться саме з такими елементами.

Те ж відноситься і до інструкції VcarrayIf () {} VcarrayElse {}. Команда VcarrayElse замінює поточний вектор розгалуження новим, з індексами, яких не було в замінюваному векторі. Варіант для інструкції if () {} else {} наведено в додатку А. З нього видно, що, за аналогією з інструкцією if () {}, текст з використанням векторної обробки цілком можна замінити скалярною обробкою.

Для шаблону vcarray цикли VcarrayWhile (vcarray <bool>) – EndVcarrayWhile; VcarrayDo-VcarrayDoWhile (vcarray <bool>) виконуються так само, як якщо б кожен елемент вектора оброблявся в окремому циклі while () або do - while (), де умовою виконання циклу була б перевірка, чи задовольняє елемент вектора якійсь логічній умові .

Цикли VcarrayWhile (vcarray <bool>) - EndVcarrayWhile; VcarrayDo-VcarrayDoWhile (vcarray <bool>) при кожній ітерації замінюють поточний вектор індексів новим, куди записуються номери елементів, які відповідають умові виконання циклу.

Всі операції в циклі проводяться тільки для елементів з останнього вектора індексів; а так як цикл виконується, поки умова виконується хоча б для 1 елемента, значить, це те ж саме, як якщо б цикл виконувався для

кожного елемента вектора окремо.

Наведені інструкції проходження, вибору і повторення забезпечують вибіркову векторну обробку будь-якого ступеня вкладеності.

Так як перераховані вище інструкції вибору і повторення формують новий вектор розгалуження, а інструкції проходження виконують всі дії тільки над елементами, номери яких є в поточному векторі розгалуження, значить, можна робити будь-які вкладення з операторів вибору і циклів, і всередині кожної інструкції обробка буде проводитися тільки для підмножини вектора, що задовольняє умові.

Шаблон vcarray, структури для організації розгалужень і набору макросів для розгалуження обробки VcarrayIf (vcarray <bool>), VcarrayElse, EndVcarrayIf, макросів повторення VcarrayWhile (vcarray <bool>) - EndVcarrayWhile; VcarrayDo - VcarrayDoWhile (vcarray <bool>) і інструкції проходження забезпечують побудову розгалужених програм векторної обробки будь-якого ступеня складності так само, як і при скалярній обробці даних.

Відповідно до правил структурного програмування для створення програми будь-якого рівня складності досить інструкцій проходження, розгалуження і повторення. Шаблон vcarray надає ці 3 види інструкцій, ідентичних тим, що є в процедурних мовах програмування, отже, за допомогою вищеназваних макросів можна написати програму будь-якого ступеня складності.

В додатку А наведено приклад програми обробки масиву даних інструкціями умовних векторних обчислень, що демонструє універсальність запропонованого методу обробки масивів екологічних даних.

Розроблений шаблон для умовних векторних обчислень був використаний в підсистемах моделювання екологічної обстановки і передбачення ризиків в системі екологічного моніторингу, а також в підсистемі відображення екологічної інформації, що, зокрема, дало

можливість відображати оперативні графіки тенденцій десятків екологічних параметрів на екранах з періодом оновлення до 200 мсек. Розглянемо завдання завантаження конвеєра і порівняльного тестування засобів умовних векторних обчислень.

Для оцінки ефективності умовних векторних обчислень, реалізованих в системі екологічної та техногенної безпеки регіону, спочатку розглянемо послідовність виконання інструкцій при скалярній обробці масиву екологічних даних (рис. 3.4).

На рис. 3.4 представлена послідовність інструкцій невеликого фрагмента програми первинної обробки екологічних даних, що знімаються з термопар, термометрів опору і інших лінійних і нелінійних аналогових датчиків. Наведена послідовність обробки представляє собою цикл за даними екологічних параметрів з (*N*+1) ітерацій, кожна з яких забезпечує масштабування і застосування фільтра першого порядку до введеного аналогового значення.

Кожна ітерація включає два умовних переходи, що змінюють послідовність обробки інструкцій в залежності від значень екологічних даних для кожного параметра окремо. Крім того, практично кожна друга інструкція залежить за даними від попередньої інструкції.

Для реальної екологічної обчислювальної системи тіло представленого циклу істотно складніше, так як, наприклад, для нелінійних датчиків потрібно зняті з них значення приводити до лінійної шкали, використовуючи поліноми і / або таблиці перетворення нелінійних значень.

Це означає, що в реальності основний цикл наведеного фрагменту обробки включає в себе вкладені цикли і додаткові розгалуження, можливо, вкладені одне в одне.

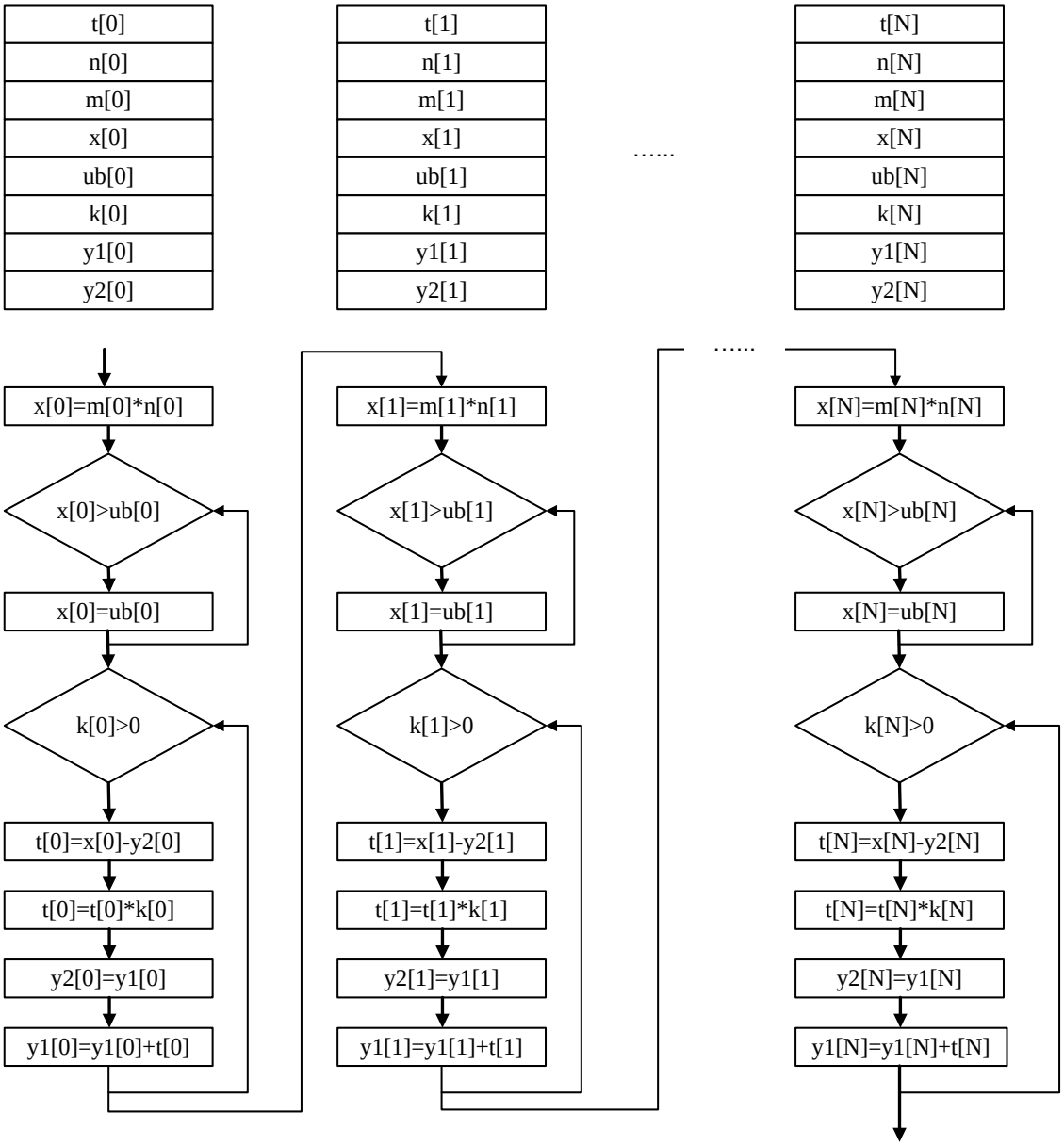


Рисунок 3.4 – Потік управління при скалярній обробці екологічних даних

Крім того, кількість інструкцій тіла циклу наведеного фрагменту в реальній підсистемі первинної обробки екологічних параметрів в сотні разів більше, ніж це представлено на рисунку 3.5. Як видно з наведеної послідовності виконання інструкцій, в кожній ітерації циклу будуть виникати проблеми із завантаженням конвеєра процесора через інструкції умовного переходу і взаємозв'язок інструкцій за даними.

Так як в реальній системі тіло циклу первинної обробки даних включає тисячі інструкцій з сотнями інструкцій умовного і безумовного

переходів, апаратні засоби боротьби з конфліктами в конвеєрі, такі як буфер передвибірки, буфер адрес переходів, буфер циклів, а також реалізована в процесорі стратегія передбачення переходів не можуть згладити всі конфлікти і в достатній мірі завантажити всі щаблі конвеєра.

Тепер розглянемо послідовність інструкцій того ж фрагмента програми первинної обробки екологічних даних, але реалізовану з використанням запропонованих в роботі засобів умовних векторних обчислень (рис. 3.5). Кожен рядок в схемі алгоритму на рисунку 3.5 представляє собою базовий цикл, який виконує $N + 1$ раз або менше одну і ту ж інструкцію над однойменними елементами даних незаблокованих індексів екологічних параметрів. Це саме можна віднести і до операцій умовного переходу, позначених на малюнку ромбами з маленькими прямокутниками з індексами всередині. Кожна така операція складається з двох інструкцій процесора: інструкції умовного переходу та інструкції запису індексу в вектор розгалуження, якщо умова виконується.

Очевидно, що коди інструкцій кожного з циклів умовної векторної обробки повністю розміщуються в буфері циклу процесора першого ступеня конвеєра, що практично виключає етап вибірки інструкцій з оперативної пам'яті. Так як кожна чергова інструкція гарантовано виконується над набором даних, іншим, ніж набори даних всіх попередніх виконуваних в циклі інструкцій, то в одному циклі повністю виключаються всі конфлікти за даними. Для циклів обробки також практично виключаються конфлікти з управління, так як напрямок переходу при виконанні базового циклу визначається самою інструкцією організації циклу, а інші інструкції розгалуження відсутні.

При умовних векторних обчисленнях конфлікти з управління можуть виникати тільки при виконанні векторних операцій розгалуження. Але з огляду на простоту циклу операції розгалуження (дві інструкції), для згладжування цих конфліктів досить наявності в конвеєрі простої дворівневої схеми передбачення переходів.

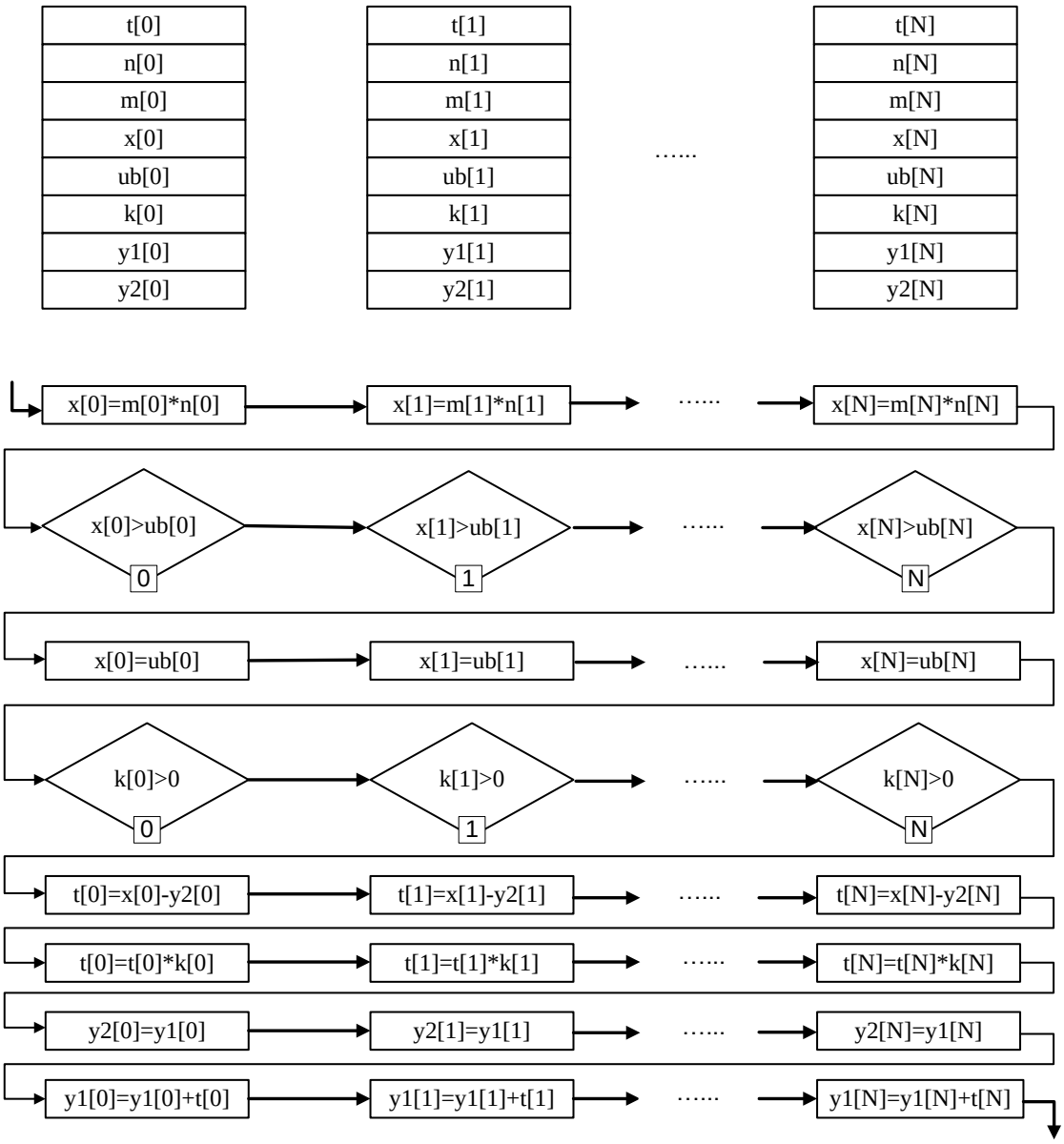


Рисунок 3.5 – Потік управління при умовній векторній обробці екологічних даних

Таким чином, при умовній векторній обробці масивів даних усуваються основні причини неповного завантаження конвеєра процесора, а значить, продуктивність процесора при такій обробці може значно підвищитися в порівнянні зі скалярною обробкою, якщо в процесорі є сучасний конвеєр зі значною кількістю ступенів конвеєра. Так як стандартний шаблон valarray бібліотеки STL мови програмування C ++ в

першу чергу був розроблений з метою ефективного використання конвеєрів та інших засобів прискорення обчислень, було проведено порівняльне тестування розробленого шаблону vcarray з цим стандартним шаблоном на прикладах, які допускають програмування за допомогою того чи іншого шаблону. Зауважимо при цьому, що шаблон vcarray може використовуватися для вирішення більш широкого кола завдань.

Перший тест полягав в наступному. Береться масив, що складається з 250000 дійсних чисел в діапазоні [0.1, 1000.1], отриманий за допомогою генератора випадкових чисел. Цей масив обробляється за допомогою шаблону vcarray в тесті:

```
VcarrayIf(vc1>(float)100.0)
vc1 = 300.0;
EndVcarrayIf
```

Потім здійснюється обробка за тим же алгоритмом, але за допомогою шаблону valarray: val1 [val1> (float) 100.0] = 300.0;

Варіант тесту для vcarray максимально схожий на варіант з використанням valarray**.** Часи обробки всього масиву склали 16 мсек і 15 мсек відповідно. Невелике зниження швидкості обробки (близько 6%) для vcarray пояснюється тим, що цей шаблон запам'ятовує масив оброблюваних індексів, а для valarray це не робиться. Практично однаковий час виконання обох варіантів тесту доводить, що перевага в швидкості умовної векторної обробки масивів в порівнянні зі звичайною скалярною обробкою досягається не за рахунок більш ретельного програмування шаблону vcarray. Результати тесту однакові, так як розмір вектора індексів в обох випадках однаковий, а також кількість операцій над масивом даних в різних варіантах також однакові. Те, що шаблон vcarray зберігає вектор індексів оброблюваних елементів, виявляється перевагою перед шаблоном valarray, якщо після розгалуження виконується більше однієї векторної операції. У другому тесті формується один вектор індексів оброблюваних елементів, а потім під цим вектором індексів виконується 20 арифметичних векторних операцій над вибраними

елементами масиву.

У valarray неможливо здійснити обчислення вектора індексів лише раз для множини подальших операцій. Саме за рахунок цього vcarray виграє за часом в 2,5 рази: часи становлять для цього тесту 62 і 157 мсек відповідно.

Дещо інший варіант попереднього тесту з використанням літералів наведено нижче. Для vcarray:

```
VcarrayIf(vc1>(float)500.0)
     vc1=499.0;
     vc1=399.0;
     // ще 18 операцій привласнення
     //…
EndVcarrayIf
```

Для valarray:

```
val1[val1>(float)500.0]=499.0;
     val1[val1>(float)500.0]=399.0;
     // ще 18 операцій привласнення
```

Цей варіант тесту показав ще кращі результати: 15 мсек (vcarray) і 141 мсек (valarray), тобто поліпшення використання vcarray в порівнянні з valarray в 9,4 рази.

Третій тест є прикладом найбільш типового використання шаблону vcarray:

```
VcarrayIf(vc1 < (float)500.0)
 vc1 /=vc2;
     VcarrayIf(vc1<(float)300.0)
          vc1 /=vc2;
          VcarrayIf(vc1<(float)200.0)
               vc1 /=vc2;
          EndVcarrayIf
     EndVcarrayIf
EndVcarrayIf
```

Саме для такої вкладеної умовної обробки і створювався vcarray. Його аналог для valarray виглядає наступним чином:

```
val1[val1<(float)500.0] /= val2;
val1[val1<(float)500.0 && val1<(float)300.0]/= val2;
val1[val1<(float)500.0     &&     val1<(float)300.0     &&
val1<(float)200.0]/= val2;
```

Тест показав однакові результати для обох шаблонів (по 16 мсек). Незважаючи на те, що розмір вектора індексів для vcarray послідовно зменшується: N = {250000, 126050, 37835}, а valarray перебирає 3 рази по

250000 елементів, зменшення кількості елементів, що перебираються, не стало причиною виграшу в часі – занадто незначною була кількість елементів, на яку зменшився вектор індексів. Тест з множенням в циклі показав виграш варіанту vcarray в 2 рази (109 мсек до 219 мсек на користь vcarray). Якщо цикл виконується понад 44 разів, з'являється помітний виграш у часі. Тест зі складанням показав перевагу vcarray в 3 рази (313 мсек і 93 мсек).

Шаблон класів vcarray показав менший час виконання у вкладених умовних операторах або циклах, коли число елементів вектора індексів суттєво зменшується від ітерації до ітерації, або коли проводиться безліч операцій з елементами після обчислення чергового вектора індексів. В описаних семи тестах ефективність використання шаблону vcarray порівнювалася зі стандартним шаблоном valarray на фрагментах програм, на яких максимальне завантаження конвеєра при скалярній обробці зазвичай не забезпечується через умови, що не дозволяють обробляти однаковим чином всі елементи векторів. На звичайних же векторних операціях розроблений шаблон vcarray перевершує за швидкістю стандартний шаблон valarray в постачанні Microsoft Visual Studio.NET версії 7.1 в середньому в 2.3 рази.

У таблиці 3.9 представлені часи виконання основних векторних операцій засобами шаблонів valarray і vcarray.

Тестування проводилося на процесорі Intel з тактовою частотою 2.4 ГГц. У всіх тестах для вимірювання часових інтервалів використовувалася API-функція GetTickCount (), що вимірює часи виконання тестів з точністю до мілісекунди.

Ті ж часи у вигляді гістограми представлені на рисунку 3.6.

Таблиця 3.9 – Час виконання основних векторних операцій шаблонами vcarray і valarray

Векторна операція	Час виконання (в милісекундах)	
	vcarray	valarray
Оператор /= для 10.000.000 чисел типу double	172	890
Оператор /= для 1.000.000 чисел типу double	16	94
Функція resize() – зміна розміру масива від 1.000.000 до 500.000 чисел	15	15
Оператор ~ для 1.000.000 цілих чисел	31	297
Оператор ! для 1.000.000 значень типу bool	47	328
Тримісний оператор «-» для 1.000.000 чисел типу float	32	312
Тримісний оператор + для 1.000.000 чисел типу float	32	312
Оператор *= для 1.000.000 чисел типу double	15	94
Оператор /= (поділ масиву на масив) для 1.000.000 чисел типу double	16	157
Оператор *= (множення масиву на масив) для 1.000.000 чисел типу double	16	157
Функція sum() для 1.000.000 чисел типу double	31	78

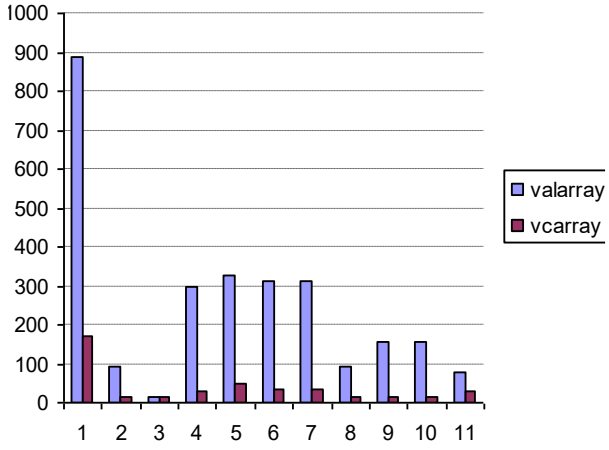


Рисунок 3.6 – Час виконання операцій різними засобами обробки векторів (горизонтальна вісь: № операції; вертикальна вісь: час виконання)

З таблиці і гістограми видно, що шаблон класів **vcarray** за часом виконання більшості операцій значно випереджає стандартний клас **valarray**.

ВИСНОВКИ

Основні результати, отримані кваліфікаційній роботі, полягають у наступному:

– здійснено аналіз існуючих методів розподілу обчислювальних ресурсів в системах екологічного моніторингу;

– досліджено методи диспетчеризації вузлів мережі ICEM, що здійснюють оптимальний розподіл завдань між робочими станціями і контролерами системи екологічного моніторингу;

– досліджено методи пріоритетного планування завдань ICEM з метою мінімізації часу виконання завдань і збільшення пропускної спроможності системи;

– досліджено метод формування мережі пунктів спостереження з с застосуванням графової моделі кластеризації;

– досліджено програмні засоби векторної обробки даних екологічного моніторингу, що ефективно завантажують конвеєр процесора ICEM за рахунок зменшення його простою через розгалуження обчислень за умовами;

– здійснено програмну реалізацію та моделювання задач диспетчеризації інтелектуальної системи екологічного моніторингу.

Експериментальні дослідження методів розподілу завдань екологічного моніторингу між комп'ютерними вузлами ICEM показали, що час виконання програм диспетчеризації контролерами з застосуванням запропонованого методу скоротився майже на 20%.

Програмно реалізований шаблон класів vcarray забезпечує більш повне завантаження конвеєра процесора і дозволяє підвищити швидкість обробки масивів даних екологічного моніторингу в порівнянні зі стандартним шаблоном класів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Куссуль Н. М., Скакун С. В., Шелестов А. Ю. Геоінформаційна інфраструктура моніторінгу навколишнього середовища та надзвичайних ситуацій. *Наука та інновації*. 2010. № 4. Том 6. С. 21–28.

2. Бахарєв В. С., Маренич А. В. Аналітичний огляд результатів наукових досліджень з проблем моніторингу довкілля в Україні. *Екологічна безпека*. 2016. № 2 (22). С. 35–42.

3. Закон України «Про Основні засади (стратегію) державної екологічної політики України на період до 2030 р.». *Відомості Верховної Ради (ВВР).* 2019. № 16. С.70.

4. Зацерковний В. І., Бурачек В. Г., Железняк О. О., Терещенко А. О. Геоінформаційні системи і бази даних. Книга 1: монографія. Ніжин: НДУ ім. М. Гоголя, 2014. 492 с.

5. Мокін В.Б., Крижановський Є.М. Геоінформаційні системи в екології: навчальний посібник. Вінниця: ВНТУ, 2014. 192 с.

6. Удовенко С.Г., Чала Л.Е., Гриньова О.Є., Яричкіна Т. Екологічний моніторинг ландшафтних ділянок з використанням регуляризованих штучних нейронних мереж. *Біоніка интелекту*. 2020. Вип. 1 (94). С. 13–22.

7. Яцишин А. В., Попов О. О., Артемчук В. О. Використання інформаційних технологій в задачах управління екологічною безпекою. *Праці Одеського політехічного університету*. 2013. Вип. 2 (41). С.289–294.

8. Tsvetkov V. Ya. Spatial Information Models. *European Researcher*. 2013. Vol. (60). № 10. P. 2386–2392.

9. Жежнич П. І., Осика В. О. Функціональні та структурні вимоги до побудови сучасних географічних інформаційних систем. *Вісн. Нац. ун-ту «Львівська політехніка»*. Львів, 2011. № 715: Інформаційні системи та мережі. С.104–113.

10. Пітак І. В., Негадайлов А. А., Масікевич Ю. Г., Пляцук Л. Д.,

Шапорев В. П., Моісеєв В. Ф. Геоінформаційні технології в екології: навчальний посібник. Чернівці, 2012. 273с.

11. Мокін В. Б., Крижановський Є. М. Геоінформаційні системи в екології: навчальний посібник. Вінниця: ВНТУ, 2014. 192 с.

12. Якубайлик О. Э., Павличенко Е. А., Ромасько В. Ю. Организация оперативной обработки данных дистанционного зондирования земли. *Международный научно-исследовательский журнал*. 2018. №11. С. 79–82.

13. Козин Е. С. Объектно-ориентированный подход привязки космических снимков. *Информационные технологии.* 2006. №12. С. 37–39.

14. Вольська С. Ю., Маргаф О., Руденко Л.Г. Геоінформаційна технологія: етапи розвитку, стан в Україні. *Укр. геогр. Журнал*. 2013. №4. С.6–14.

15. Белов П. Г. Системный анализ и моделирование опасных процессов в техносфере. М.: Академия, 2003. 512 с.

16. Растоскуев В. В., Шалина Е. В. Геоинформационные технологии при решении задач экологической безопасности. СПб: ВВМ, 2006. 256 с.

17. Шенброт И. М., Алиев В. М. Проектирование вычислительных систем распределенных АСУ ТП. М.: Энергоатомиздат, 2009. 88 с.

18. Елисеев В. В., Ларгин В. А., Пивоваров Г. Ю. Программно-технические комплексы АСУ ТП: учебн. пособие. К.: Издательско-полиграфический центр «Київський університет», 2003. 429 с.

19. Ицкович Э. Л. Классификация современных контроллеров и их сетевых комплексов. *Компас промышленной реструктуризации*. 2014. № 7. С. 21–28.

20. Щербаков Е. В., Щербакова М. Е., Рязанцев А. И. Особенности SCADA-системы пакета "КВАРЦ". Вестник ХГТУ. Херсон. 2003. № 2 (18). С. 188–192.

21. Рязанцев А. И., Кардашук В. С. Система экологического мониторинга окружающей среды. *«Радіоелектроніка. Інформатика. Управління»*. Запоріжжя, ЗНТУ. 2016. № 2. С. 128–132.

22. Єрохін А. Л., Турута О. П. Оцінка ефективності обслуговування користувача в інформаційній мережі. *Право і безпека*. 2009. №3. С.164–166.

23. Селиверстов Е. Ю. Обзор методов решения задачи планирования параллельных алгоритмов. *Электронный научно-технический журнал «Инженерный вестник»*. 2014. №12. С. 541–555.

24. Різник О. М. Концепція відкритої динамічної системи. *Математичні машини і системи*. 2020. № 3. С. 3–22.

25. Рязанцев А. И., Щербакова М. Е. Методы балансирования нагрузки в управляющих вычислительных системах. *Інформатика та комп'ютерні технології*: матеріали третьої науково-технічної конференції молодих учених та студентів. Донецьк, ДонНТУ. 2007. С. 231–233.

ДОДАТОК А

Приклад обробки даних екологічного моніторингу з використанням умовних векторних обчислень

Шаблон **vcarray** програми векторної обробки даних екологічного моніторингу:

```
    template <class T > class vcarray
    { public:
    // конструктори класів
    vcarray();                                  //vcarray з
розміром size()==0
    explicit vcarray(size_t n);          //vcarray з n
елементами зі значенням T()
    vcarray(const T& val, size_t n); //vcarray з n
елементами зі
                                        //значенням val
    vcarray(const T* pv, size_t n);   //vcarray з n
елементами зі
                                        //значеннями
pv[0], pv[1],…,pv[n-1]
    vcarray(const vcarray& vc);       //копія vc
    vcarray(const valarray& va);        //перетворення
valarray в vcarray
    …
// деструктор класів
    ~vcarray();
    // оператори – члени класу:
// оператор "="
    vcarray<T>&   operator=(const   vcarray<T>&   vc);
//копіювання vc
    valarray<T>&     operator=(const     T&     val);
//присвоєння val
                                                    //
кожному елементу
    // оператор "[]"
    T operator[](size_t) const;
    T& operator[](size_t);
    // унарні оператори
    vcarray operator- () const;          // -vc[i] для
кожного елемента
    vcarray operator+ () const;         //  +vc[i]  для
кожного елемента
    vcarray operator~ () const;         //  ~vc[i]  для
```

```
кожного елемента
    vcarray operator! () const;        //  !vc[i]  для
кожного елемента
    // обчислювані присвоювання
    vcarray<T>& operator*=(const vcarray<T>& vc);
    vcarray<T>& operator*=(const T& val);
    // аналогічно для: /=, %=, +=, -=, ^=,&=, |=,<<=
и >>=
    // f(vc[i]) для кожного елемента
    vcarray<T> apply (T f(T)) const;
    vcarray<T> apply (T f(const T&)) const;
    // сума всіх елементів
    T sum() const;
    // мінімальний і максимальний елементи
    T min() const;
    T max() const;
    …
    private:
    T* ptr;
    size_t length;
    };
```

Крім одномісних і двомісних операторів, які є членами шаблону класів vcarray, є також повний набір тримісних операторів, які не є членами шаблону класів, і визначені в ранзі функцій - помічників шаблону vcarray. Це логічні, арифметичні та тригонометричні операції, а також кілька функцій, що забезпечують зручне для користувача створення масиву даних:

```
    //бінарні операції:
    template<class  T>  vcarray<T>  operator*(const
vcarray<T>&, const vcarray<T>&);
    template<class  T>  vcarray<T>  operator*(const
vcarray<T>& , const T& );
    template<class T> vcarray<T> operator*( const T&,
const vcarray<T>&);
    //аналогічно: /, +, -, ^, &, |,<<, >>, &&, ||, %
    // логічні операції:
    template<class T> vcarray<bool> operator== (const
vcarray<T>&, const vcarray<T>&);
    template<class  T>  vcarray<  bool  >  operator==
(const vcarray<T>& , const T& );
    template<class  T>  vcarray<  bool  >  operator==  (
const T&, const vcarray<T>&);
```

```
    //аналогічно:  !=, <, >, <=, >=
    // математичні функції:
    template<class     T>     vcarray<T>     abs(const
vcarray<T>& );
    //аналогічно: acos, asin, atan, cos, cosh, exp,
log, log10, sin, sinh, sqrt, tan и tanh
    template<class     T>     vcarray<T>     pow(const
vcarray<T>& , const vcarray<T>& );
    template<class     T>     vcarray<T>     pow(const
vcarray<T>& , const T& );
    template<class T> vcarray<T> pow(const T& , const
vcarray<T>& );
    //аналогічно atan2
```

Приклад обробки масиву даних екологічного моніторингу з використанням умовних векторних обчислень:

```
    int _tmain(int argc, _TCHAR* argv[])
    {
    int
nValuesForVcarray[]={354,1,2,3,44,5,6,44,8,9};
    vcarray<int>          vc0(nValuesForVcarray,sizeof
nValuesForVcarray/
    sizeof *nValuesForVcarray),   vc1(4,10);
    vcarray<int> vc3(vc0);
    cout<<"vc0 = ";
    print(vc0);          //  Виведення  значень  усіх
елементів масиву
    cout<<"vc1 = ";
    print(vc1);
    SetVcarraySize(10);
    VcarrayIf(vc0   ==   44)   //   Формування   вектора
індексів елементів,
         // що відповідають умові (умовний
         // вираз може бути будь-яким)
        vc0/=15;                  // Обчислення нових
значень елементів,
                                        //         що
відповідають умові (вираз
                      // може бути будь-яким)
        cout<<"VcarrayIf(vc0   ==   4)\n       vc0   /=
15;\nvc0 = ";
        print(vc0);             // Виведення значень
елементів масиву,
```

```
                    //  що  відповідають  умові,  на
екран
    VcarrayElse          //     Формування     вектора
індексів та
        // обробка не оброблених елементів
        vc3 = 17;
        cout<<"VcarrayElse\n     vcarray<int>  vc3  =
17;\nvc3 = ";
        print(vc3);
    EndVcarrayIf    //  Повернення  до  попереднього
вектору індексів
    cout<<"VcarrayWhile(vc0 < 110)\n    vc0 *=2;\nvc0
= ";
    int nIterationNumber = 0;
    VcarrayWhile(vc0  <  110)  //  Циклічна  умовна
векторна
       // обробка (цикл виконується, поки є
       // хоча б один елемент, що відповідає
       // умові цикла)
        vc0 *= 2;
        cout<<"Iteration          Number          "<<
nIterationNumber++ <<":\n";
        print(vc0);
    EndVcarrayWhile     // Повернення до попереднього
вектору індексів
    vc3 = 11;
    cout<<"VcarrayDo\n                    vc3      +=
1;\nVcarrayDoWhile(vc3 < 21)\nvc30 = ";
    nIterationNumber = 0;
    VcarrayDo           // Циклічна умовна векторна
       // обробка з аналізом умови наприкінці циклу
        vc3 += 1;    //+1 ко всем элементам, меньшим
21
        cout<<"Iteration          Number          "<<
nIterationNumber++ <<":\n";
        print(vc3);
    VcarrayDoWhile(vc3 < 21)
    return 0;
    }
```

ДОДАТОК Б

Відомість кваліфікаційної роботи магістра

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	101 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Чмуж В.С.			Методи розподілу обчислювальних ресурсів в інтелектуальній системі екологічного моніторингу	Шифр групи	Код напр./спец.
Перевірив		Удовенко С.Г.				СШІздм-21-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					