НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА СИСТЕМНОГО ПРОГРАМУВАННЯ І
СПЕЦІАЛІЗОВАНИХ КОМП'ЮТЕРНИХ СИСТЕМ

«На правах рукопису»
УДК 004.05

«До захисту допущено»
Завідувач кафедри СПСКС

_____Віталій РОМАНКЕВИЧ

"___"_____2020р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 123 Комп'ютерна інженерія
(Спеціалізовані комп'ютерні системи)

на тему: «Спосіб організації тестування у багатошинних системах»

Виконала: студентка ІІ курсу, групи КВ-93мп
Олена КОВАЛЕНКО _____

Науковий керівник проф., д.т.н., проф. Олексій РОМАНКЕВИЧ _____

Рецензент проф., д.т.н., проф. Валерій СИМОНЕНКО _____

Консультант з нормоконтролю доцент, с.н.с, к.т.н. Юлія БОЯРІНОВА_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – другий (магістерський)
за освітньо-професійною програмою
Спеціальність 123 Комп'ютерна інженерія
(Спеціалізовані комп'ютерні системи)

ЗАТВЕРДЖУЮ
Завідувач кафедри СПСКС
_____ Віталій РОМАНКЕВИЧ
(підпис)
1 листопада 2019р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Коваленко Олени Павлівни

1. Тема дисертації **«**Спосіб організації тестування у багатошинних системах**»**,
науковий керівник дисертації професор кафедри СП і СКС, д-р техн. наук Романкевич О.М., затверджені наказом по університету від «12» листопада 2020 р. № 3298-С.

2. Термін подання студентом дисертації 14 грудня 2020 р.

3. Об'єкт дослідження: багатошинні багатопроцесорні системи.

4. Предмет дослідження: тестування багатошинних багатопроцесорних систем.

5. Перелік завдань, які потрібно розробити: аналіз існуючих способів організації тестування, аналіз розвитку багатошинних систем, алгоритм пошуку «0-ланцюжків», пошук умов для побудови відмовостійких систем при певних заданих параметрах, побудова GL-моделі для багатошинної системи для прогнозування поведінки в потоці відмов.

6. Перелік ілюстративного матеріалу - презентація

7. Перелік публікацій: «Алгоритм пошуку “0” - ланцюжків у діагностичних графа» (XXII конференція молодих вчених ПМК 2019 року), «Багатошинні багатопроцесорні системи з обмеженням по відмовостійкості» (VII міжнародна науково-технічна internet-конференція 2020 року (НУХТ)).

8. Дата видачі завдання 1 листопада 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення літератури за тематикою проекту	28.11.2019	
2.	Розроблення та узгодження технічного завдання	26.12.2019	
3.	Аналіз існуючих рішень	22.02.2020	
4.	Підготовка матеріалів першого розділу дипломного проекту	28.03.2020	
5.	Підготовка матеріалів другого розділу дипломного проекту	25.04.2020	
6.	Підготовка матеріалів третього розділу дипломного проекту	15.09.2020	
7.	Підготовка матеріалів четвертого розділу дипломного проекту	11.10.2020	
8.	Підготовка графічної частини дипломного проекту	25.11.2020	
9.	Оформлення документації дипломного проекту	21.11.2020	
10.	Попередній огляд матеріалів диплому на кафедрі	26.11.2020	

Студент _____ Олена Коваленко
(підпис)

Науковий керівник дисертації _____ Олексій РОМАНКЕВИЧ
(підпис)

РЕФЕРАТ

Дисертація виконана на 80 аркушах, вона містить перелік посилань на використані джерела з 29 найменувань. У роботі наведено 35 рисунків, 7 таблиць.

Актуальність теми. На сьогодні сфера використання багатошинних багатопроцесорних систем безперервно розвивається в різноманітних областях науки, бізнеса й виробництва. А також багатошинні системи є складовими електростанцій та літаків, де помилки є критичними, тому покращення параметрів даних систем буде актуальним і в майбутньому. Організація тестування з високою точністю у даних системах забезпечить вчасне виявлення несправних елементів і їх заміну.

Зі збільшенням попиту на подібні системи зростає потреба і в забезпеченні оцінки основних параметрів. Відмовостійкість дозволяє системі продовжувати функціонувати в разі виходу з ладу одного або декількох її компонентів. Відмовостійкі комп'ютерні системи існують вже багато років, але на сьогодні для того, щоб передбачити поведінку такої системи в потоці відмов найчастіше використовуються GL-моделі. Оскільки в GL-моделях існує можливість об’єднання базових моделей в одну і на сьогодні з’являються все складніші системи з більшою кількістю процесорів, то GL-моделі будуть і в майбутньому мати попит.

Якщо система будується як відмовостійка, то існує необхідність у введені надлишкового апаратного й програмного забезпечення. Тому важливою задачею є створити такі початкові умови, щоб забезпечити відмовостійкість при мінімально необхідній надлишковості задля економії затрат на виробництво такої системи.

Зв’язок роботи з науковими програмами, планами, темами. Дисертаційна робота виконувалась згідно з планом науково-дослідних робіт кафедри системного програмування і спеціалізованих комп’ютерних систем

Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

Мета і задачі дослідження. Метою дисертаційної роботи є організація тестування багатошинних багатопроцесорних систем.

Для досягнення вказаної мети було розв'язано такі задачі:

- досліджено уже існуючі методи тестування та діагностування;
- розроблено алгоритм пошуку 0-ланцюжків у діагностих графах;
- досліджено багатошинні багатопроцесорні системи, їх класифікацію та специфіку;
- знайдено залежності, що надають інформацію, як обрати певні параметри багатошинних багатопроцесорних систем при певних вхідних даних;
- побудовано GL-модель багатошинної системи;
- розроблено програму, що виконує роботу згідно з розробленим алгоритмом.

Об'єктом дослідження є багатошинні багатопроцесорні системи.

Предметом дослідження є організація тестування багатошинних багатопроцесорних систем та побудова їх GL-моделей.

Методи дослідження. Для розв'язання поставленої задачі використовувалися методи теорії алгоритмів та програмування; методи теорії ймовірності та математичної статистики, методи теорії автоматів та організації самотестування.

Наукова новизна одержаних результатів складається з таких положень:

уперше поставлено задачу знаходження залежностей у багатошинних системах, що дозволяють обирати кількість міжшинних процесорів, а також число процесорів на кожній шині згідно до початкових даних;

удосконалено алгоритм пошуку «0-ланцюжків» у діагностичних графах багатопроцесорних систем;

побудовано GL-модель для системи, що складається з трьох шин, яка відображає поведінку у потоці відмов.

Практичне значення одержаних результатів. Запропоновані методи можуть бути використані при розробці багатошинних багатопроцесорних систем, а також для забезпечення їх тестування. Алгоритм пошуку «0-ланцюжків» в результаті вказує справний процесор, таким чином відбується пошук інших справних процесорів, які і беруть участь у взаємному тестуванні системи. На практиці алгоритм має переваги, адже працює швидко і коректно для великої кількості процесорів.

Побудована GL-модель показує поведінку багатошинної системи в потоці відмов, тому дає можливість системі розрахунку надійності виявляти, як поведе себе система при заданому наборі несправних процесорів. Модель заданої системи легко будується на основі базових моделей при розбитті відповідно на підсистеми. Базова модель є неорієнтовним графом, де кожне ребро є булевою функцією, значення якої дорівнює нулю або одиниці. Всі ребра зі значенням нуля відкидаються, після чого зв'язність графа відповідає роботоздатності системи, а незв'язний граф свідчить про протилежне. Таким чином побудова GL-моделі надає відомості про поведінку системи. Більше того у подальшому за допомогою побудованої моделі можна оцінювати параметри надійності.

Апробація результатів дисертації. Основні положення й результати роботи представлено на науковій конференції «Прикладна математика та комп'ютинг» (2019 р.).

Публікації. Результати дисертації викладено в наукових працях, у тому числі:

- XII конференція молодих вчених ПМК 2019 року;
- VII міжнародна науково-технічна internet-конференція 2020 року (НУХТ).

Ключові слова: багатошинні багатопроцесорні системи, тестування багатопроцесорних систем, GL-моделі.

ЗМІСТ

Список термінів, скорочень та позначень

AWT (Abstract Window Toolkit) – бібліотека графічного інтерфейса;

GUI (Graphical user interface) – графічний інтерфейс користувача;

JDK (Java Development Kit) – комплект для розробників мови Java;

JVM (Java Virtual Machine) – віртуальна машина Java;

Swing – бібліотека для створення GUI в Java;

ВБС – відмовостійка багатопроцесорна система;

ЕОМ – електронно–обчислювальна машина;

ОС – операційна система.

ВСТУП

На сьогодні багатошинні системи (БС) використовуються в основних сферах науки і техніки, в тому числі при управлінні об’єктів авіації та єнергостанцій. Такий високий рівень відповідальності вимагає відповідного рівня забезпечення параметрів швидкодії та працездатності, а також вчасного виявлення несправностей при тестуванні. Забезпечення відмовостійкості є одним з найпоширенішим засобом підвищення надійності багатошинних систем. Тому ще при розробці БС обираються параметри системи, отже однією з поставлених задач даної роботи є пошук залежностей, що допоможуть побудувати відмовостійку систему з вказаною кількістю максимально допустимого числа відмов.

Відмовостійкі системи існують вже багато років. Наприклад, резервні сервери широко застосовуються в центрах обробки даних. Зазвичай відмовостійкі системи налаштовані таким чином, що при виході з ладу будь-якого окремого елемента система реконфігурується таким чином, що справні процесори беруть на себе функції процесорів, що вийшли з ладу. В результаті, дані не втрачені, комп'ютери продовжують працювати. В БС процесори на шині в разі їх несправності можуть бути замінені міжшинними процесорами.

Однією з найважливіших вимог до багатошинних багатопроцесорних систем є забезпечення не лише відмовостійкості, а й можливість взаємного тестування між процесорами, а також встановлення їх несправності. Тому у даній роботі однією із поставлених задач є пошук способу полегшення тестування й знаходження несправних процесорів.

Зі збільшенням обсягу даних, що необхідно одночасно оброблювати, одношинні системи втратили свою актуальність, адже виникала затримка через очікування почергової обробки. Тому виникла необхідність у розпаралелюванні процесів і багатошинних системах, які і на сьогодні

використовуються у більшості сферах, адже в порівнянні з одношинними системами БС значно виграють по швидкодії.

1 АНАЛІЗ БАГАТОПРОЦЕСОРНИХ СИСТЕМ ТА СПОСОБИ ОРГАНІЗАЦІЇ ЇХ ТЕСТУВАННЯ

1.1 Обгрунтування теми магістерської роботи

На сьогодні багатошинні системи (БС) охоплюють велику кількість різноманітних сфер життя людини. Багатопроцесорні системи ще з перших років появи зайняли своє місце при складних обчисленнях. З роками складність багатопроцесорних обчислювальних систем все зростала відповідно до рівня розвитку інших технологій та введення їх у повсякденне життя людини. Забезпечення основних параметрів багатопроцесорних систем є важливою задачею і сьогодні. Зокрема здійснення процесу тестування без часових втрат також забезпечує безперервну роботу БС. Несправності подібних систем, що не були виявлені вчасно, можуть привести до катастрофічних наслідків, наприклад для літальних чи космічних апаратів. Під час роботи цих апаратів відсутня можливість обслуговування, тому важливо забезпечити відмовостійкість до певної кількості несправностей, що так чи інакше будуть виникати[1].

Ще при розробці багатошинних систем необхідно правильно обрати параметри системи, щоб система залишалася роботоздатною при максимально допустимій кількості відмов. При розрахунку параметрів необхідно враховувати, що при необхідності після виникнення збоїв у системі є взаємозаміння елементи. Наприклад для багатошинних систем шина у разі потреби може замінити несправні процесори справними з міжпроцесорних. При цьому обернена заміна неможлива.

Необхідно забезпечувати відмовостійкість система, хоча в той же час БС може бути не безвідмовною. Адже поняння безвідмовності порушується, коли хоча б один елемент виходить з ладу, а відмовостійкість дозволяє системі продовжувати повне функціонування без втрат навіть при виникненні певної множини несправностей.

Для розуміння того, чи здатна система надалі виконувати роботу, ще в процесі експлуатації необхідно проводити взаємне тестування системи. При роботі системи на межі можливостей важливою задачею є вчасне розуміння неможливості подальшої роботи системи та виправлення ситуації.

При проектуванні багатошинних багатопроцесорних систем використовуються сучасні методи побудови, що дозволяють не лише закласти програмну реалізацію, а й систему, що передбачає поведінку БС у потоці відмов, та виявляє при яких наборах відмов вказаної кратності система ще продовжуватиме функціонування. Якщо ж число відмов перевищить дозволену кількість, то далі система ще деякий час буде виконувати частково свою роботу, поки повністю не вийде з ладу. Тому важливою задачею є вчасно зупинити роботу БС, не допустивши аварійних випадків.

Зазвичай для підвищення надійності таких систем вводиться апаратна надлишковість. Важливо розрахувати кількість додаткових елементів правильно, щоб це значно підвищило основні показники система, але не зменшуючи швидкодію та розміри. Тобто вводиться мінімально необхідна надлишковість.

Існує декілька етапів розробки відмовостійких багатопроцесорних систем (ВБС), на кожному з яких проводиться тестування. На першому етапі виконується перевірка правильності роботи під час безпосередньо виготовлення БС. На другому етапі приводиться перевірка під час експлуатації правильності виконання основного функціоналу. Третій етап недовготривалий, але періодично контролює стани елементів системи під час короткочасних перерв у роботі техніки.

Після проходження будь-якого з цих етапів відбуваються певні модифікації, наприклад додавання ще одного модулю, чи збільшення числа елементів, вбо ж їх взаємна заміна. Як наслідок, надійність збільшується і система є стійкою до оптимально допустимого числа відмов. Тому так важливо є організація процесів тестування[2,3].

1.2 Поняття багатопроцесорних обчислювальних систем. Історія їх виникнення

Багатопроцесорні системи (БС) основані на об’єднанні процесорів на спільному полі оперативної пам’яті. Це поле називається роздільною пам’яттю (Shared Memory). Управління забезпечується однією загальною операційною системою. При цьому досягається більш швидкий обмін інформацією між процесорами, ніж між ЕОМ в багатомашинних обчислювальних системах (комплексах), і більш висока сумарна продуктивність системи. Іноді їх називають “істинними” мультипроцесорами[4].

В таких системах, як правило, число паралельних процесорів невелике і управляє ними центральна операційна система. Процеси обмінюються інформацією через загальну оперативну пам’ять. При цьому виникають затримки через міжпроцесорні конфлікти. При створенні великих мультипроцесорних ЕОМ (мейнфреймів, суперЕОМ) затрачуються чималі зусилля по збільшенню пропускної здатності оперативної пам’яті. В результаті апаратні витрати збільшуються в квадратичній залежності, тим не менш продуктивність системи не збільшується пропорційно числу процесорів. Складні засоби зниження міжпроцесорних конфліктів в оперативній пам’яті суперкомп’ютерів серії CRAY X-MP/Y-MP дозволяють отримати коефіцієнт прискорення не більше 3,5 для чотирьохпроцесорної конфігурації системи.

По топології міжмодульних функціональних і управляючих зв’язків і організації роботи виділяються два типи багатопроцесорних систем МКМД:

- З загальною шиною;
- З використанням багатовходової пам’яті (багатошинні обчислювальні системи)[5].

В проекті Solomon було розроблено процесори, що могли працювати і викоконували операції над вмістом одного чи кількох регістрів, це припало на 1962 рік.

З 70-х років розпочинається історія розвитку багатопроцесорних обчислювальних систем, а також з'являється перши суперкомп'ютер CRAY-1.

Було реалізовано ідентичний принцип роботи і в машині ILLIAC IV. Векторними називалися процесори, що виконували дії над векторами, при цьому використовуючи одну команду. Такі процесори використовувалися при розробці суперкомп'ютерів Cray-1[6].

З'явилася нова класифікація для комп'ютерів зі введенням векторних, іншим дісталася назва скалярні. А з їх розвитком було вирішено, що такі суперскалярні процесори забезпечують таку роботу, що процесор сам обирає, які операції виконуються паралельно. Ця ідею мала місце в процесорах Intel i960 та AMD 29050.

Удосконаленням фон-нейманівської архітектури є розпаралелювання потоків. Подібна технологія використовується успішно не лише при одночасній багатопоточності SMT, а й при багатопоточності на рівні кристала.

А різниця цих підходів заключається в тому, що поняття потоку не однакове. DEC Alpha EV4 21064 був першим процесором з багатопоточністю. Даний процесор забезпечує збільшення продуктивності на третину за рахунок того, що відбувається поділ команд на два потоки, а процесор сам обирає з якого з них команда буде виконуватися[7].

Першими представниками архітектури CMP (ядра на кристалі) стали процесори, призначені для використання в серверах, це був простий тандем, в таких приладах на одній підкладці розміщуються два, по суті, незалежних ядра. Крім економії місця подібне рішення дає відчутну економію енергії, оскільки частина компонентів є спільною для обох ядер.

Про паралелізм комп'ютерних систем вчені наполегливо говорили протягом всіх останніх років. Однак до тих пір, поки подолання пов'язаних з цим складнощів, наприклад з необхідністю поділу додатків на потоки, компенсувалося безперервно зростаючою продуктивністю процесорів, ніхто з виробників зі зрозумілих причин інвестувати в цю перспективу не хотів. Тому до появи багатоядерних процесорів паралельні обчислення залишалися атрибутом суперкомп'ютерів.

Ідея переходу на багатоядерні процесори з'явилася недавно, десь починаючи з 2005 року, але вона не нова. Ще в далекі 60-ті роки переваги декількох процесорних ядер перед одним обгрунтував Сеймур Крей, потім він реалізував свій задум в суперкомп'ютері CDC 6600. Але в силу консервативності поглядів проектувальників подальшого розвитку цей підхід до проектування центрального процесорного пристрою не отримав.

Відродження багатоядерності відбулося завдяки інженерам корпорації Digital Equipment. У другій половині 90-х задумалися про багатоядерність в мікропроцесорах; це сталося при переході процесора від Alpha 21164 (EV5) до Alpha 21264 (EV6). Тоді дослідникам вдалося встановити дві важливі закономірності, що поширюються на процесори. По-перше, виявилося, що для лінійного росту продуктивності одноядерних процесорів потрібно забезпечити квадратичне зростання числа транзисторів. По-друге, як наслідок, також нелінійно зросте складність проектування. Таким чином, підсумовування продуктивності декількох ядер дасть ту ж сукупну продуктивність, що і одне ядро при меншій кількості транзисторів. Питання в тому, як об'єднати потужності окремих ядер, в цьому полягає корінна проблема багатоядерності. Реакцією на виявлені проблеми став проект Piranha, який передбачав створення 8-ядерного процесора, де кожне ядро мало носити окрему кеш-пам'ять для команд і даних, для спільної роботи процесори об'єднувалися комутатором. Однак після переходу у Compaq, проект Piranha процесор так і не був реалізований[8].

Паралельно з проектом Piranha група дослідників зі Стенфордського університету на чолі з Кунле Олокотуном працювала над процесором Hydra. Як ядро використовувався процесор MIPFS 4600. Робота дала хороші результати.

Згодом Sun Microsystems викупила цю компанію і, замінивши ядро MIPFS 4600 на UltraSPARC II, а також збільшивши число ядер удвічі, випустила експериментальний процесор Niagara, який отримав в серії найменування UltraSPARC T1.

У IBM розробки багатоядерних процесорів помітно пов'язані з виробництвом ігор. Так, процесор Xenon призначений для ігрової консолі. Інший проект Cell, включає ядро SMT Power і вісім ядер, які називають синергетичними процесорними елементами (Synergistic Processing Element, SPE), що працюють за принципом SIMD. Основне ядро Power виконує команди з системи команд PowerPC, підтримуючи спеціалізовану систему команд SPE. З очевидним запізненням свої пропозиції зробили корпорації AMD і Intel: боротьба на багатоядерному полі для виробників процесорів архітектури x86 є одним з найактуальніших питань сьогодення[3,7].

1.3 Специфіка багатопроцесорних систем

При побудові багатопроцесорної архітектури може використовуватися одна з декількох концептуальних моделей з'єднання обчислювальних елементів.

На рис.1 показана загальна структура МП-системи: пов'язана архітектура із загальною пам'яттю з розподіленою обробкою даних і переривань вводу-виводу. Вона повністю симетрична, тобто всі процесори функціонально ідентичні і мають однаковий статус, і кожен процесор може обмінюватися з будь-яким іншим процесором.

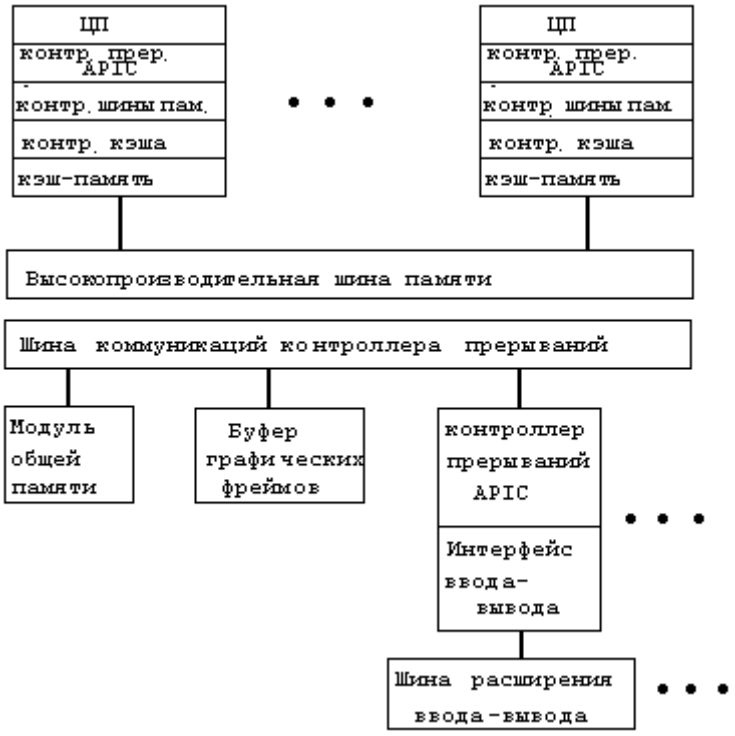


Рисунок 1 - Структура багатопроцесорної системи

Симетричність має два важливі аспекти: симетричність пам'яті та вводу-виводу. Пам'ять симетрична, якщо всі процесори спільно використовують загальний простір пам'яті і мають у цьому просторі доступ з одними і тими ж адресами. Симетричність пам'яті передбачає, що всі процесори можуть виконувати єдину копію ОС. Вимога симетричності вводу-виводу виконується, якщо всі процесори мають можливість доступу до одних і тих же підсистемах вводу-виводу, причому будь-який процесор може отримати переривання від будь-якого джерела. З розвитком кластерних технологій покращуються умови для використання багатопроцесорних комп’ютерів в різноманітних сферах бізнесу, науки та економіки[5-7].

А початок було закладено в науковій сфері, де без подібних складнитх систем не можливо було вирішити складні обчислювальні задачі. З розвитку усіх інших сфер життя, зросла складність поставлених задач і було покладено початок використання комп’ютерних технологій та електронної системи документообігу. Тому все більше проблема побудови подібних систем

набувала актуальності. Багатопроцесорні обчислювальні системи охоплювали все більше сфер, що потребували обробку транзакцій та роботу з базами даних. А також на XXI століття припав активний розвиток сфери телекомунікацій, де аналогічно потребувались високорозвинені технології. Однією з декількох поширених сфер використання подібних систем є системи обробки транзакцій тут і зараз(online transaction processing). З розвитком корпоративних компаній паралельно йшов і розвиток технологій для обробки великої кількості інформації, а також виникла необхідність покращувати такі параметри, як продуктивність, надійність та масштабованість, і зменшувати час затримки задля забезпечення мінімально допустимого часу очікування користувачами. Подібні системи для корпоративних обчислень повинні увесь час перебувати в робочому стані, і короткочасні неполадки у системі приведуть до критичних наслідків й величезних збитків.

Процеси розширення сфер застосування відбуваються паралельно зі зростанням кількості завдань, де необхідно використовувати високорозвинені обчислювальні технології. У світовій практиці було складено список глобальних і прикладних проблем. Щоб вирішити ці проблеми, необхідно використовувати обчислювальні ресурси. Такий список задач, що потребуються вирішення, складається з 14-ти позицій і має назву "Grand challenges"[8]:

- генетика та зміна генетичного коду;
- створення фармацевтичних препаратів та їх клінічні дослідження;
- прогнозування погодніх умов, клімату й майбутніх атмосферних змін;
- наука матеріалів;
- створення напівпровідникових приладів;
- проблематика новітніх розділів біології;
- надпровідність;
- квантова динаміка у майбутньому;
- термоядерний синтез під повним контролем;

- астрономія;
- задачі транспортного значення;
- гідродинаміка;
- дослідження надр Землі;
- підвищення ефективності згоряння палива;
- інформаційні геосистеми;
- наука про світовий океан;
- розробка штучного інтелекту задля розпізнавання зображень і мови.

За допомогою використання багатопроцесорних обчислювальних систем існує можливість підвищення параметрів, таких як надійність системи. Це відбувається за допомогою вчасної заміни окремих вузлів та компонентів, що вийшли з ладу, на справні. Таким чином забезпечується безперервність й відмовостійкість для систем критичного значення.

Паралельні розрахунки є набагато ефективніші при обчисленнях високої продуктивності. Існує велика кількість наукових розрахунків для систем, що побудовані з процесорів невеликої вартості та з підтримкою одночасних обчислень великої кількості задач паралельно.

Зазвичай системи для глобальних обчислень створюються з великої кількості комп’ютерів. Це складний процес, що включає в себе декілька важливих етапів таких як, інсталяція та експлуатація, а також одночасне управління цими комп’ютерами. Також необхідно постійно узгоджувати процес згідно з технічних вимог до паралельних і високорозвинених систем. Також при розробці потрібно слідкувати за міжпроцесорними зв’язками між процесорами чи структурними елементами, і координувати процес в паралельному режимі.

Для вирішення описаних вище проблем та забезпечення усіх параметрів наефективніше використовувати один і той же образ операційної системидля цього кластера. Хоча при реалізації виникають складнощі, тому такий підхід актуальний лише для систем невеликого розміру.

Ці проблеми найпростіше вирішуються при забезпеченні єдиного образу операційної системи для всього кластера. Однак реалізувати подібну схему вдається далеко не завжди, і зазвичай вона застосовується лише для невеликих систем.

Для оцінки ефективності роботи обчислювальних систем для реальних задач, уже існує певний набір тестів. Наприклад програма LINPACK основана на вирішенні системи алгебраїчних рівнянь. Програмне забезпечення є достатньо клаліфікованим, навіть використовується для оцінки параметрів при формування списку 500 найпотужніших комп'ютерів у світі. Хоча описана вище програма має свій недолік, що проявляється в неможливості оцінки роботи комунікаційної складової суперкомп'ютера., адже LINPACK працює паралельно[9].

1.4 Класифікація багатопроцесорних обчислювальних систем

Особливість БОС із загальною шиною (Shared Memory Proccessing), що мають SMP-архітектуру, заключається в тому, що всі скруктурні одиниці системи, а саме процесори, модулі пам'яті, пристрої введення-виведення, з'єднуються з однією загальної шиною. Ширина такої шини може бути від одного до декількох байтів. Архітектура SMP на сьогодні є стандартом для ряду сучасних багатопроцесорних систем (наприклад DEC Alpha Server AXP).

Зв'язок між парою модулів системи виконується в реальному режимі по шині, при чому за один прохід передається лише один пакет даних від одного джерела інформації. І поки не закінчиться один процес передачі даних, необхідно перечекати, щоб можна було розпочати передачу наступного пакета.

До переваг багатопроцесорних систем із загальною шиною відноситься:

- порівняно невисока вартість, та функціонально нескладна організація системи;
- реконфігурація системи за допомогою заміни або ж видалення модулів за досить невелий час.

До недоліків описаних вище систем:

- можливе зменшення параметрів системи при додаванні нових модулів, зокрема падіння продуктивності;
- обмеження системи згідно з числом пропускої здатності загальної шини;
- при зупинці роботи шини система повністю виходить з ладу;
- витрати продуктивності та часу на конфліктні ситуації, що виникають у разі, коли на одну шину претендують більше одного модуль[6].

Для того, щоб зменшити число збитків, використовується окрім загальної пам’яті, ще й місцева пам’ять.

Подібна організація з використанням загальної шини є досить популярною при побудові як багатопроцесорних систем, так і багатомашинних обчислювальних комплексів.

Шинна архітектура, що була раніше закладена раніше, процесор і кеш розміщувалися на одній платі, що була встановлена на задню панель. На сьогодні апаратні можливості дозволяють вмістити чотири процесори на одній платі.

У подібних системах і колективні, і приватні дані можуть зберігатися в кеші. Приватними даними називаються такі дані, що використовуються лише одним процесором, а колективними – ті, що можуть бути у загальному користування декількома елементарними одиницями. Процес кешування елементів приватних даних є аналогічним до однопроцесорної машини з кеш-пам’яттю.

Серед переваг можна виділити скорочення часу затримку, смуги пропускання, що забезпечує також загальне скорочення кількості обмінів. Хоча є і певний ряд недоліків, до яких відноситься і когерентність кеш-пам’яті.

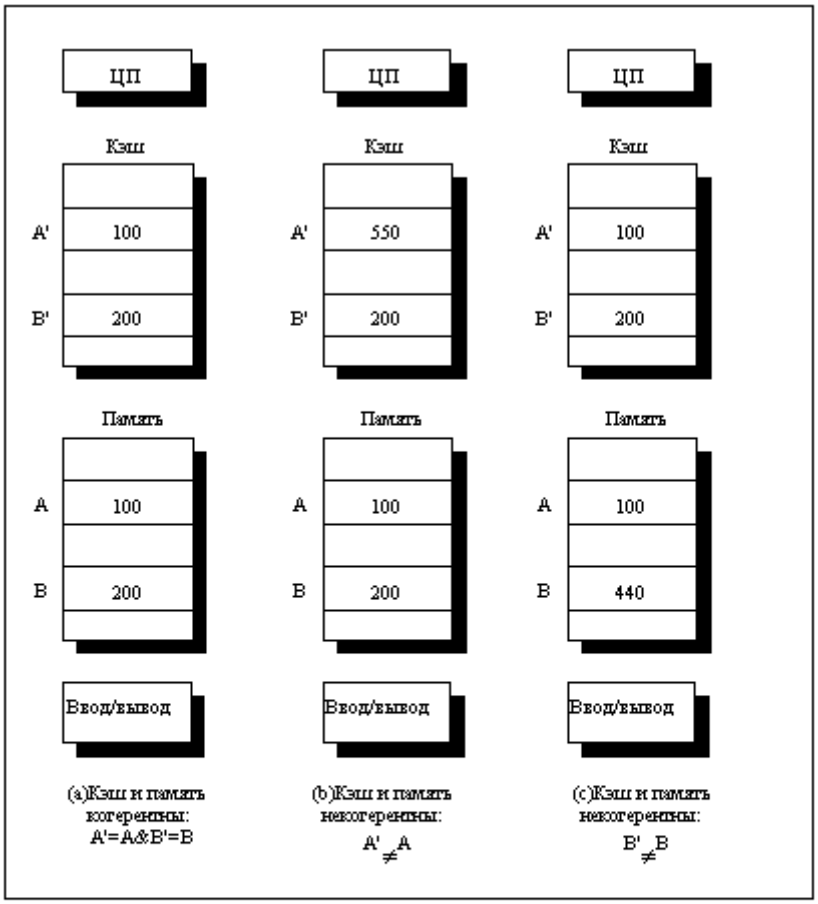
Ця проблема виникає через те, що значення структурні одиниці зберігається в двох різних процесорах. Існує чимало аспектів для проблеми когерентності. Апаратним механізмом називається такий протокол, що забезпечує вирішення даної проблеми. Подібна схема використовується зазвичай в невеликих мультипроцесорних системах, і називається протоколом когерентності кеш-пам’яті.

Описані вище протоколи поділяються на два класи[5]:

1. Протоколи, що основані на довідниках (directory based). Інформація про фізичну пам’ять міститься лише в одному довіднику, хоча апаратно довідник можливо розкиданий по різних вузлах БОС.
2. Протоколи спостереження (snooping). Кожному кешу, що вміщує копію даних блоку фізичної пам’яті, ставиться у відповідність копія інформації про стан. У даних протоколів немає централізованої системи записів. Загалом у даному випадку кеш розміщують на загальній шині так, що контролери кешів переглядають кожного разу шину задля визначення копій певного блоку.

У багатопроцесорних системах, що використовують мікропроцесори з кеш-пам'яттю, під'єднані до централізованої загальної пам'яті, протоколи спостереження набули популярності, оскільки для опитування стану кешів вони можуть використовувати заздалегідь існуюче фізичне з'єднання - шину пам'яті.

Протоколи спостереження в останній час стали популярними, особливо в багатопроцесорних обчислювальних системах, що складаються з мікропроцесорів з кеш-пам’яттю. Вони стали поширеними за рахунок того, що перевірка стану кешів виконується без створення зайвих структур, адже використовується шина пам’яті.



A' и B' - кэшированные копии элементов A и B в основной памяти

a) Когерентное состояние кэша и основной памяти.

b) Предполагается использование кэш-памяти с отложенным обратным копированием, когда ЦП записывает значение 550 в ячейку A. В результате A' содержит новое значение, а в основной памяти осталось старое значение 100. При попытке вывода A из памяти будет получено старое значение.

c) Подсистема ввода/вывода вводит в ячейку памяти B новое значение 440, а в кэш-памяти осталось старое значение B.

Рисунок 2 - Ілюстрація проблеми когерентності кеш-пам'яті

На рисунку 2 зображено проблему когерентності. Необхідно за кожного проходу перевірки елемента даних повертати записане значення за останній проміжок часу. В цьому і заключається основна проблематика когерентності. Однак існує певна проблема затримки, адже миттєвості даного процесу зчитування заважає той факт, що зчитування інформації відбувається одним процесором, а запис іншим. А також виникає колапс при таких ситуаціях, коли до однієї частини пам’яті намагаються добратися декілька процесорів для протилежних цілей таких, як зчитування та запис,то немає ніяких гарантій, що після процесу зчитування повернуться коректні

дані, а не уже змінені. Для того, що дана проблема була вирішена, використовується певний тип пам’яті, а також відбувається реалізація передачі й організація розпаралелювання одночасних процесів. Також має сенс записувати момент перезаписування даних, щоб мати можливість правильно налагодити синхронізацію обчислень у системі[10].

Для того, щоб вирішити проблему, що описується вище, необхідно забезпечити організація двох операцій, та їх правильну взаємодію.

Щоб виконати операцію читання певної області БОС, необхідно переконатись, що в даний момент відповідна комірка не використовувалася іншим процесом для запису інформації, інакше коректність отриманих даних піддається сумніву.

Дана властивість пов’язана з вчасним визначенням стану процесора, адже якби операція постійно повертала лише б попереднє значення, то це б доказало, що когерентність відсутня.

Щоб забезпечити операцію запису для однієї комірки кількома послідовними командами, необхідно виконувати цей процес таким чином, що інші процесори очікують поки закінчяться послідовні операції запису.

Забезпечити строгу послідовність операцій запису є більш складною задачею. Якщо припустити, що організація операцій запису відбувається не послідовно, то в такому випадку процесори P1 і P2 виконають запис один за одним, і в кінцевому варіанті прослідкувати дані, що записані першим з цих процесорів P1, буде неможливою задачею, і для читання будуть доступні лише дані P2. Якщо ж записування буде відбуватися не послідовно, то виникне ситуація, в якій P1 намагатиметься достукатися до даних, над якими уже працює процесор P2. Тому запис даних першим процесором буде виконуватися досить з великою затримкою, адже час буде витрачений на очікування. Якщо при некоректній організації запису двох процесорів, третій процесор буде намагатися зчитати дані з тієї ж комірки, то замість очікуваних результатів послідовного зчитування записаних значень процесорами P1 і P2, невідомо, чи не буде втрачена частина даних. Якщо ж

навіть ми отримаємо інформацію, що було записано другим процесором, то дані, що записані першим процесором, уже будуть втрачені. Щоб забезпечити правильність виконання цих операцій, використовується властивість, що називається серіалізація – це послідовне зчитування інформації процесором.

Існує декілька методів підтримування властивості когерентності. Один з них заключається в тому, що перед початком операції записування інформації необхідно мати певні права на виконання. Оскільки під час запису безпосередньо протокол перекриває собою всі інші копії, то такий тип називається write ivalidate protocol. Цей тип є найпопулярнішим і часто використовується в усіх існуючих типів схем. Право на запис, що надається при вище описаному протоколі, забезпечує відсутніх інших копій комірки пам’яті, в яку відбувається запис, тому це надає гарантію того, що в комірці буде записано те, що потрібно. Для того, щоб наглядно побачити забезпечення властивості когерентності, опишемо таку ситуацію, коли відразу після операції запису іде операція зчитування другим процесором. Для початку записування даних в обрану область пам’яті необхідно отримати права доступу. І поки в процесора є ці права, операція читання буде намагатися зчитати дані безуспішно. Як тільки буде закінчено запис, буде можливо початок читання даних. Якщо ж припустити, що ще один процесор буде намагатися записати дані в комірку, то згідно даного протоколу процесор перед записом повинен створити копію для роботи з нею. І той з процесорів, що має права доступу анулює копію іншого і тільки після цього розпочинається запис. Інший процесор почне роботу уже з оновленою копією даних після того, як відбувся запис першим процесором.

Наступний тип протоколу працює по принципу оновлення всіх копій лише у випадку запису в цю комірку пам’яті. У нього є дві загальновживані назви: протокол з оновленням та з трансляцією (write update/broadcast protocol). Для того, щоб зменшити кількість вимог до смуги пропускання, виконується перевірка наявності частини слова в пам’яті. Якщо ж пошук не

знаходить жодного співпадіння, то оновлення кеш-пам’яті робити не потрібно[3,9].

Виконаємо порівняння двох протоколів:

1. При серії послідовного запису декількома процесорами необхідно враховувати мінімально необхідну кількість операцій. Використовуючи протокол запису з анулюванням необхідно лише одна операція анулювання, а от при протоколі запису з оновленням потрібно виконати невідомо скільки операцій для перевірки.
2. Протокол запису з анулюванням виконує операцію читання на більш глобальному рівні, за раз зчитуючи блок даних, що безумовно економить час виконання. А протокол запису з оновленням зчитує інформацію з кеш-пам’яті по слову, що вимагає більшої кількості операцій й затримує роботу багатопроцесорної системи, адже необхідно виконати декілька дій трансляції.
3. Схема запису з оновленням працює таким чином, що одразу після запису даних в комірку, відбувається трансляція записаної інформації і можливо прочитати нові дані з кеш-пам’яті. Для того, щоб після запису інформації в комірку при іншому типу протоколу, прочитати записані дані необхідно перечекати поки анулюються всі створені копії і буде надано доступ до оновленої області пам’яті.

Робота описаних вище схем дуже схожа по принципам на схеми для опрацювання кеш-пам’яті з записом через зворотнє копіювання та з наскрізним записом. Схема з наскрізним записом працює аналогічно до протоколу з оновленням, адже дані миттєво підлягають оновленню після закінчення процесу записування. Протокол з анулюванням зазвичай потребує невеликого за розміром трафіку, так само як і схема з записом зворотнього копіювання. Оскільки при використанні протоколу з анулюванням обирається блок, то основна відмінність між цими двома схемами залежить майже повністю від розміру блоку, що буде оброблюватися.

Існують багатопроцесорні системи з багатовходовою пам’яттю, і в цьому випадку до кожної шини підключається свій пристрій введення-виведення інформації. Число входів до кожного модуля обмежує кількість модулів, що можуть бути підключені.

Для того, щоб оперативно організувати роботу з подібними БОС актуальним виходом є реалізація розшарування оперативної пам’яті і таким чином близькі адреси знайдуть своє місцезнаходження в сусідніх модулях.

До переваг багатопроцесорних багатошинних систем відносяться:

- можливість роботи і в однопроцесорному режимі;
- збільшення швидкодії в передачі пакетів даних.

Серед недоліків можна виділити наступне:

- необхідна оперативна пам’ять є дорогою при купівлі;
- збільшена кількість різноманітних ліній зв’язку між елементами системи ускладнює апаратну побудову та зменшує надійність.

Аналогічні принципи про конструюванні використовуються і в мейнфреймах. Розглянемо два способи побудови глобальних систем з розподіленою пам’яттю. Спершу опишемо процес, що заключається в ігноруванні такої властивості системи, як когерентність за рахунок виключення апаратних процесів, що цьому сприяють. При цьому основна увага приділяється забезпеченню масштабованості пам’яті системи. По вище описаним принципам було розроблено комп’ютер T3D, що належить фірмі CRAY Research. Між елементами пам’яті даної системи рівномірно розподіляється пам’ять, а зв’язки забезпечуються за допомогою будь- якого можливого типу мережі. Перевагою даних комп’ютерів та систем є те, що доступ до пам’яті може надаватися і локально, і віддалено. Комп’ютер сам обирає найбільш вдалий тип доступу за допомогою спеціально налаштованих контролерів, які можуть визначити на локальному пристрої розташовані дані чи ні. Якщо ж дані розташовані в іншому модулі пам’яті, то спеціальні контролери віддаленої пам’яті формується повідомлення для того, щоб добратися до необхідних даних.

Є ще один варіант для вирішення проблеми когерентності, а саме кешування даних. Таку схему кешування можливо реалізувати за допомогою програмного забезпечення, яке і буде управляти когерентністю системи. Копіювання групою інформації значно б спростило даний метод, хоча якщо не брати до уваги цей варіант, то даний спосіб реалізації може здійснюватися повність програмно, не беручи до уваги апаратні можливості. До недоліків подібної схеми роботи відноситься обмеженість програмних засобів для підтримки. Тому дана організація підійде в тих випадках, коли програма має чудово розвинену паралельну структуру ще при програмній реалізації.

Архітектура машин типу Cray T3D називається масовою та паралельною. Вимоги, що представляються до машин вище описаного типу, мають зв'язок між собою. До прикладу якщо збільшити обсяг комп'ютера, то в нього можна буде помістити більшу кількість процесорів, і відповідно збільшиться довжина каналів передачі інформації. Значення довжини каналів обернено протилежне до тактової частоти. Відповідно до цього при збільшення їх довжини, тактова частота зменшується пропорційно.

Зростання розміру машин до 64К сталося як наслідок підвищення рівня інтеграційних схем, та дозволило збільшити щільність, тобто на пристрої більше розміщується елементів. При одних і тих же даних може бути різна топологія мережі при передачі даних.

Також при розробці апаратного забезпечення пристроїв існує ще один варіант побудувати систему базуючись на довідникові, що містить у собі інформацію про всі стани одиниць пам'яті, що готові до процесу кешування. Довідник вміщує в собі велику кількість інформації зокрема про те, що підлягав змінам певний блок даних. В залежності від виду протоколу в деяких з них частина загальної інформації вміщується в кеш-пам'ять. Апаратно при розробці пам'ять і довідник ставляться поряд на схемі.

Така структура є перевагою, що забезпечує простоту, а також зберігання усією необхідної інформації в одному місці. До мінусів відноситься великий розмір, що по величині відповідає самій пам'яті. Хоча

цей недолік виникає лише при системах з великою кількістю процесорів. А якщо це число не переважає порядка сотень, то апаратні можливості дозволять системі зі словником справно виконувати передачу інформації. Тобто це є проблемою лише у випадках, коли кількісна характеристика елементів системи перевищує число в кілька сотень одиниць.

У випадку, коли система досить велика і це пов’язано з використанням одного довідника, то ця проблема піддається вирішенню за допомогою розділення довідника на структурні частини. І після цього операції з частина довідника можливо виконувати паралельно. Така методика значно підвищує число смуги пропускання[9].

На рисунку 3 зображено варіант розподілення пам’яті для таких машин.

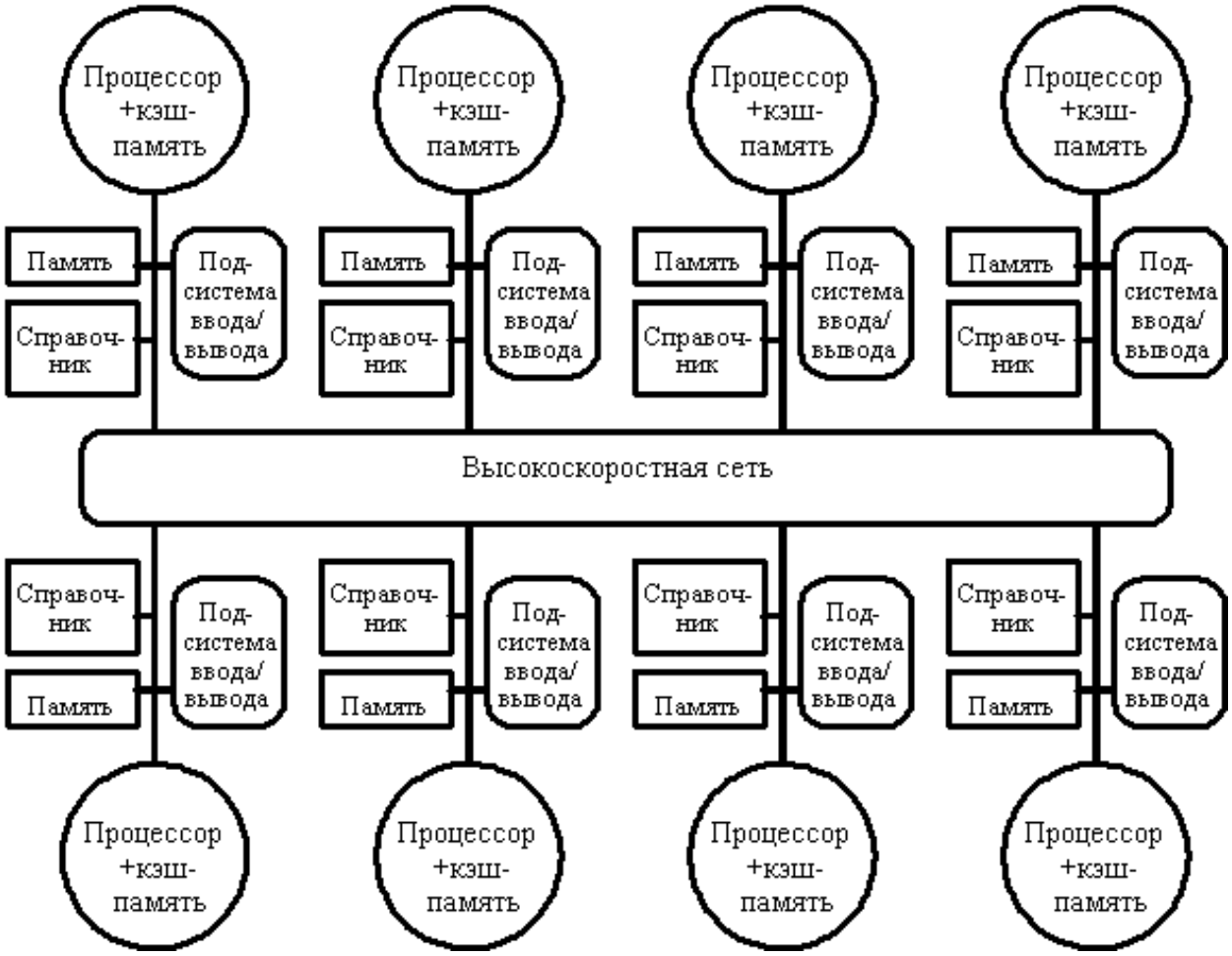


Рисунок 3 - Архітектура системи з розподіленою зовнішньою пам'яттю

1.5 Спосіб організації тестування на основі Препарату-Метца-Чена

Для того, щоб забезпечити правильну роботу відмовостійких систем необхідно вчасно виявити несправні процесори. Для цього проводиться взаємотестування елементів системи. Розглянемо методику, згідно з якою проводиться організація діагностування системи. Задача вчасного розпізнавання несправностей є на сьогодні досить актуальною, як і зменшення часових витрат на процеси діагностування. Відмовостійку систему прийняти позначати k-out-of-n. Найбільшу практичну цінність для діагностування графів на сьогодні несе побудова Препарату-Метца-Чена (ПМЧ-модель). Нехай загальна кількість процесорів дорівнює n, а мінімальне число допустимих несправностей системи m не перевищує чотирьох.

Для побудови моделі багатопроцесорної системи використовується циркулянтний граф, що має суттєві переваги у порівнянні з іншими можливими структурами. Одним із плюсів такого графа є можливість створення на його основі системи з великим числом процесорів, що неможливо для гіперкуба.

На першому етапі відбувається формування графа-циркулянта. Вершини служать процесорами, а ребра – зв’язками між відповідними елементами. Значення ребра приймають в залежності від результату тестування: 1, якщо справний процесор тестує несправний, 0 – якщо справний тестує несправний і x при тестування несправним процесором будь-якого іншого. Алгоритм побудови графа-циркулянта вказано на рисунку 4.

Нехай існує багатопроцесорна система з діагностичним графом G(m,n), при чому допустиме число відмов менше половини загального числа процесорів. Для того, щоб з’ясувати стани усіх елементів достатньо n+2m перевірок. Припустимо, що число несправних процесорів дорівнює m,- у такому випадку виходить ситуація, коли всі несправні процесори тестують

однаковий процесор. І хоча виникає невизначеність, варто звернути увагу, що такий процесор буде один, і всі ребра, що входять в даний процесор, є несправними. А процесор, що залишився, є справним. У випадках, коли число відмов буде менше, то завжди мінімум один процесор є справним[11].

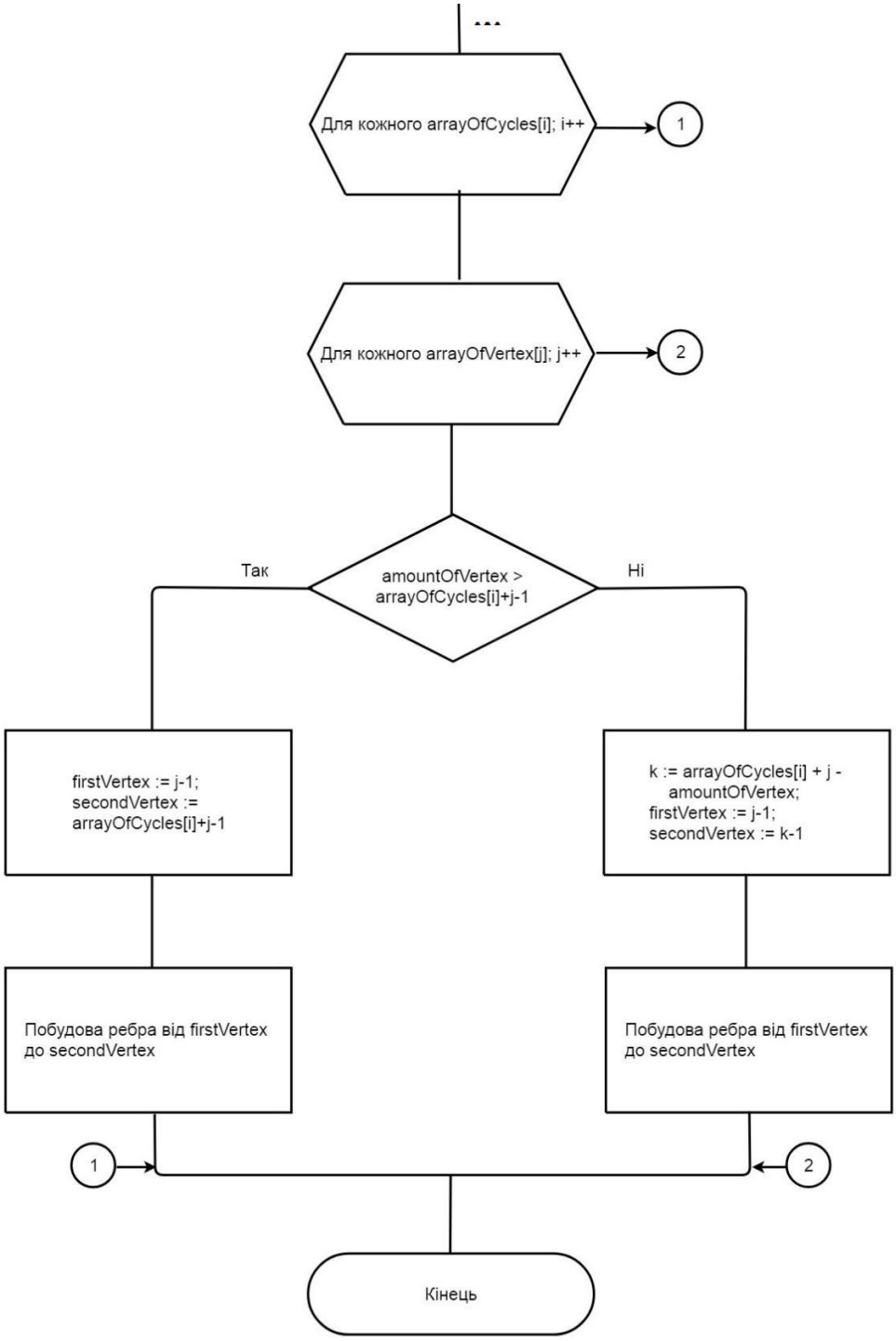


Рисунок 4 – Алгоритм побудови діагностичного графа-циркулянта

На другому етапі відбувається формування 0-ланцюжків. Випадковим чином обирається пара процесорів із зв’язком. Якщо при перевірці першим процесором другого результат 0, то виконується подальше тестування третього процесора, що раніше не був задіяний.

Формування ваги для кожного ребра відбувається так, як це показано на рисунку 5.

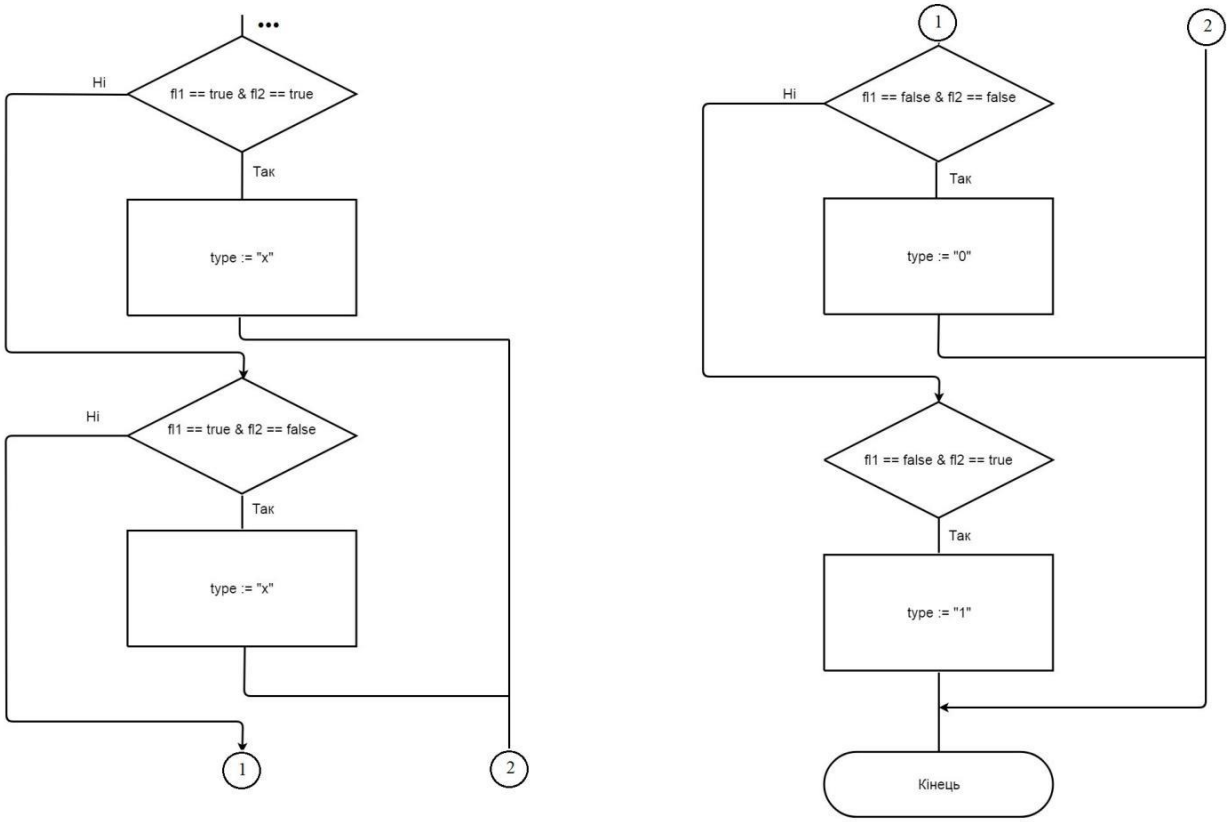


Рисунок 5 – Формування ваги для ребра графа

Якщо ж результатом перевірки буде число 1, то відповідна пара видаляється, а взаємне тестування продовжується. Останній процесор ланцюжка може тестувати будь-який процесор, що ще не був включений. Таким чином відбувається пошук усіх ланцюжків. Процес пошуку 0-ланцюжків можна спостерігати на рисунку 6.

Третій етап полягає в тому, що усі знайдені ланцюжки об’єднуються. Процес додавання розпочинається з найдовшого ланцюжка. Обов’язковою

умовою є те, що тестування останнім процесором найдовшого ланцюга проводиться над процесором наступного ланцюга при однакових вагах цих вершин. В результаті утворюється єдиний ланцюжок, зменшений на декілька елементів.

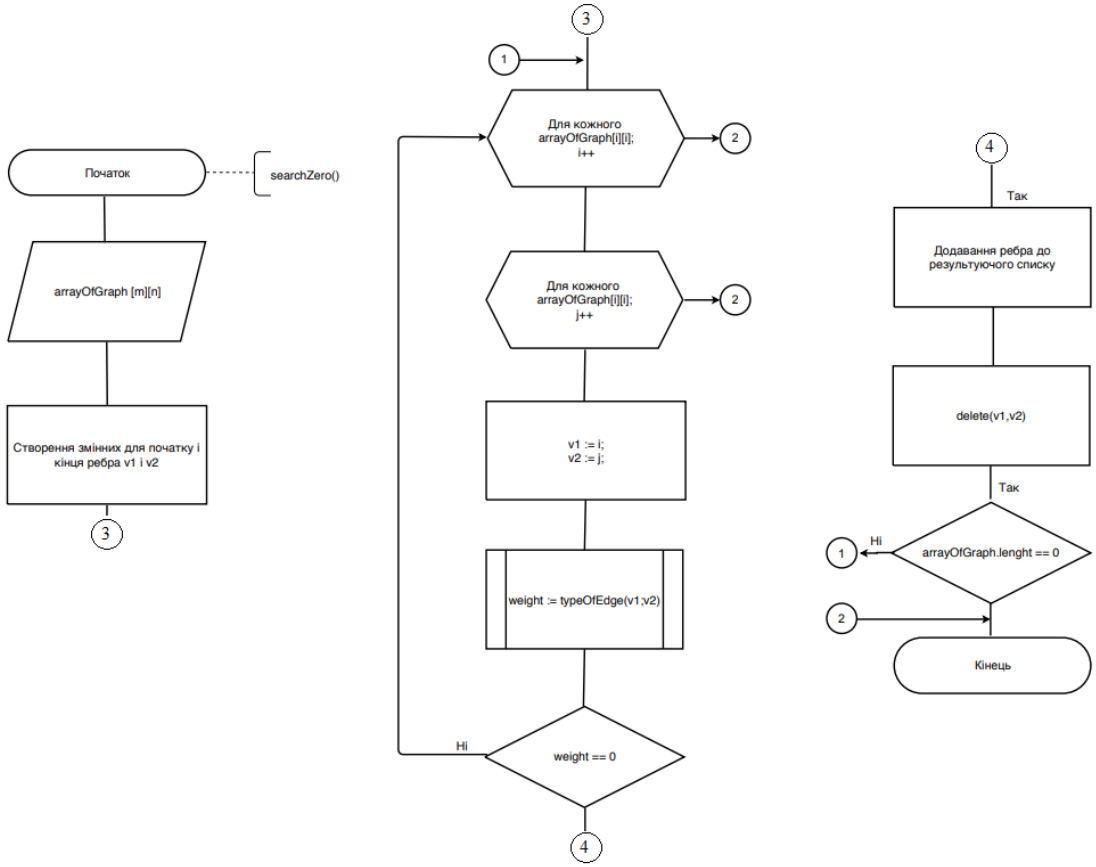


Рисунок 6 – Формування 0-ланцюжків

На рисунку 7 показано приклад того, як відбувається об’єднання двох ланцюжків.

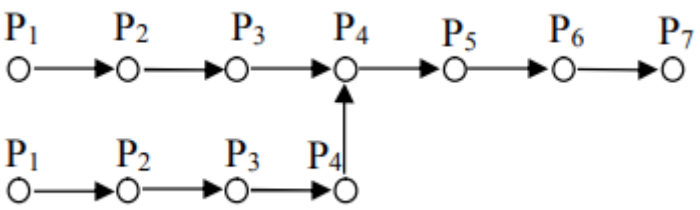


Рисунок 7 – Об’єднання ланцюжків в одне ціле

Так і відбувається об’єднання усіх ланцюгів попарно. Аналогічно відбувається і для дерев. Обираються вершини з двох дерез з однаковою вагою. Після цього необхідно сформулювати твердження для того, щоб знайти справний процесор. При умові, що $R+\alpha > m$, останній процесор з вагою R буде справним, де α – це несправні процесори, що залишилися без ланцюжка.

Розглядається крайній випадок, коли серед процесорів, що залишились поза деревом, буде більше половини несправних процесорів. При таких умовах є можливість визначити справний процесор[12].

На четвертому етапі відбувається аналіз отриманих даних. На минулому етапі було визначено справний процесор, тому тепер відбувається безпосередньо сам процес тестування.

Справний процесор починає перевірку з першого елемента ланцюжка, або ж обирає процесор з множини, що ще не підлягала тестуванню. Якщо всі процесори, що були протестовані в зоні досяжності одним справним процесором, виявилися несправними, то в такому разі необхідність подальшого тестування відпадає і всі інші процесори є справними.

Якщо ж було визначено справні процесори, то вони повторюють дії першого процесора почергово встановлюючи стани кожного процесора. Якщо буде знайдено усі несправні процесори, то автоматично інші процесори є справними. Це випливає з того, що кожному несправному процесору ставиться у відповідність мінімально один справний. Таким чином закінчується процес тестування

Необхідно визначити кількість перевірок, що потрібно виконати для того, щоб встановити стан усіх процесорів системи. Якщо розглянути кількість операції, що проводилася, починаючи з формування першого ланцюжка, то можна вирахувати число необхідних перевірок $S = n + 2m$. Цифра два береться для перестраховки, адже на четвертому етапі неможливо точно визначити, скільки перевірок проводиться. Насправді в більшості випадків це число є значно меншим[11].

Розглянемо алгоритм пошуку 0-ланцюжка більш детально.

Припускається, що максимальне число несправних модулів T в обчислювальній системі, що самодіагностується (ОСС) не перевищує (N-1)/2 (N-число модулів ОСС); число t реально справних модулів в ОСС невідомо ($t \leq T$).

Нехай існує паралельно t-діагностична ОСС, що складається із N модулів vi, i= 1, …, N, що задається діагностичним графом G = (V, E), V - множина вершин, що відповідають модулям ОСС, E - множина комутуючих зв'язків wij ($i \neq j$, i,j = 1, 2 ,…, N), що забезпечують проведення діагностичного експерименту. При цьому, під елементарною тестовою перевіркою розуміється процес тестування модуля vj модулем vi, результат якого rij описаний ПМЧ-моделлю. В подальшому таку перевірку будемо позначати (vi,vj).

Період часу, протягом якого виконується така перевірка є тактом діагностичного експеримента. Для ПМЧ-моделі справедливі наступні твердження:

rnn = 0, rnf = 1, rff = x, rfn = x;

Де n - справний модуль, f - несправний модуль, x - невідомий результат (може бути 0 або 1). Припускається, що протягом одного такта кожний модуль може бути або тестуючий або тестований, при цьому тестуючий модуль може одночасно перевіряти будь-яке число модулів (ми розгдядаємо ОСС з комутуючими діагностичними графом), тоді як тестований модуль перевіряється тільки одним модулем .

Для метода ПМЧ допустиме число несправних процесорів $T =](n-1)/2[$. Візьмемо за основу ту ж нижню границю для n, n > 2T , не дивлячись на те, що графи, що розглядаються, мають менше діагностичних зв'язків, ніж в методі ПМЧ. Відмітимо, що із властивостей ПМЧ-моделі слідують наступні положення:

Якщо на діагностичному графі має місце дуга з результатом vij = 1, то можна з впевненістю стверджувати, що хоча б один з процесорів i, j несправний.

Якщо на діагностичному графі має місце 0 - ланцюг довжиною m+1 модулів, а в системі допускається не більш ніж m відмов, то останній модуль справний [3].

Складність пошуку “0” - ланцюжків заключається в тому, що їх кількість наперед невідома. Якщо виконувати пошук простим перебором пар вершин, то для великої кількості вершин алгоритм буде програвати у швидкодії.

В якості вхідних даних алгоритму виступає діагностичний граф, вершинами якого є процеси з нумерацією від 1 до N, де N - це кількість вершин, а ребра відповідають результату взаємотестування процесів за ПМЧ-моделлю.

Оскільки дані про граф знаходяться в двовимірному масиві, то перед початком роботи необхідно впорядкувати його. Враховуючи властивості діагностичного графа, спершу виконується видалення ребер з вагою “x” та “1”. Потім двома циклами for виконується послідовна попарна перевірка. В результаті формується одновимірний масив, заповнений вершинами. Масивів може бути декілька. Таким чином, за допомогою алгоритму, в подальшому спрощується пошук несправних процесорів.

Висновки до розділу 1

У сучасному світі багатопроцесорні системи розвиваються стрімко, але зміна апаратної архітектури тягне за собою зміну програмного забезпечення. У першому розділі було розкрито поняття "багатопроцесорні системи", а також розглянуто рішення проблем, пов'язаних з їх тестуванням.

Аналізується можливість повного діагностування багатопроцесорних систем при використанні умов і моделі методу Препарату Метца-Чена. Відомо, що воно завжди можливо, якщо число несправних процесорів системи менше половини їх загальної кількості. На основі запропонованого аналізу діагностичного графа системи показано, що повне діагностування можливо і при більшому допустимому кількості відмовили процесорів (але не більше $n-2$) і визначені умови, коли діагностування дійсно неможливо. Якщо повне діагностування БС неможливо, то аналіз діагностичного графа системи це показує і дає можливість виявити, стан яких процесорів залишається невизначеним.

2 ХАРАКТЕРИСТИКА GL- МОДЕЛЕЙ ТА ОСНОВНІ ТВЕРДЖЕННЯ

2.1 Особливості GL-моделей K(2,N)

За останні роки значно зросла складність сучасних обчислювальних систем, що управляють надзвичайно складними об’єктами критичного значення. Окрім забезпечення можливості тестування подібних систем виникає завдання підвищення рівня надійності даної системи. На сьогодні досить часто розробники вводять певну надлишковість по апаратурним параметрам збільшенням кількості одиниць процесорів, що працює ніби страховка при виході з ладу певної кількості процесорів. Хоча точно оцінити надійність і досі залишається нелегкою задачею. Для такої оцінки необхідно задіяти великі об’єми обчислювальної техніки, щоб оцінити всі можливі варіанти роботи системи при різних наборах вхідних параметрів. І навіть ті моделі, що уже існують на сьогодні не можуть забезпечити необхідну точність обчислення такого важливого параметра, як надійність. Не всі сучасні розвинені багатопроцесорні системи можуть бути досліджені існуючими моделями, адже, як правило, вони вміщують велику кількість процесорів, а також різноманітну топологію зв’язків між модулями або шинами. Саме у подібних складних випадках найдоцільніше використовувати GL (графо-логічні) –моделі. Їх перевагою є можливість аналізувати будь-яку багатопроцесорну систему, в тому ж числі й багатошинну. (11-19)

GL-модель відмовостійкої багатопроцесорної системи (ВБС) представляється неорієнтовним графом *G*. Кожне ребро графа – це булева функція, аргументи якої є змінні x_i $(i = 1, ..., n)$. Змінна дорівнює або нулю, або ж одиниці в залежності від несправності чи справності елемента відповідно. Якщо ж ребро графа приймає значення 0, то відповідне ребро видаляється з графа. Якщо після такого ряду виключень граф залишається зв’яззний, то система роботоздатна. Базовою ВБС називається така система, до складу якої входять n елементів, і роботоздатність зберігається у випадку

m відмов будь-яких процесорів. Така система буде *K(m, n)* при *(0 < m < n)*. Хоча на практиці багатопроцесорних систем, що представляються виключно базовою практично не існує. Тому виникає маса задач, вирішення яких проводиться шляхом перетворення базової моделі. Для того, щоб забезпечити найкраще побудування складніших моделей з використанням базових, необхідно якнайкраще дослідити просту GL-модель *K(m, n)*.

В даному розділі реберні функції будуть мати функції такого вигляду, як описано в формулі 1.

$$h_i = x_i \vee \prod_{j=1,[\frac{n-1}{2}]} x_{g_{(i+j) \bmod(n)}} \quad (1)$$

В даному випадку *n* – це число вершин графа *G*, а $i = 1,\ldots,n$.

Даний граф може втратити зв'язність лише, якщо мінімум троє змінних з множини $\{x_i, \ldots, x_n\}$ набудуть нульового значення. Тобто дана GL-модель описує поведінку базової ВБС при $m = 2$.

На рисунку 8 зображено приклад системи *K(2,8)*, та відповідна їй GL-модель.

$$h_1 = x_1 \vee x_2 x_3 x_4$$
$$h_2 = x_2 \vee x_2 x_4 x_5$$
$$h_3 = x_3 \vee x_4 x_5 x_6$$
$$h_4 = x_4 \vee x_5 x_6 x_7$$
$$h_5 = x_5 \vee x_6 x_7 x_8$$
$$h_6 = x_6 \vee x_7 x_8 x_1$$
$$h_7 = x_7 \vee x_8 x_1 x_2$$
$$h_8 = x_8 \vee x_1 x_2 x_3$$

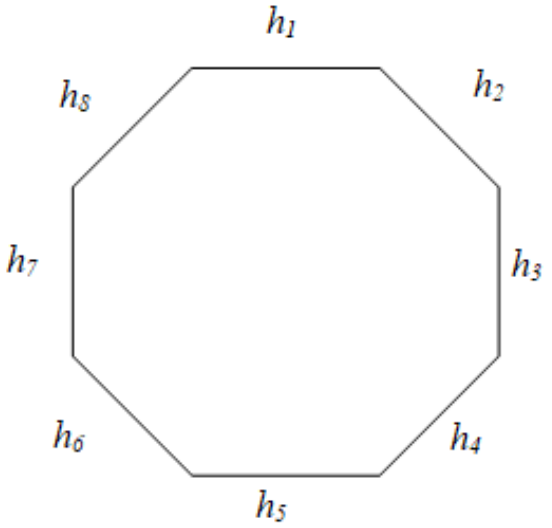


Рисунок 8 - GL-модель для системи K(2,8)

До плюсів даних систем можна віднести доволі просте формування функцій на ребрах. Адже щоб отримати h_{i+j} необхідно добавити *j* по *mod (n)* до кожного індекса змінних функції h_i.

Послідовність номерів процесорів системи *{g1, g2, g3, ..., gn}* і визначає вигляд реберних функцій систем типу *K(2, n).* За допомогою циклічного зсуву вправо можна отримати множину реберних функцій. Базовим буде той вектор, перша цифра якого буду дорівнювати одиниці, тобто (1,2,3,4,5,6,7,8). Відповідно до множини можна побудувати орієнтовний циклічний граф. Вершинами у нього будуть номера елементів системи, а зв'язки відповідають відношенню один до одного відповідних елементів (рис. 9).

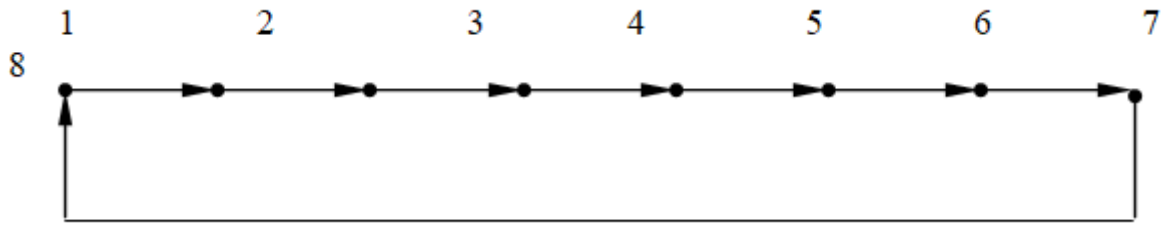


Рисунок 9 – Орієнтовний циклічний граф

Впорядкована послідовність нулів й одиниць називається вектором стану даної системи, при чому якщо елемент дорівнює 1, то вважається справним, а якщо рівний 0, то відповідно несправний. Позначемо відстань між *i*-м та *j*-м елементами довжиною L_{ij}. Ті вектори стану системи, що містять три 0 відносяться до класу *S*.

На практиці часто виникають ситуації, коли системи не є роботоздатними при виникненні p відмов, але при q залишаються здатними виконувати всі функції справно. Тому необхідно показати той факт, що при різних перестановках наборів відмов одна й та ж система може по-різному працювати.

Для того, щоб продемонструвати цю властивість системи можна провести внутрішні ребра з власними реберними функціями. Якщо додати занадто велику кількість внутрішніх ребер, то існує ймовірність втратити

зв’язність системи, тому важливою задачею є додавання якомого меншої кількісті внутрішніх реберних функцій.

При відмові трьох елементів системи *K(2,8)*, виникає необхідність видалення трьох ребер. А для збереження зв’язності даної системи необхідно провести два додаткових ребра – це мінімальне число, що необхідне для забезпечення роботоздатності системи.

Запишемо формулу (1) у вигляді:

$$h_i = x_i \vee \Pi(R_i), \quad (2)$$

де R_i – це множина змінних, що розташовані правіше знаку кон’юнкції у формулі для реберних функцій.

Отже, для того, щоб при наявності вектору стану класу S, реберна функція типу (2) набула значення 0, необхідно і достатньо забезпечення дфох умов:

- $x_i = 0$;
- наявність хоча б однієї змінної, що відповідала би нульовому компоненту, у множині R_i.

У випадку, коли відстань у базовому векторі між *i*-м та *j*-м елементами системи не перевищує величину $[\frac{n-1}{2}]$, видаляється ребро з даної GL-моделі.

В такому випадку $x_j \in R_i$.

Якщо з’являється вектор стану системи *K(2, n)* з класу S, то необхідно видалити з графа 2 ребра у випадку, коли одна з відстаней базового вектору стає більша за величину $[\frac{n-1}{2}]$.

2.2. Спосіб побудови GL-моделей та їх перетворення

Для існуючого сьогодні в світі рівня промислового і технічного розвитку характерно широке застосування складних багатопроцесорних

систем, які налічують десятки і сотні модулів. Багатопроцесорні системи використовуються в багатьох галузях науки і техніки, в тому числі при управлінні об'єктів авіації, станцій. Подібне застосування виставляє жорсткі вимоги до забезпечення високого рівня надійності систем їх управління.

Для відображення систем та їх поведінки у потоці відмов використовується велика кількість різних видів моделей, але найважливішу позицію на сьогодні займають саме графо-логічні (GL). Вони поєднують теорію графів та булевих функцій. Таким чином допомагають показати поведінку системи у потоці відмов. Такі системи використовують при обчисленні надійності, як одного з найважливіших параметрів системи. Цей процес відбувається за рахунок найпростіших обчислень та за допомогою ряду статистичних експериментів системи.

Схема розрахунку надійності системи без використання інших засобів для підвищення надійності виконується наступним чином: у відповідність ставиться двополюсний граф, в якому кожне ребро відповідає одній одиниці, а саме модулю даної системи, а вершини мають послідовне з’єднання між собою. Такий граф за допомогою простих перетворень виходить елементарний ланцюг, за допомогою якого виникає можливість легко записати певну логічну структуру і по ній зробити розрахунки надійності.

В описаному вище випадку для кожного окремого ребра приписується індикаторна булева змінна x_i, де $i = 1,...,n$. Тут і далі n – кількість структурних одиниць, а саме процесорів у системі, що визначає наявність чи повну відсутність ребра. При несправності хоча б одного модуля системи втрачається зв’язність і видаляється дане ребро[20].

Для експерименту побудуємо модель, що показує поведінку 0-відмовостійкої системи $K(0,n)$. Якщо об’єднати два полюси графа однією вершиною, то отриманий граф буде мати вигляд елементарного циклу і буде очевидним, що він показує поведінку 1-відмовостійкої системи. Тобто така система готова витримати 2 відмову процесора чи модуля.

Побудуємо модель для m-відмовостійкої системи *K(m,n).* Для початку потрібно зробити заміну на ребрах булевими функціями, при чому необхідно не забувати врахувати важливу деталь, що при менше ніж m відмов системи всі функцію дорівнюють 1, і лише при числі відмов рівній m, з'являється ребро зі значенням 0. В подібному випадку ланцюг сформує поведінку для *(m-1)* –відмовостійкої системи. При умові коли відмов стане ще більше, то однозначно можна сказати, що як мінімум два ребра матимуть нульові значення. Тобто після переходу від ланцюга до циклічної системи можна буде легко змоделювати поведінку m-відмовостійкої системи *K(m,n).*

Для підвищеної складності системи досить важко скласти логічну структурну одиницю, тому для того щоб розрахувати надійність за допомогою моделей проводиться ряд статистичних експериментів з ними. З першого огляду даних методів побудови можливо не зовсім все видається таким простим, хоча насправді дана методика має практично важливе значення. Для того, щоб це показати виконаємо побудову моделі для *K(4,8).*

Для початку необхідно окремо виділити два випадка, а саме з *(1,n)* та *(n,n).* Пам'ятаючи про попередні результати, для вище описаних параметрів реберні функції обраховуються як кон'юнкція та диз'юнкція з n елементів. Саме тому правдивим є твердження, що після розбиття множини змінних на підмножини по парам спрощує подальші перетворення і дозволяє відразу знайти всі можливі реберні функції для моделі. Якщо припустити, що допустимо 1 чи 2 відмови в потоці відмов, то це призводить до кон'юнкції або диз'юнкції:

$$К'(1, 2)= xi\ xi+1$$

$$К'(2, 2)= xi \vee xi+1$$

В таблиці 1 зображено розподіл змінних по підмножинам й варіанти того, як можуть розприділятися відмови у даних підмножинах. В даному випадку було обрано розприділити множину, що складається з 8-ми

аргументів, розбити на чотири підмножини, що і видно з першого рядка таблиці.

Таблиця 1 - Перший етап побудови

{x1,x2}	{x3,x4}	{x5, x6}	{x7, x8}
K (1, 2)	K (1, 2)	K (1, 2)	K (1, 2)
K (2, 2)		K (1, 2)	K (1, 2)
K (2, 2)	K (1, 2)		K (1, 2)
K (2, 2)	K (1, 2)	K (1, 2)	
K (1, 2)	K (2, 2)	K (1, 2)	
K (1, 2)	K (2, 2)		K (1, 2)
	K (2, 2)	K (1, 2)	K (1, 2)
K (1, 2)	K (1, 2)	K (2, 2)	
K (1, 2)		K (2, 2)	K (1, 2)
	K (1, 2)	K (2, 2)	K (1, 2)
K (1, 2)	K (1, 2)		K (2, 2)
K (1, 2)		K (1, 2)	K (2, 2)
	K (1, 2)	K (1, 2)	K (2, 2)
K (2, 2)	K (2, 2)		
K (2, 2)		K (2, 2)	
K (2, 2)			K (2, 2)
	K (2, 2)	K (2, 2)	
	K (2, 2)		K (2, 2)
		K (2, 2)	K (2, 2)

В таблиці 2 описано функції, що у нас вийшли. Згідно з цими даними отримано кільцевий граф, що складається з 19-ти ребер. При розбитті на чотири підмножини було отримано 19 реберних функцій, але можливі й інші варіанти розбиття.

Таблиця 2 - Набір функцій для *K (4,8)*

f1 = x1x2 ∨ x3x4 ∨ x5 x6 ∨ x7 x8
f2= x1 ∨ x2 ∨ x5 x6 ∨ x7 x8
f3= x1 ∨ x2 ∨ x3x4 ∨ x7 x8
f4= x1 ∨ x2 ∨ x3x4 ∨ x5 x6
f5= x1x2 ∨ x3 ∨ x4 ∨ x5 x6
f6= x1x2 ∨ x3 ∨ x4 ∨ x7 x8
f7= x3 ∨ x4 ∨ x5 x6 ∨ x7 x8
f8= x1x2 ∨ x3x4 ∨ x5 ∨ x6
f9= x1x2 ∨ x5 ∨ x6 ∨ x7 x8
f10= x3x4 ∨ x5 ∨ x6 ∨ x7 x8
f11 = x1x2 ∨ x3x4 ∨ x7 ∨ x8
f12= x1x2 ∨ x5 x6 ∨ x7 ∨ x8
f13 = x3x4 ∨ x5 x6 ∨ x7 ∨ x8
f14= x1 ∨ x2 ∨ x3 ∨ x4
f15= x1 ∨ x2 ∨ x5 ∨ x6
f16= x1 ∨ x2 ∨ x7 ∨ x8
f17= x3 ∨ x4 ∨ x5 ∨ x6
f18= x3 ∨ x4 ∨ x7 ∨ x8
f19= x5 ∨ x6 ∨ x7 □ x8

Для прикладу розглянемо й інший варіант, де вхідна множина розділяється на дві підмножини, що показані в першому рядку таблиці 3. Також там розписано різноманітні комбінації *K '(i, j)*.

Таблиця 3 – Комбінації *K '(i, j)*

{x1,x2,x3,x4}	{ x5,x6,x7,x8 }

К’(4, 4)	
К’(3, 4)	К’(1, 4)
К’(2, 4)	К’(2, 4)
К’(1, 4)	К’(3, 4)
	К’(4, 4)

Відповідно до таблиці 3 можливо виписати наступні рядки:

$$x1 \vee x2 \vee x3 \vee x4$$

$$x5 \vee x6 \vee x7 \vee x8$$

$$К’А(3,4) \vee x5x6x7x8$$

$$К’А(2,4) \vee К’В(2,4)$$

$$x1x2x3x4 \vee К’В(3,4)$$

Перші два рядки є уже сформованими реберними функціями. А щоб визначити інші функції, необхідно розписати й провести пошук певної сукупності функцій *К '(3,4)* і *К' (2,4).* Це і є наступний етап побудови. Таблиці формуються аналогічно до попередніх. Підмножини А і В розділимо на свої підмножини відповідно для першого на *{x1, x2}* і *{x3, x4},* а для другого - *{x5, x6}* і *{x7, x8}.*

Виконаємо розбиття кожного з підмножин змінних на свої підмножини: підмножина А на *{x1, x2}* і *{x3, x4}*, підмножина В на *{x5, x6}* і *{x7, x8}.* Для випадку *К '(3,4)* можемо скласти свою таблицю розподілу трьох відмов по підмножини потужності 2 підмножини А. А щодо випадку з *К'(3,4)* складемо окрему таблицю розподілу з трьох відмов на підмножини. Результат наглядно показано в наступній таблиці 4, що дає можливість виписати реберні функції за допомогою певних перетворень одного з рядків вище:

$$x1 \vee x2 \vee x3\ x4 \vee x5x6x7x8$$

$$x1x2 \vee x3 \vee \square\ x4 \vee x5x6x7x8.$$

Таблиця 4 – Другий етап побудови

{x1,x2}	{x3,x4}
K'A(2,2)	K'A(1,2)
K'A(1,2)	K'A(2,2)

Аналогічно до уже зробленого складається таблиця для розподілу двох відмов з урахуванням уже знайомих підмножин(див. Таблиця 5)

Таблиця 5 – Третій етап побудови

{x1,x2}	{x3,x4}
K'A(2,2)	K'A(1,2)
K'A(1,2)	K'A(2,2)

Для підмножини B виконуються ті ж самі дії, результат яких можна побачити в таблицях 6 й 7.

Таблиця 6 – Четвертий етап побудови

{x5,x6}	{x7,x8}
K'B(2,2)	K'B(1,2)
K'B(1,2)	K'B(2,2)

Таблиця 7 – П'ятий етап побудови

{x5,x6}	{x7,x8}
K'B(2,2)	
K'B(1,2)	K'B(1,2)
	K'B(2,2)

Враховуючи той факт, що усі *K'(i,j)* задовільняють вище виділені особливості, то можна виписати реберні функції, що ми і шукали:

f1 = x1 ∨ x2 ∨ x3 ∨ x4

f2 = x1 ∨ x2 ∨ x3 x4 ∨ x5x6x7x8

f3 =x1x2 ∨ x3 ∨ x4 ∨ x5x6x7x8

f4= x1 ∨ x2 ∨ x5 ∨ x6

f5= x1x2 ∨ x3 x4 ∨ x5 ∨ x6

f6= x3 ∨ x4 ∨ x5 ∨ x6

f7= x1 ∨ x2 ∨ x5 x6 ∨ x7 x8

f8= x1x2 ∨ x3 x4 ∨ x5 x6 ∨ x7 x8

f9= x3 ∨ x4 ∨ x5 x6 ∨ x7 x8

f10 = x1 ∨ x2 ∨ x7 ∨ x8

f11 = x1x2 ∨ x3 x4 ∨ x7 ∨ x8

f12= x3 ∨ x4 ∨□x7 ∨ x8

f13= x1x2x3x4 ∨ x5 ∨ x6 ∨ x7 x8

f14= x1x2x3x4 ∨ x5 x6 ∨ x7 ∨ x8

f15= x5 ∨ x6 ∨ x7 ∨ x8

На практиці після побудови моделей наглядно показано, що кількість ребер графа, тобто кількість реберних функцій, значно переважає по кількості в порівнянні з числом модулів ВБС. Але існує і позитивна сторона, а саме той факт, що не завжди функції залежать від усіх змінних. Це забезпечує швидшу обробку реберних функцій та обчислення значень цих функцій[20-23].

Згідно з уже побудованими моделями можна зробити висновок, що для однієї ВБС може існувати більше ніж один варіант побудови моделі. Хоча найбільшу цінність несуть ті моделі, що є відносно найпростішими, тобто з найменшою кількість реберних функцій чи найбільш простим їх виглядом. На практиці для отримання найоптимальнішого варіанту побудови GL-моделі необхідно виконувати розбиття множин на рівні підмножини по потужності, або ж максимально намагатися досягти рівності. Уже

досліджено, що існує можливість перетворити GL-моделі для того, щоб отримати значно спрощений варіант. Таким чином моделі, що виходять після перетворення, обов'язково мають меншу кількість реберних функцій, не дивлячись на можливість ускладнення самих функцій всередині. За рахунок таких перетворень досягається зменшення часу обробки як кожної окремо взятої реберної функції, так і всієї моделі повністю. Відповідно збільшується максимально можлива кількість перевірок за один статистичний експеримент. Таким чином при збільшенні кількості перевірок збільшується ' точність розрахунку надійності ВБС[24].

Моделі, що були побудовані, відносяться до базових. Базові GL-моделі характеризуються тим, що є обов'язково відмовостійкі й їх кратність не повинна перевищувати числа *m*. На практиці найцікавіше розглядати системи, що принципово відрізняються від базових. Для прикладу цікавим випадком є система, що може бути не лише m-відмовостійка, а й залишатися роботоздатною при деяких комбінаціях, починаючи з *(m+1)*-ї відмови. Побудова GL-моделей, що не відносяться до базових, є досить непростою задачею, одним з рішенням якої є додавання до системи додаткових ребер. Також ще однією з задач є знаходження й визначення числа ребер, що повинні випасти з графа при появі певних відмов у потоці, при яких система залишається відмовостійкою, тобто граф моделі не втрачає зв'язність. Доказано той факт, що при *(m + 1)*-ій відмові, з GL-моделі обов'язково повинні випасти два ребра, а максимальна кількість ребер, що можуть бути видалені, дорівнює *m + 1*.

2.3 Особливості використання універсальних GL-моделей

На сьогодні згідно з уже існуючим рівнем розвитку і наукового прогресу, світова практика показує досить часте використання

багатопроцесорних систем, що містять у собі сотні елементарних одиниць. Багатопроцесорні системи охоплюють велике число різноманітних сфер, наприклад різні підрозділи науки, а також в управлінні розповсюджених об’єктів авіації та атомних електростанцій. Оскільки дані структури входять до складу досить складних систем, то важливою задачею є забезпечення параметрів систем, зокрема надійності.

Для забезпечення правильності й точності при розрахунку надійності системи використовуються різні моделі, але на сьогодні найбільш актуальним є побудова GL-моделей. До найбільших переваг даного методу відносять той факт, що модель основана на теорій графів, а також теорії булевих функцій. Це дозволяє проводити управління складних об’єктів при появі відмов процесорів й виконувати розрахунки надійності системи при ряді експериментів з графо-логічними моделями[25].

Навідмінно від інших існуючих моделей, особливістю даної графо-логічної моделі є її представлення за допомогою неорієнтовного графа з ребрами, що відповідають булевим функціям. Якщо граф зв’язний, то система є відмовістійкою й виконує всі функції справно. В залежності від значення, що може приймати реберна функція, вирішується, чи буде видалятися дане ребро. Якщо відповідна реберна функція приймає нульове значення, то відповідне ребро викидається.

Для початку необхідно розглянути базові ВБС. Такі системи складаються з n процесорів й продовжують повне функціонування при числі відмов не більше за m будь-яких структурних одиниць. На практиці базові системи позначаються таким чином K(m,n) .

Для прикладу наведемо декілька подібних систем на рисунках 10, 11.

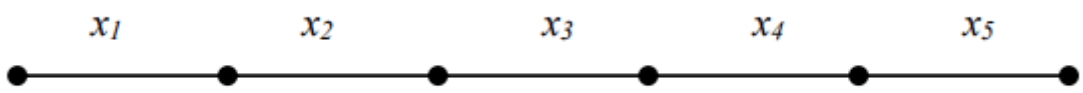


Рисунок 10 - ВБС K(0,5)

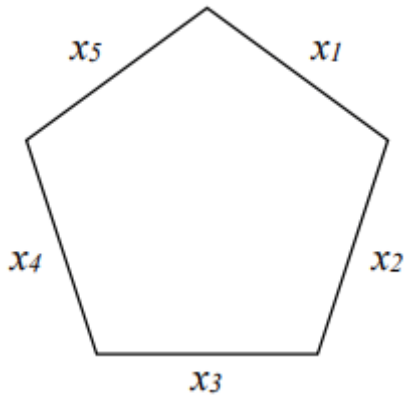


Рисунок 11 - ВБС K(1,5)

Чим більше максимальне число допустимих відмов, тим складніша і графо-логічна модель. При побудові найбільша увага приділяється саме знаходженню реберних функцій. Побудована модель може одночасно бути і базовою, і циклічною.

Розглянемо таку систему *K(m,n)*, що є роботоздатною при кількості відмов менше за *(m+1)* елементів системи. Для початку необхідно виконати розбиття множини змінних на рівні підмножини відносно потужності, а тепер модель системи представлена набором таких реберних функцій:

$$F_{m,[\frac{n}{2}]}$$

$$F_{m-1,[\frac{n}{2}]} \vee F'_{1,n-[\frac{n}{2}]}$$

$$F_{m-2,[\frac{n}{2}]} \vee F'_{2,n-[\frac{n}{2}]}$$

$$\dots$$

$$F_{2,[\frac{n}{2}]} \vee F'_{m-2,n-[\frac{n}{2}]}$$

$$F_{1,[\frac{n}{2}]} \vee F'_{m-1,n-[\frac{n}{2}]}$$

$$F'_{m,n-[\frac{n}{2}]}$$

В даному випадку під F мається на увазі певну сукупність реберних функцій для графо-логічної моделі. Множині А належать функції типу *F*, а до множини В – типу *F'*.

Як завжди, для окремих крайніх випадків реберні функції знаходяться простими операції кон'юнкції та диз'юнкції набору змінних x_i:

$$F_{1,n} = x_1 \cdot x_2 \cdot \ldots \cdot x_{[\frac{n}{2}]} \qquad F'_{1,n} = x_{[\frac{n}{2}]+1} \cdot x_{[\frac{n}{2}]+2} \cdot \ldots \cdot x_n$$

$$F_{n,n} = x_1 \vee x_2 \vee \ldots \vee x_{[\frac{n}{2}]} \qquad F'_{n,n} = x_{[\frac{n}{2}]+1} \vee x_{[\frac{n}{2}]+2} \vee \ldots \vee x_n$$

А от для подальшої побудови функцій при $1<m<n$ проводяться подальші розбиття, які полягають в тому, щоб розбити функції на менш складні. Наприклад, для системи з параметрами (2,6) будуть такі етапи розбиття:

1)

$$K(2,6) \rightarrow \quad F_{2,3} \quad F_{1,3} \vee F'_{1,3} \quad F'_{2,3},$$

$$F_{1,3} = x_1 \cdot x_2 \cdot x_3$$

$$F'_{1,3} = x_4 \cdot x_5 \cdot x_6$$

2) Розкладемо функцію на більш прості складові:

$$F_{2,3} \rightarrow \quad F_{2,2} \quad F_{1,2} \vee F'_{1,1}$$

$$F_{2,2} = x_1 \vee x_2$$

$$F_{1,2} = x_1 \cdot x_2$$

$$F'_{1,1} = x_3$$

Відповідно:

$$F_{2,3} \rightarrow \quad x_1 \quad \vee \quad x_2$$
$$x_1 \cdot x_2 \quad \vee \quad x_3$$

3) Аналогічно і для подібної функції з множини B:

$$F'_{2,3} \rightarrow \quad x_4 \quad \vee \quad x_5$$
$$x_4 \cdot x_5 \quad \vee \quad x_6$$

4) Отже, результуючим набором реберних функцій буде:

$$K(2,6) \rightarrow \quad x_1 \vee x_2$$
$$x_1 \cdot x_2 \vee x_3$$
$$x_1 \cdot x_2 \cdot x_3$$
$$x_4 \cdot x_5 \cdot x_6$$
$$x_4 \vee x_5$$
$$x_4 \cdot x_5 \vee x_6$$

Важливим фактором є знання про точну кількість ребер, що випаде з графа після появи певного набору відмов. Тому будемо розглядати таке твердження: при відмові двох елементів у системі з множини реберних функцій, що було отримані після подібного розбиття, лише одна функція приймає нульове значення.

Для початку необхідно зауважити такий факт, що при початкових установках змінних з лише одним значенням нуля. В такому разі жодна з реберних функції, що будуть формуватися для цієї системи, не буде рівна 0. Згідно з описаним вище алгоритмом відбувається поділ множини на дві найбільш рівні частини $\{x_1, x_2, \ldots, x_{[\frac{n}{2}]}\}$ й $\{x_{[\frac{n}{2}]}+1, \ldots, x_n\}$.

Якщо нульові елементи будуть в різних підмножинах, то реберна функція $\varphi = x_1 \cdot x_2 \cdot x \ldots x_{[\frac{n}{2}]} \vee x_{[\frac{n}{2}]+1} \cdot x_{[\frac{n}{2}]+2} \ldots \cdot x_n$ також дорівнюватиме 0. Використовуючи попередній вираз стає зрозуміло, що серед інших реберних функцій не може бути нульових, адже вони вміщують максимальні дві змінні зі значенням 0. При обставинах, коли всі несправні модулі знаходяться в одній підмножині, тоді $\varphi \neq 0$, але обов'язково при подальших розбиттях

знайдеться функція, що дорівнюватиме нулю й обов’язково матиме у різних підмножинах парочку нульових змінних.

Як наслідок з попередньо доказаного твердження, в системі K(m, n) при умові, що сталося *m* відмов, буде втрачено одне ребро даної базової графо-логічної моделі.

З попереднього доведення зрозуміло, що для забезпечення зв’язності графа, неможливо щоб двохреберні функції були нульові в один і той же проміжок часу. Розподіл функцій відбувається послідовно, крок за кроком генеруються нові булеві функції для ребер графа.Кожна така функція є диз’юнкцією. Враховуючи усе вищеописане, очевидним є факт, що хоча б одна одиниця у функції виявиться нулем.

При наявності двох несправностей у системі графо-логічна модель або ж взагалі не втрачає ребер, або ж виключається одне ребро з системи. В реальних сферах базових систем майже немає, більш актуальними є системи, що залишаються роботоздатними й повність функціонують навіть при певному наборі відмов.

Актуальним є проблема відображення реакції моделі на відмови з однаковою кратністю. Для того, щоб вирішити дану проблему, можливо виконати такі дії, як проведення додаткових ребер з булевими функціями, які дорівнюють одиниці при появі таких однакових за кратність відмов, або ж інакше рівні нулю.

Розрахунок параметрів БВС можливий ще на етапі побудови GL-моделі, а саме обчислення надійності за допомогою статичного моделювання роботи системи в потоці відмов. Для того, щоб забезпечити високу точність отриманих результатів, необхідно збільшити кількість експериментів з різноманітними наборами відмов. Чим більше число модулів моделі, тим важчою і недоцільною стає задача пошуку параметра надійності, тому необхідно зменшувати час проведення кожного експерименту.

Число ребер, що необхідно провести для забезпечення роботоздатності, зменшується при умові, що граф втратить найменшу кількість ребер при

появі відмов у ВБС. Тому важливою задачею є пошук найоптимальнішого варіанту з мінімально необхідним числом внутрішніх ребер.

Візьмемо таку багатопроцесорну систему, яка залишається роботоздатною при появі двох, чи навіть трьох відмов. Згідно з раніше отриманими результатами, при максимально можливій кількості відмов трьох процесорів, граф може втратити два або ж три ребра. Тобто для того, щоб гарантувати зв'язність системи, необхідно провести 1 або 2 внутрішніх ребра.

В іншому ж випадку GL-модель втрачає рівно два ребра при появі трьох відмов.

У випадку побудови графо-логічної моделі для системи K(2,n) помітно, що алгоритм поділяється на три дії, що зображено на рисунку 12.

На рисунку нище зображено загальну схема, адже кожний з 1-го та 3-го етапів поділяється на менші дії аналогічного характеру, але вже для обробки меншої кількості даних. А на другому етапі формуються функції вигляду Fi, яка стає рівною нулю тільки в одному варіанті, а саме коли існують нульові елементи з обох сторін від знаку диз'юнкції.

На рисунку 6 зображено загальну схема, адже кожний з 1-го та 3-го етапів поділяється на менші дії аналогічного характеру, але вже для обробки меншої кількості даних. А на другому етапі формуються функції вигляду Fi, яка стає рівною нулю тільки в одному варіанті, а саме коли існують нульові елементи з обох сторін від знаку диз'юнкції[26,27].

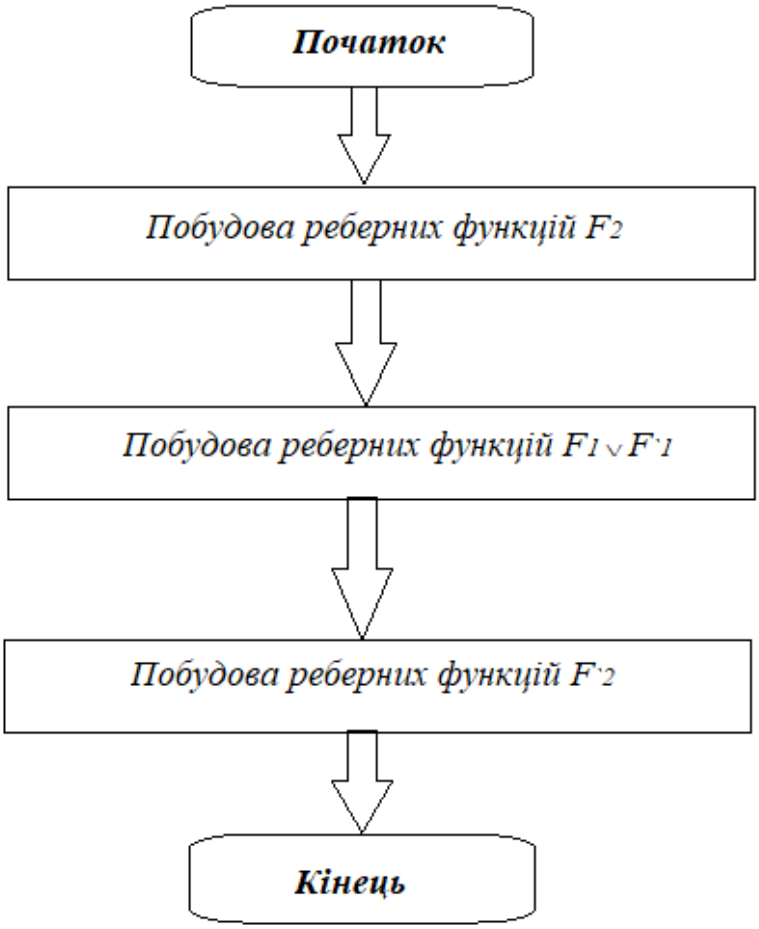


Рисунок 12 – Частина алгоритму побудови GL-моделі для K(2,n).

Висновки до розділу 2

Однозначно існує проблематика спроможності до тестування при проектуванні цифрових схем: проектування ВБС, що мають порівняно прості моделі їх поведінки в потоці відмов, з тим, щоб їх легше було тестувати в процесі виконання статистичних експериментів при розрахунку надійності.

Можна відзначити, що дослідження так званих систем «k з n» присвячено досить багато робіт особливо в останні роки. K-out-of n: G - система працездатна, якщо k модулів з n функціонують, тобто в нашому визначенні це - базова система. Цей інтерес зайвий раз підкреслює теоретичну і практичну важливість проблеми, з одного боку, і складність самої проблеми (а ВБС базові - найпростіші об'єкти) - з іншого.

Для того, щоб побудувати систему, що буде залишатися роботоздатною навіть при появі двох або трьох несправностей, необхідно і достатньо провести одне додаткове внутрішня ребро, при цьому складність системи залишається такою ж мінімальною. Віповідно й час проведення експерименту не втратить у швидкодії при подібних модифікаціях з додаванням внутрішніх ребер.

3 ПОБУДОВА ВЛАСНОЇ GL-МОДЕЛІ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 GL -модель багатопроцесорної системи, що складається з трьох шин

Багатопроцесорні системи використовуються в багатьох галузях науки і техніки, в тому числі при управлінні об'єктів авіації, станцій. Подібне застосування виставляє жорсткі вимоги до забезпечення високого рівня надійності систем їх управління.

Для розрахунку надійності складних відмовостійких систем в даний час використовується безліч різних моделей, серед яких важливе місце займають графо-логічні (GL). Їх особливістю є поєднання достоїнств теорії графів і булевих функцій, що дозволяє аналізувати поведінку відмовостійких багатопроцесорних систем (ВБС) управління складними об'єктами в потоці відмов за допомогою доступних обчислювальних засобів і виконувати розрахунок надійності ВБС шляхом проведення статистичних експериментів з моделями.

GL-модель може бути побудована для системи будь-якої складності і відповідає її поведінці в потоці відмов. Щоб побудувати модель для багатошинної системи, необхідно умовно розділити її на простіші частини й підрівні. Модель являє собою неорієнтовний граф, кожне ребро якого відповідає булевій функції, що називається реберною функцією. Якщо реберна функція приймає значення нуля, то відповідне ребро виключається. Як тільки граф втрачає зв’язність, система перестає бути роботоздатною. За аргумент реберної функції береться вектор стану системи, що складається зі значень булевих змінних, кожна з яких відображає стан одного з процесорів системи. Якщо змінна дорівнює одиниці, то процесор справний, а якщо одиниці – несправний.

Для побудови графо-логічної моделі візьмемо систему з кількістю процесорів N=42 і максимально можливим числом несправностей T=6, що

зображена на рисунку 13. Система має три шини по 10 процесорів на кожній. Між кожною парою шин є по 4 процесора, але мінімальна кількість справних процесорів між кожною парою шин дорівнює 2, а на кожній шині – по 8. Щоб побудувати модель вищеописаної моделі, необхідно виділити основні підсистеми. Кожна з таких підсистем іноді може ділитися ще на декілька підсистем. Подібні системи називаються ієрархічними.

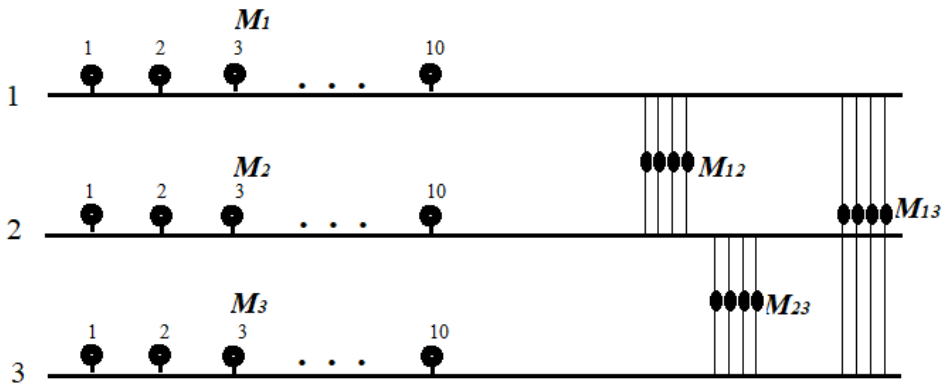


Рисунок 13 – Схематичне зображення багатошинної системи

Візьмемо загальний випадок багатошинної системи з N процесорами, серед яких допускається T несправних. Багатошинна багатопроцесорна система – це система, що складається з двох чи більше шин, кожна з яких містить певну кількість процесорів, а також кожна пара шин обов'язково пов'язана через визначену кількість процесорів. Для забезпечення її відмовостійкості на початку визначаються максимальні значення для кількості справних процесорів на кожній шині і числа міжшинних процесорів. Менше цих значень не можна допускати, інакше система стане не повністю роботоздатною і, відповідно, втратить відмовостійкість. Важливою задачею є знаходження залежностей між відомими параметрами багатошинної системи та l й m – число процесорів на кожній шині й кількість міжшинних процесорів відповідно.

Для початку опишемо параметри, які заздалегідь відомі розробнику:

- N – загальна кількість процесорів;
- T – максимально допустиме число несправних процесорів, при яких система є роботоздатною;
- k – кількість шин;
- q_i – число справних процесорів на шині, менше якого не можна допускати;
- S – число міжшинних процесорів між двома шинами, менше якого не можна допускати.
- А також параметри, що мають невизначений характер на початку:
- m – число міжшинних процесорів між кожною парою шин;
- l_i – число процесорів на кожній шині.

В даній роботі приймаємо, що значення q_i та l_i будуть однакові для кожної шини, тому відповідно маємо такі прості обмеження: $m \geq S,\ l \geq q$.

Кількість всіх міжшинних легко визначається за співвідношенням:

$$\sum_i m_i = m \cdot C_k^2 = m \cdot \frac{k!}{(k-2)! \cdot 2!} \quad (1)$$

Усі процесори умовно можна поділити на міжшинні, число яких можна вирахувати через формулу (1), а кількість процесорів на шинах дорівнює добутку l і k. Загальна кількість процесорів буде дорівнювати сумі процесорів на шинах і міжшинних, що описано у формулі (2).

$$N = l \cdot k + m \cdot C_k^2 \quad (2)$$

З формули (2) за необхідністю можливо виділити параметр l.

Нехай значення r_i – кількість процесорів, що вийшли з ладу на i-ій шині, при чому $\sum_i r_i \leq T$.

Ми приймаємо, що для забезпечення роботоздатності системи необхідно, щоб всі шини були роботоздатними, для цього міжшинні процесори можуть приєднуватися до процесорів на шині у випадках, коли їх число на шині стає менше допустимого. В такому разі приєднані процесори

виключаються з числа міжшинних. І при цьому випливає таке обмеження: $r_i < l - q_i$.

Багатошинна система поділяється на підсистеми. Для кожної підсистеми відбувається побудова відповідної моделі, тобто для ієрархічної системи будується ієрархічна модель (ІМ). Всі підсистеми об'єднуються за допомогою моделі $M(0,5)$, де число 5 відповідає кількості груп. ІМ поділяється на такі групи:

1) Моделі, що відповідають поведінці процесорів на кожній шині: $M_1(2,10)$, $M_2(2,10)$, $M_3(2,10)$.
2) Моделі, що відповідають поведінці міжшинних процесорів: $M_{12}(2,4)$, $M_{13}(2,4)$, $M_{23}(2,4)$.
3) Моделі $M_1^*(1,3)$, $M_2^*(1,3)$, $M_3^*(1,3)$.
4) $M_{\text{вс}}(6,42)$
5) $M_{\text{мш}}(1,3)$

Побудуємо одну з моделей, а саме $\text{M}_1(2,10)$ Для початку необхідно описати реберні функції:

f1 = x1 ∨ x2x3x4x5

f2 = x2 ∨ x3x4x5x6

f3 = x3 ∨ x4x5x6x7

f4= x4 ∨ x5x6x7x8

f5= x5 ∨ x6x7x8x9

f6= x6 ∨ x7x8x9x10

f7= x7 ∨ x8x9x10x1

f8= x8 ∨ x9x10x1x2

f9= x9 ∨ x10x1x2x3

f10 = x10 ∨ x1x2x3x4

Граф даної моделі має вигляд, що представлений на рисунку 14

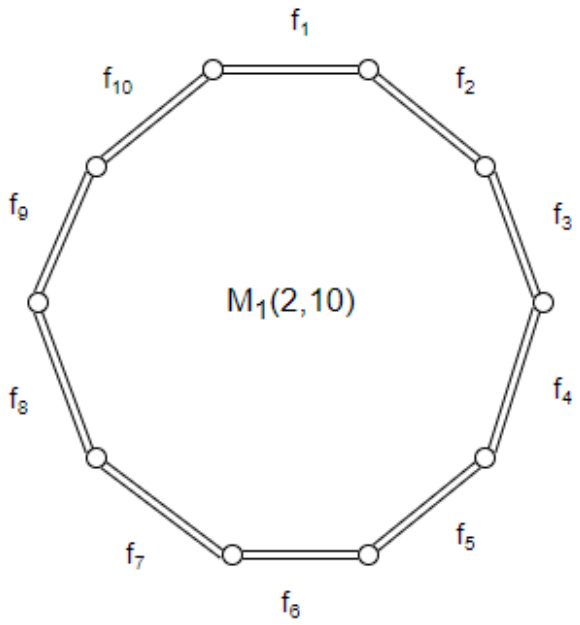


Рисунок 14 – Модель $M_1(2,10)$

Інші моделі будуються по аналогічній схемі. Як результат буде побудована GL-модель для багатошинної системи К (6, 42)(див. рис. 15).

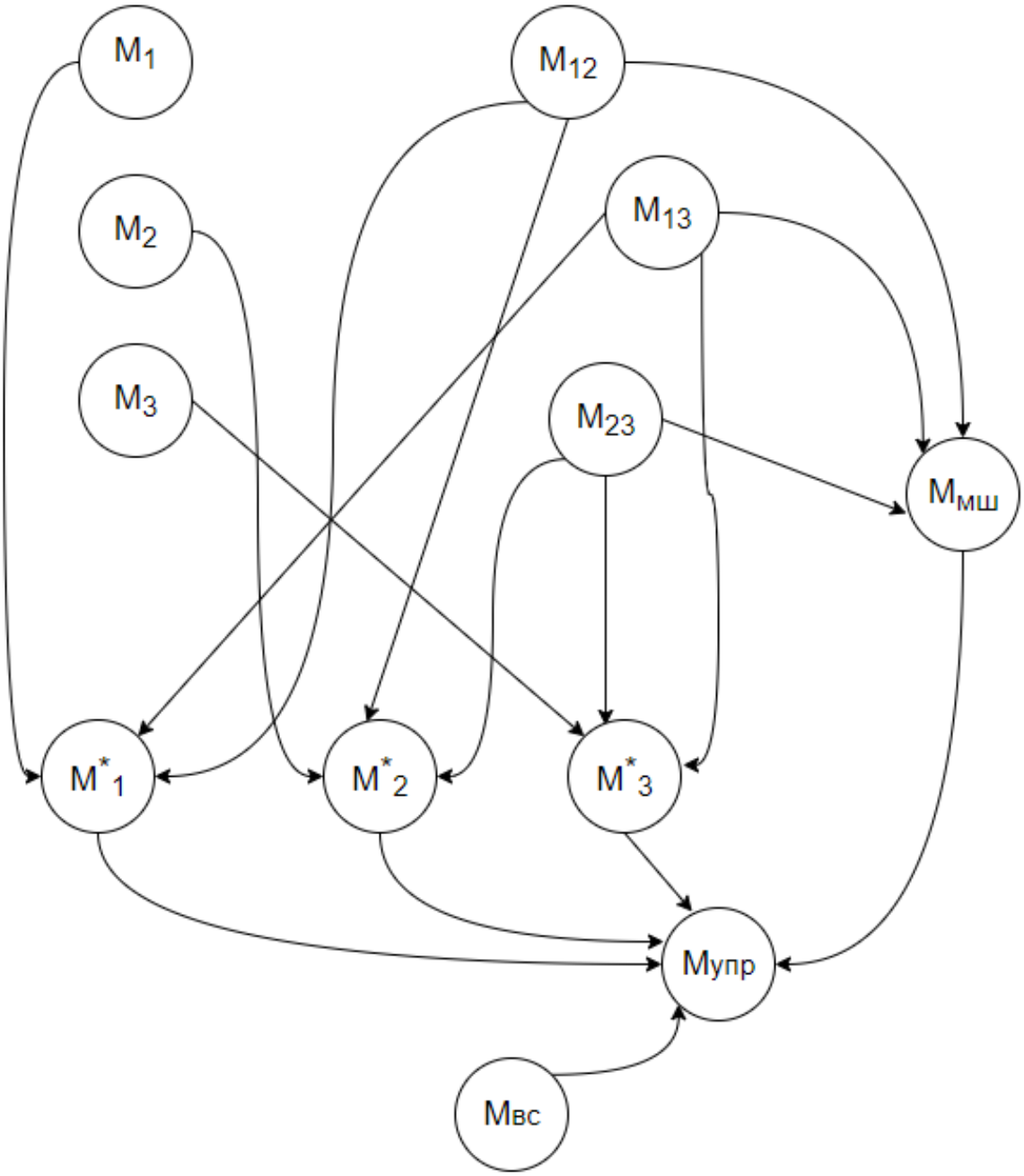


Рисунок 15 – GL-модель для БС, що складається з трьох шин

Також для кращого розуміння процесу формування моделей на рис. 16 описано алгоритм функціонування базової моделі.

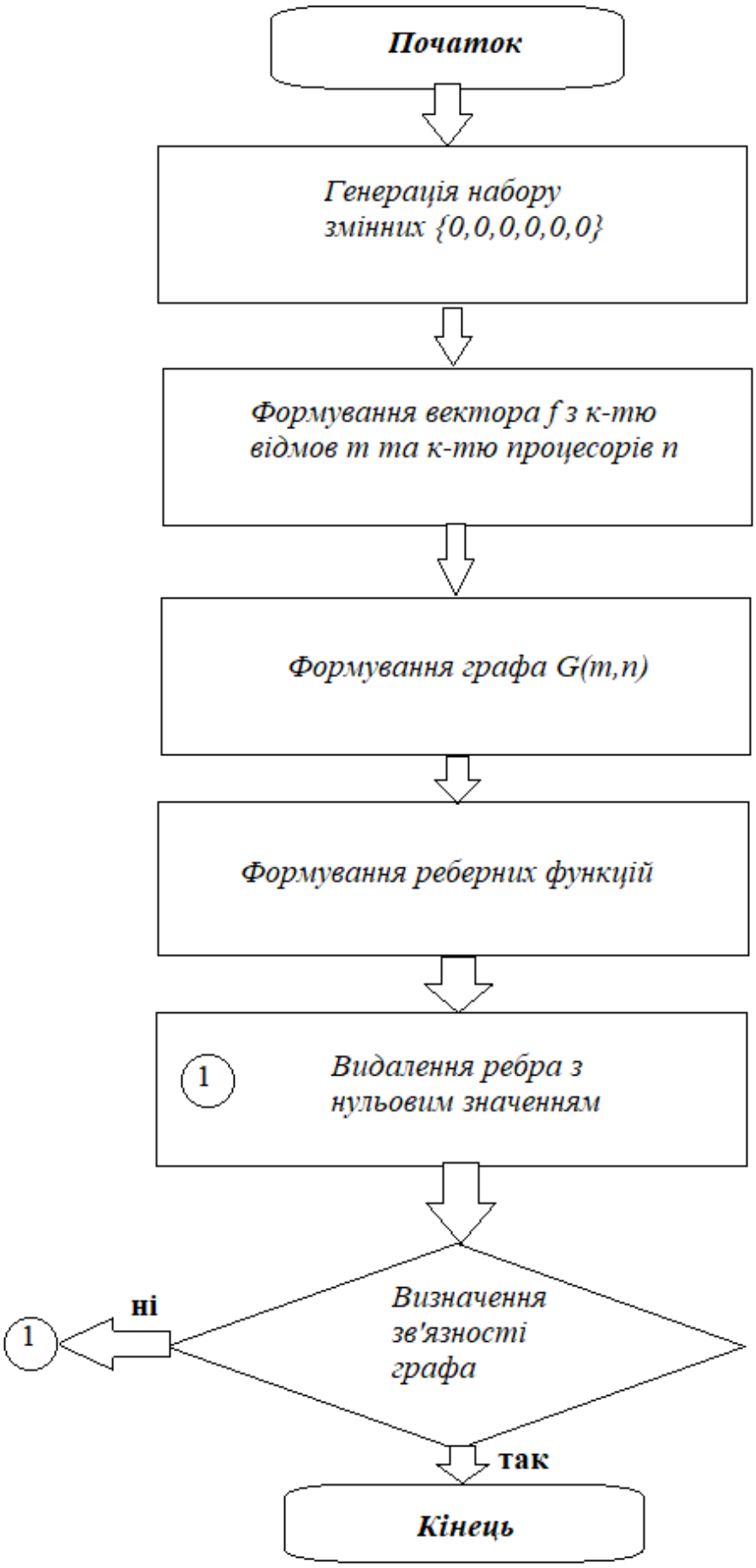


Рисунок 16 – Алгоритм роботи базової моделі

3.2 Реалізація програми для організації тестування

3.2.1 Опис обраних програмних засобів

Для розробки даної програми було обрано мову програмування Java. Згідно рейтингу 2020 року саме ця мова займає перше місце (див. рис.17). В Java добре організована система збирача сміття. Наприклад в C++ збирання сміття необхідно виконувати самостійно, тобто розробник напряму взаємодіє з пам'яттю, що однозначно негативно впливає на процес, адже при роботі з великим обсягом даних існує ймовірність вчасно не запустити процес збирання. А в Java подібні проблеми не виникають[28].

Рисунок 17 – Рейтинг мов програмування 2020 року

Розробники даної мови програмування заклали велику кількість засобів та бібліотек для написання коду, що допомагають автоматично вирішити проблеми, з якими приходиться боротися вручну в інших мовах програмування.

Оскільки Java є строго типізованою, то більшість помилок виявляються ще на попередніх етапах роботи компілятора та інтерпретатора. Однією з найбільш важливих особливостей даної мови програмування є досить передбачуване виконання програм.

Надійність Java досягається за рахунок автоматичної організації управління пам’яттю. Адже основні причини збоїв програми це помилки при управлінні пам’яттю та неправильно описана обробка виключень. Друга проблема теж вирішена за допомогою введення об’єктно-орієнтованого механізму для обробки виняткових ситуацій.

Порівняно з C++, де раніше з’явилася обробка виняткових ситуацій, в Java було використано абсолютно інший підхід. Тут необхідність використання обробки винятків є обов’язковою умовою для правильності написання коду.

Не дивлячись на всі переваги, ця мова програмування могла б стати просто однією з чергових в історії розвитку програмування. Але Java побудована таким чином, що на виході компілятора утворюється байт-код, який згодом виконується за допомогою Java Virtual Machine(JVM). Віртуальна машина це інтерпретатор команд для байт-коду(див. рис. 18).

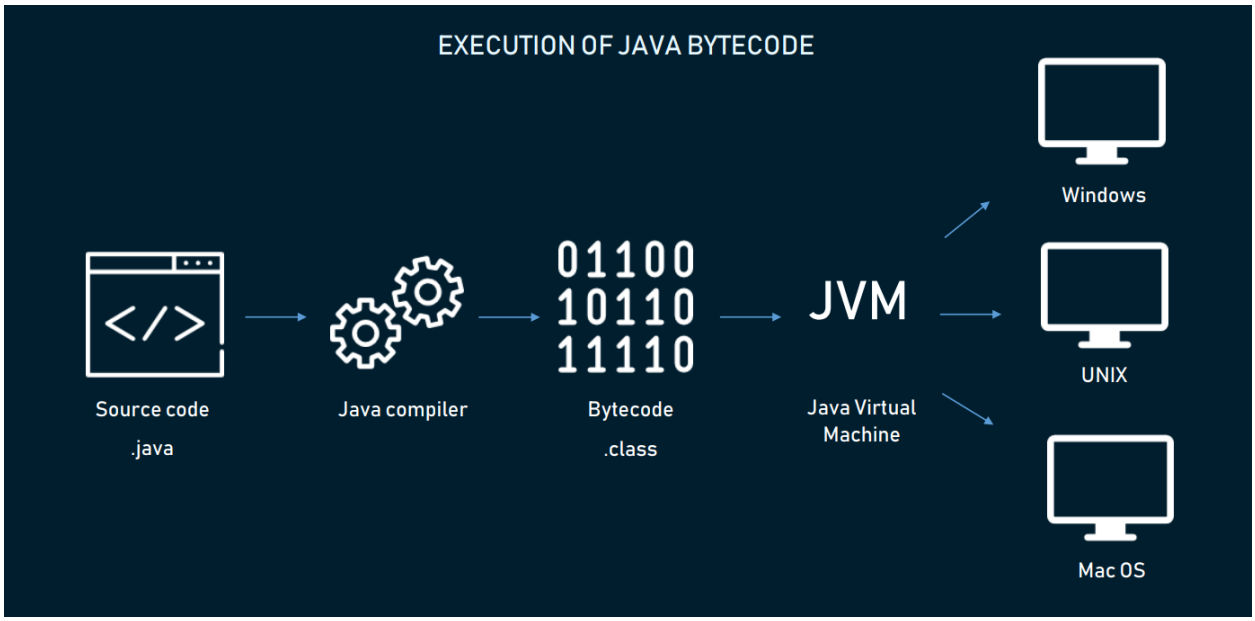


Рисунок 18 – Процес компіляції програми

Описаний вище підхід компіляції програми надав певні переваги, до яких можна віднести можливість виконувати програму на будь-якому пристрої, незалежно ні від типу процесора, ні від операційної системи. Тобто байт-код виконується однаково для будь-якої віртуальної машини й не потрібно робити спеціальні налаштування для середовища. Другою перевагою є захищеність програм від спроб посторонніх осіб зашкодити програмі. Таким чином комп'ютер перебуває під захистом.

Всі об'єкти знаходяться в динамічно виділеній пам'яті (heap). Доступ до будь-якого об'єкту відбувається лише за допомогою посилань. Існує спеціальне значення null, що описує той факт, що посилання не пов'язане з жодним з об'єктів.

Віртуальна машина здатна виконувати перевірку сумісності типів ще на етапі інтерпретації. Саме наявність дескрипторів надає можливість генерувати винятки при виявленні помилок.

Відмінність посилань у Java від вказівників C/C++ полягає в наступному:

- в Java неможливо звернутися до попереднього об'єкта в пам'яті за допомогою посилань;
- в Java посилання існують лише для об'єктів. Неможливо звернутися через посилання до змінної примітивного типу чи функції;
- в Java не існує можливості працювати з посиланнями, як зі значенням примітивного типу.

Мова програмування Java є об'єктно-орієнтованою мовою програмування. В даній концепції розробник визначає тип та структуру даних й потім наділяє цю структуру, що називається об'єктом, набором певних методів[29].

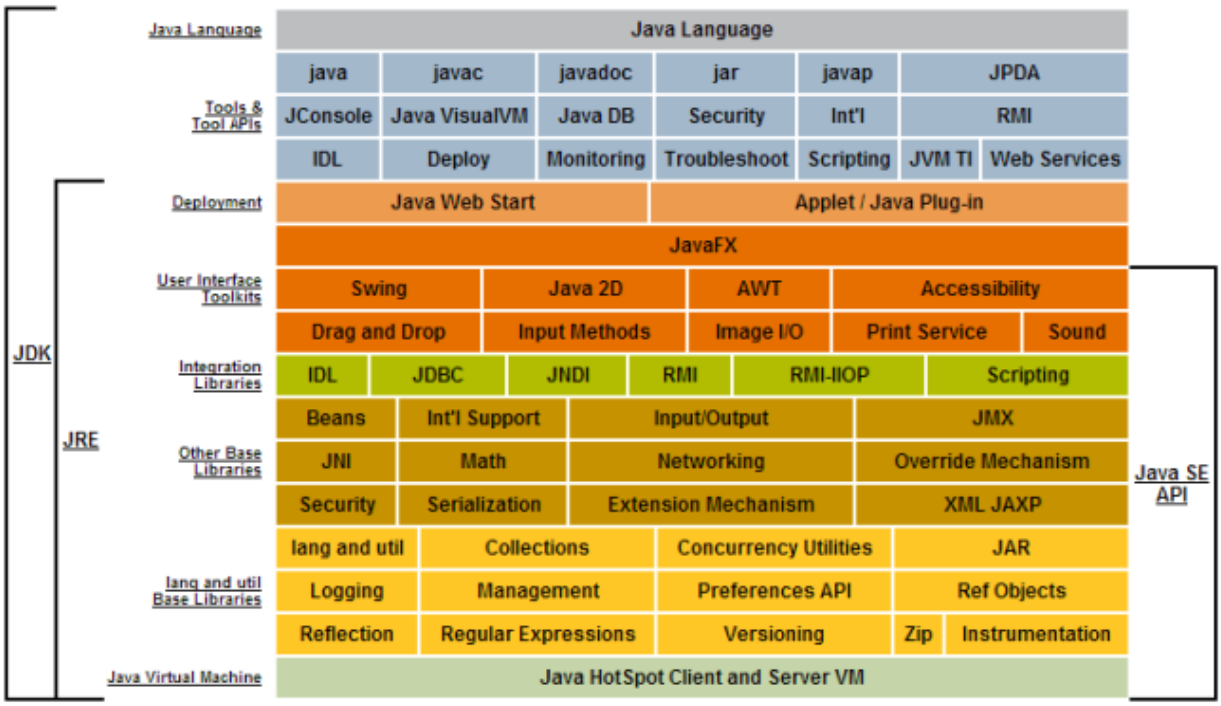


Рисунок 19 – Основні складові частини Java

Клас є моделлю ще не створеного об’єкту, тобто він описує як влаштований об’єкт, що є своєрідним нарисом майбутнього об’єкта. Основними принципами ООП є абстракція, інкапсуляція, наслідування та поліморфізм. Інкапсуляція дозволяє в майбутньому змінювати реалізацію без обов’язкових змін інтерфейсу.

Навідмінно від C++ в Java можливе лише одиничне наслідування, що полягає в таких пунктах:

- граф наслідування є деревом;
- кожний клас має мінімум одного нащадка;
- всі об’єкти є дочірніми по відношенню до класа Object (рис. 19).

За допомогою поліморфізму можна перевизначити методи, вибір метода буде відбуватися на етапі компіляції.

Існує ще один підхід – процедурне програмування, де використовуються наперед визначені функції, а також інструкції для роботи з ними. Різниця між цими підходами показана на рис. 20.

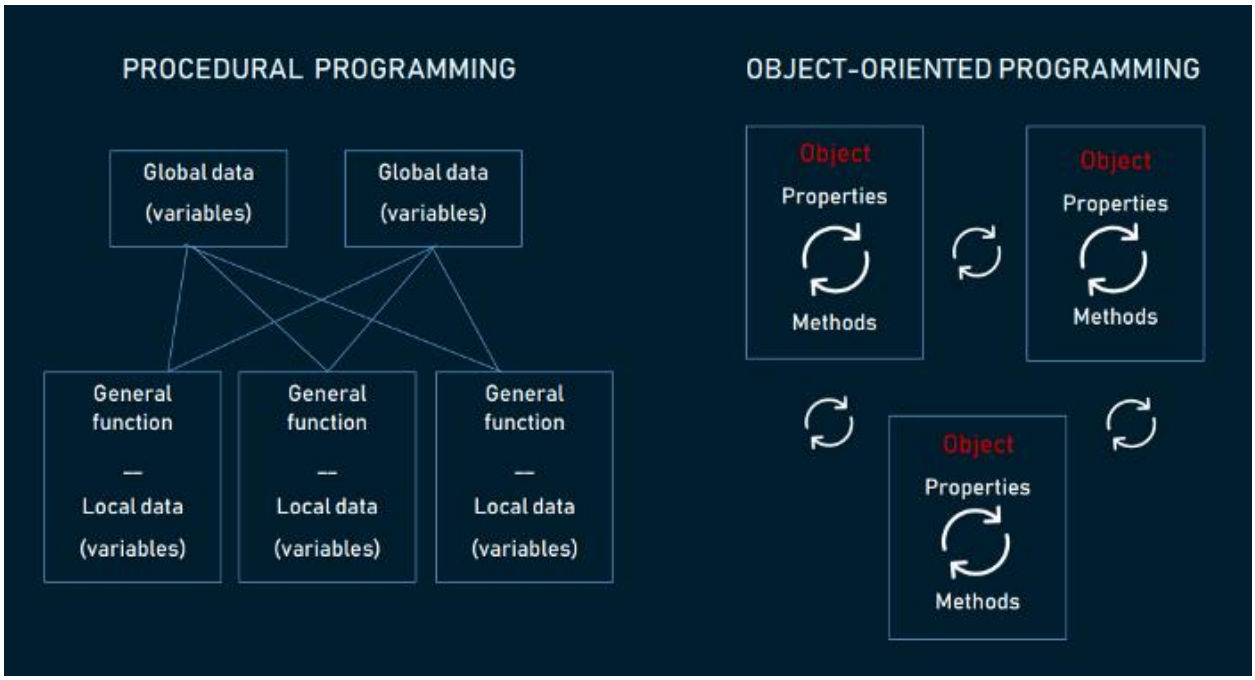


Рисунок 20 – Порівняльна схема двох підходів програмування

Графічний інтерфейс користувача (GUI) надає звичайним користувачам ПК зрозуміле відображення роботи програми, а також забезпечує можливість працювати з даними та безпосередньо самою програмою. Плюс до всього до GUI належать різні складові, такі як вікно, кнопки, мітки, поля, списки та інші. Деякі з компонент мають властивість відкривати нове вікно чи виконувати методи.

Пакет бібліотеки Java *.awt (Abstract Window Toolkit) є незамінним помічником при створенні програм для операційної системи (ОС) Windows. Він допомагає створювати інтерфейс, що візуально виглядає звичним та просто для користувача. Вперше у Java стало можливим створювати подібні додатки, що могли працювати на будь-якому комп’ютері, що підтримувався JVM. До цього графічні інтерфейси створювалися окремо під кожне середовище. Компоненти цієї бібліотеки є занадто великими і потребують чимало місця у пам’яті. Контейнер вміщує в собі уже створені раніше об’єкти, таким чином забезпечується завантаження компонент.

Пізніше було розроблено ще більш просту у використанні бібліотеку Swing, в якій з’явилися такі складні компоненти, як таблиці, дерева й панелі з

вкладками. Компоненти Swing є значно легшими, а сама бібліотека не має залежності від платформи. Дана бібліотека була побудована за допомогою одного з шаблонів проектування MVC.

До особливостей бібліотеки Swing можна віднести наступне:

- великий вибір компонент та їх налаштування;
- різноманітність елементів управління;
- легка вага;
- вигляд може змінюватися.

Swing і AWT відрізняються тим, що перший набір інструментів є покращеною версією другої бібліотеки[28,29].

На рисунку 21 зображено повну ієрархію класів бібліотеки.

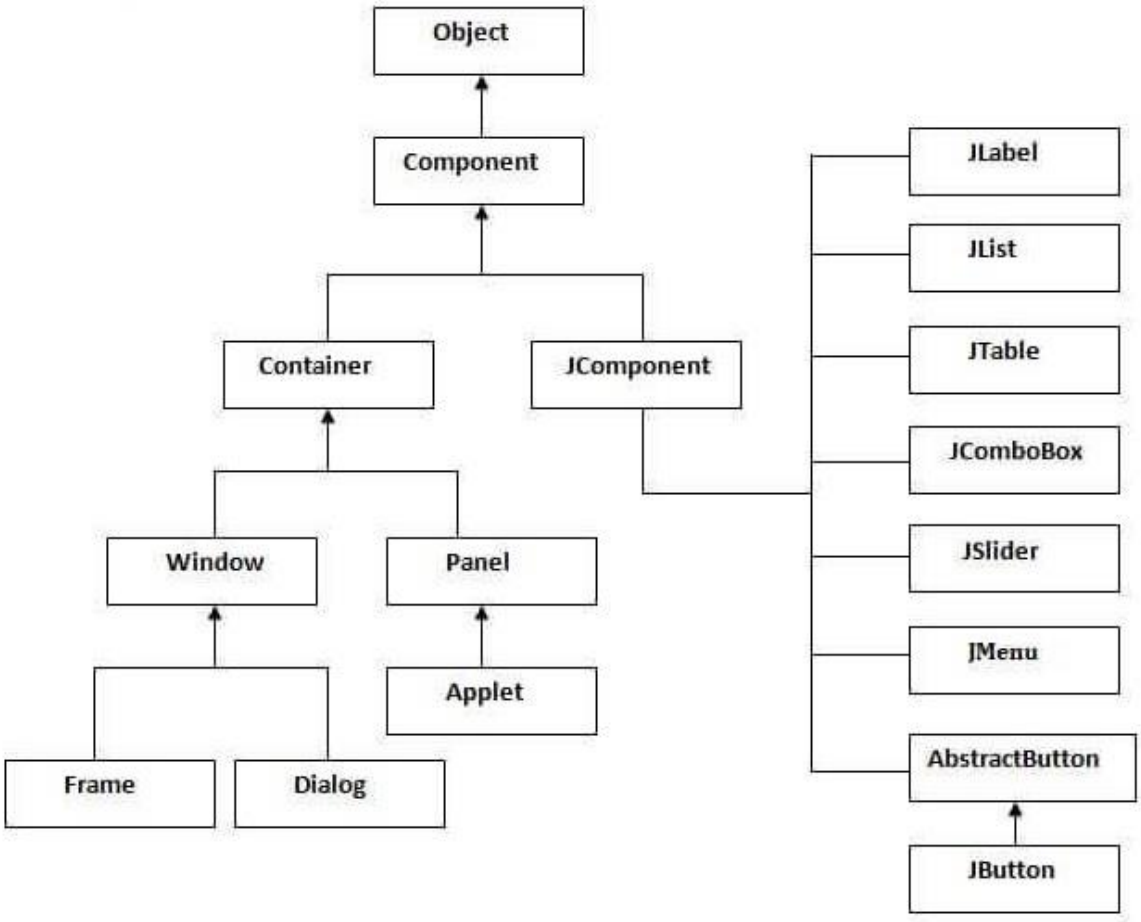


Рисунок 21 – Схема наслідування класів бібліотеки Swing

3.2.2 Інструкція користувача

Розроблена програма виконує наступні функції:

1) Побудова діагностичного графа-циркулянта з заданими параметрами.
2) Знаходження 0-ланцюжків.
3) Знаходження справного процесора.
4) Тестування системи на предмет виявлення справності чи несправності процесорів.

На рисунку 22 відображено початкове вікно, що відображається при запуску програми.

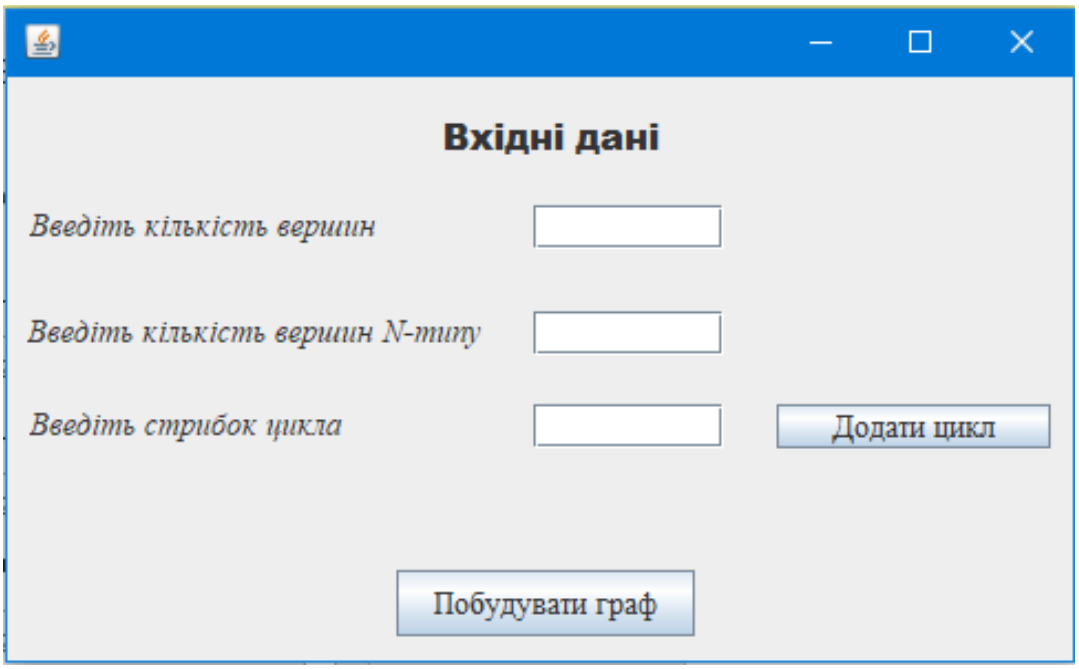


Рисунок 22 – Вікно задання вхідних даних

Програма будує циклічний граф, рандомно обираючи які процесори будуть несправні. Для того, щоб переглянути останній створений граф, неохідно вікрити папку з проектом або програмою й відкрити файл формату .png. Структура схема взаємозв’язків між модулями проекту показано на рисунку 23. Для оформлення графічного інтерфейсу користувача використовувався дизайнер WindowsBuilder.

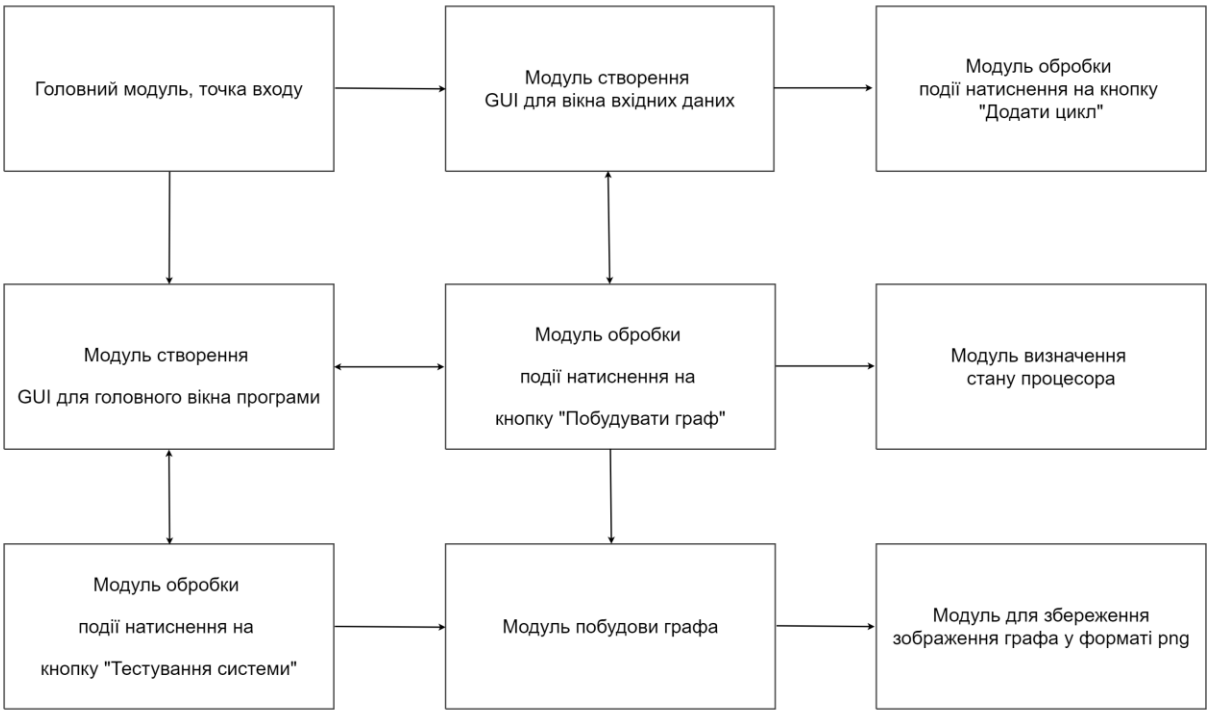


Рисунок 23 – Структурна схема взаємодії модулів програми

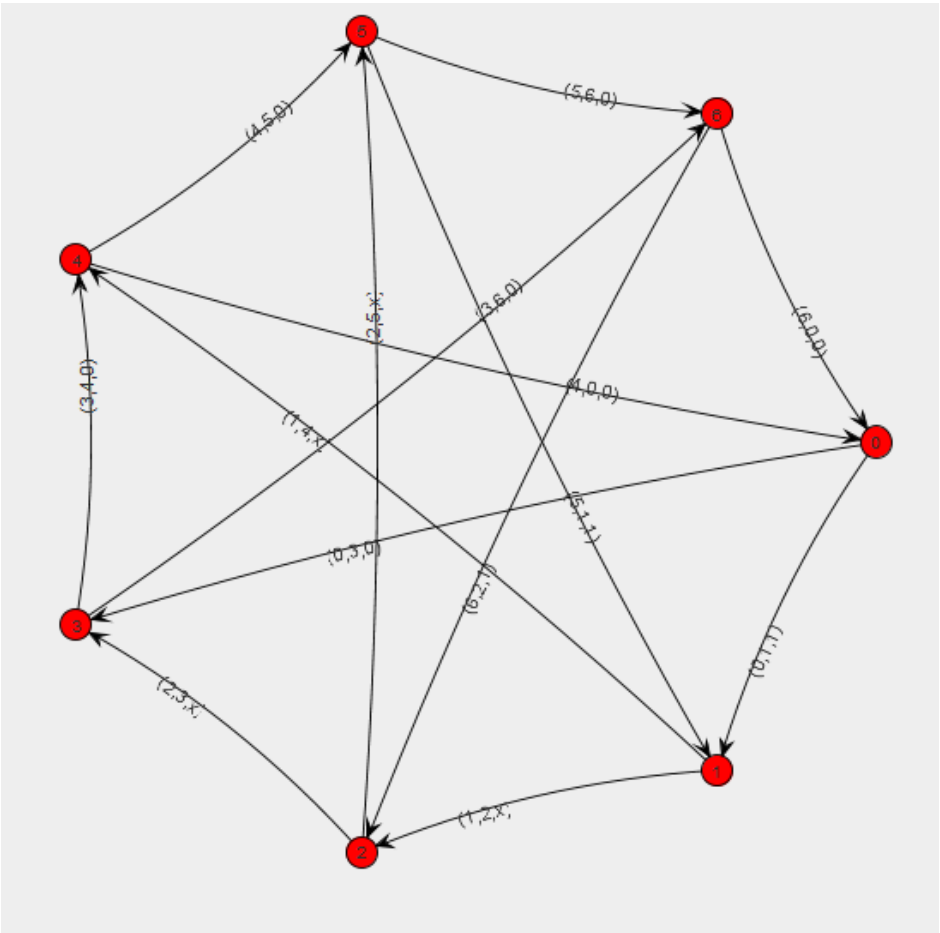


Рисунок 24 – Приклад ПМЧ моделі для багатопроцесорної системи

На рисунку 24 зображено приклад побудови ПМЧ-моделі для системи, що складається з 7-ми процесорів та максимально допустимим числом несправностей 2. Додані цикли через один і три процесора.

Для зручності було додано ще додаткові функції. На рисунку 25 зображено можливість виділення однієї або декількох вершин та їх відтягування. Це стало можливим завдяки використанню обробника подій на пристрої, такі як клавіатура та мишка. У звичайному стані вершини мають червоне забарвлення, а при виділенні – жовте. На кожній вершині вказано її порядковий номер, починаючи з нуля. Кожне ребро має вагу з трьома змінними. Перше і друге числа відповідно – це номера вершин, з якої виходить ребро та у яку входить. Третє число може набувати трьох значень: 0, 1 або x згідно з принципами побудови ПМЧ-моделі.

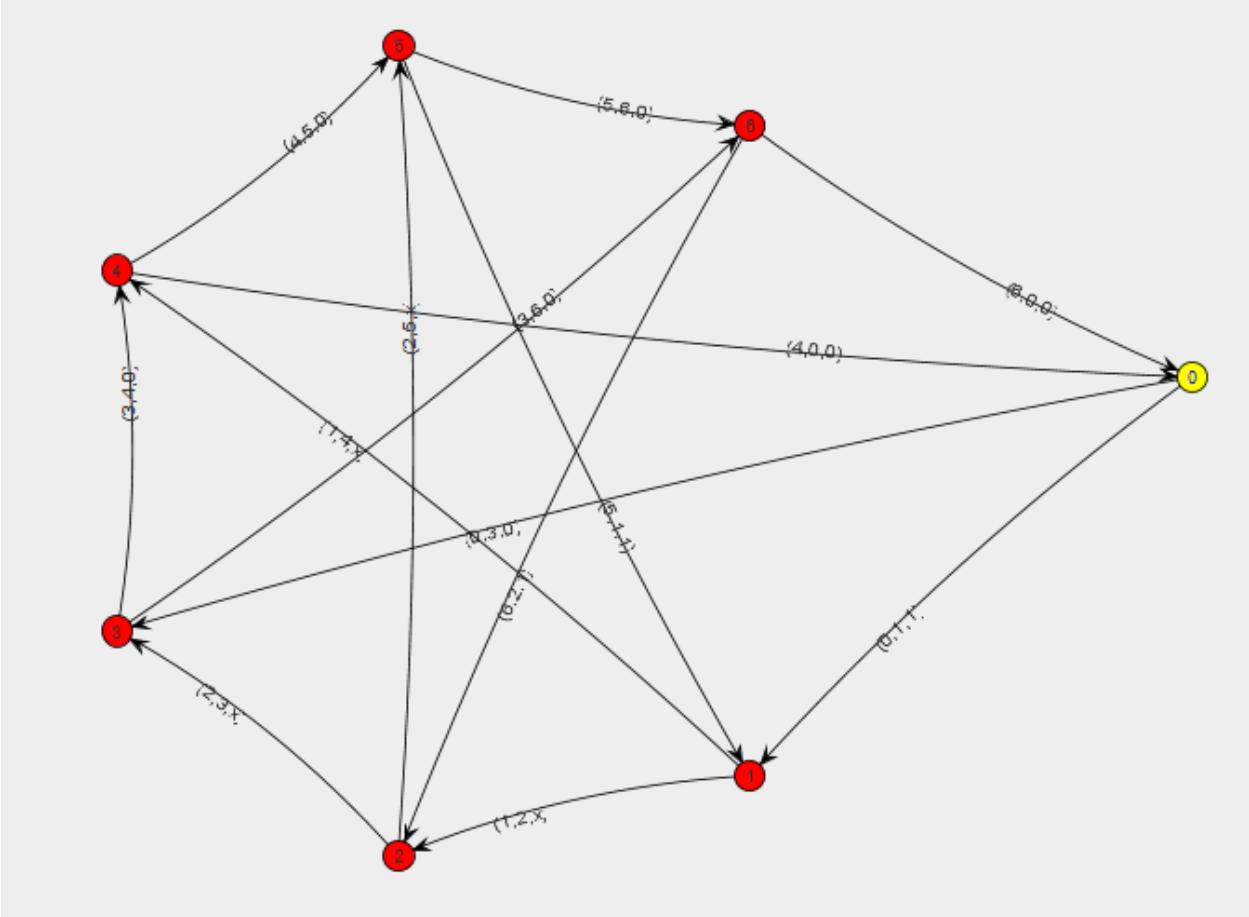


Рисунок 25 – Приклад роботи функціоналу програми

На рисунку 26 відображено ПМЧ-модель для багатопроцесорної системи, що складається з 20-ти процесорів, та з допустимим числом несправних процесорів 6. Для даного графа додано два цикли з відстанню 2 і 5 вершин.

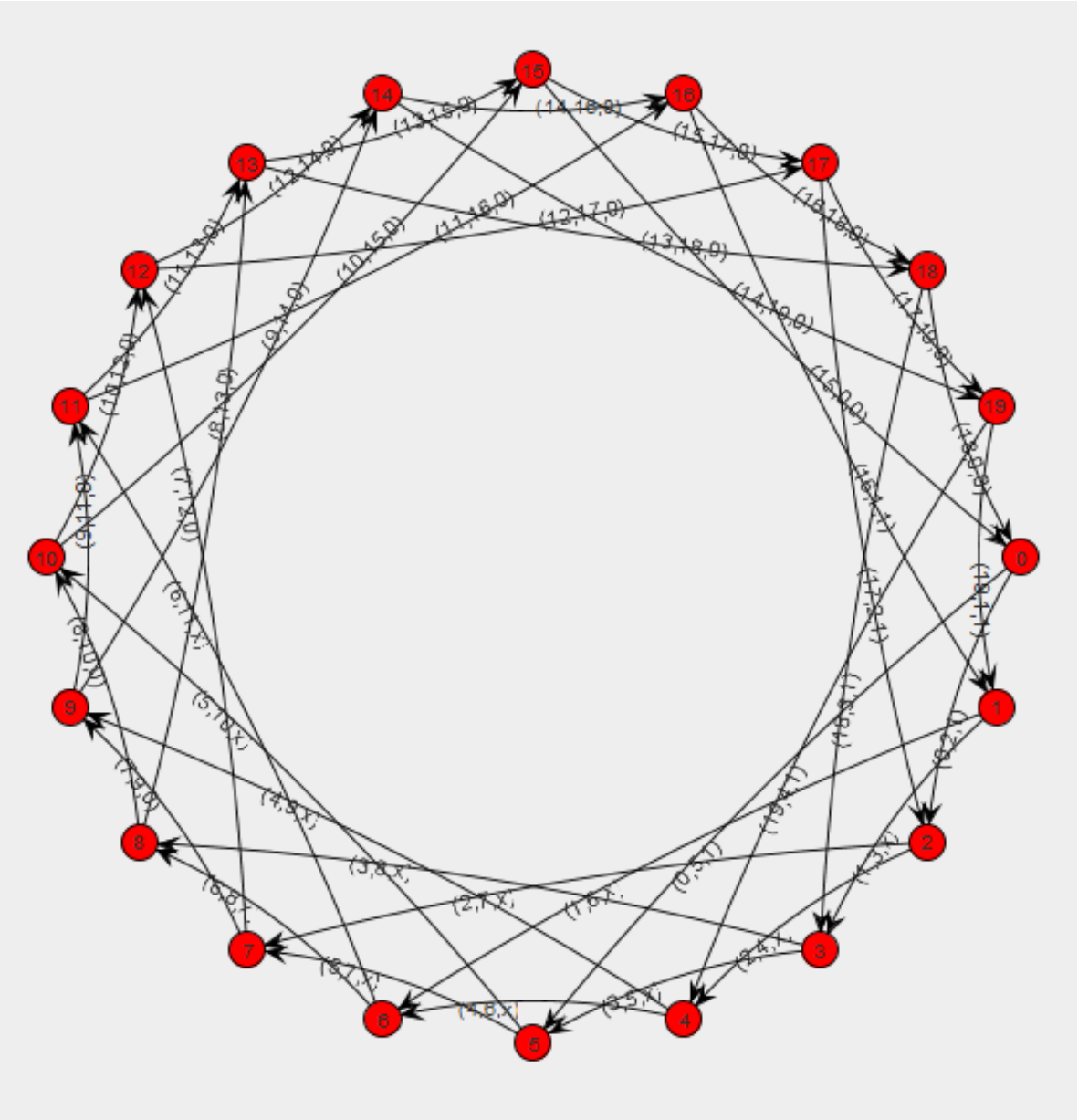


Рисунок 26 – Граф-циркулянт для системи з *n=20, m=6*

Використовуючи мишку можна змінити масштаб побудованого графа, що і показано на рисунку 27. Реалізовано можливість зміни місцезнаходження декількох вершин одразу. При необхідності можна перемістити повністю ввесь граф, виділивши всі вершини.

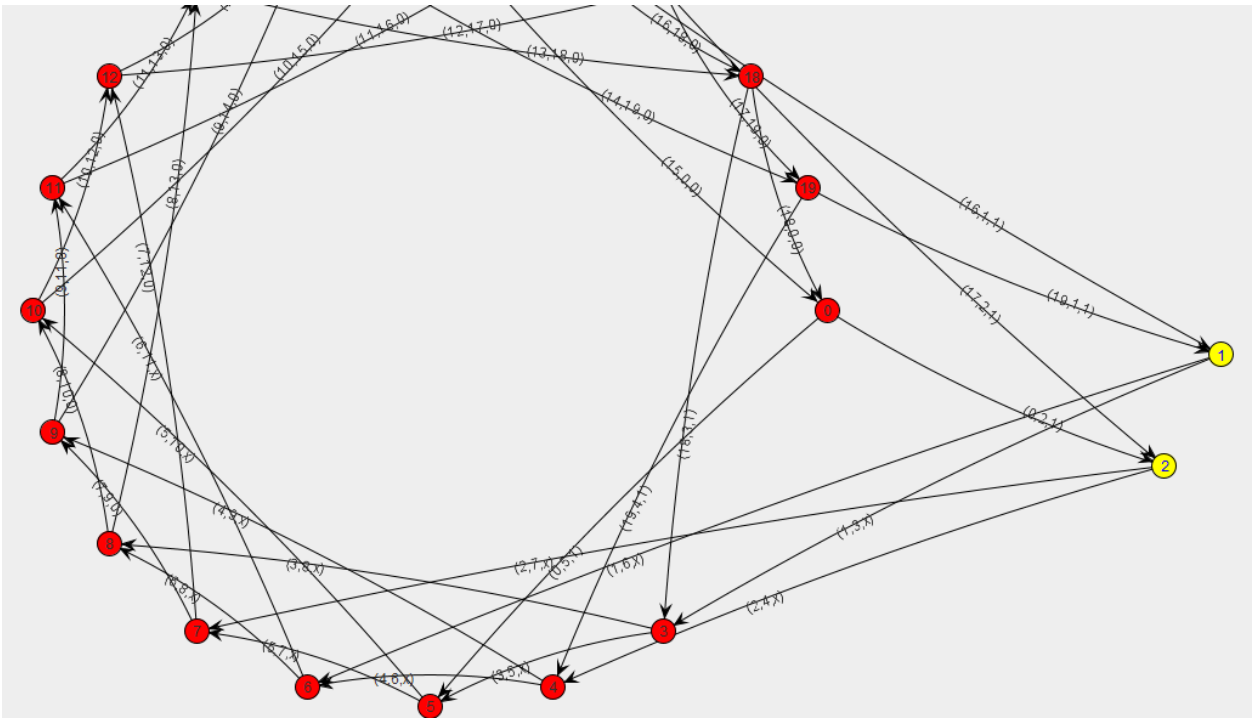


Рисунок 27 – Масштабування графа

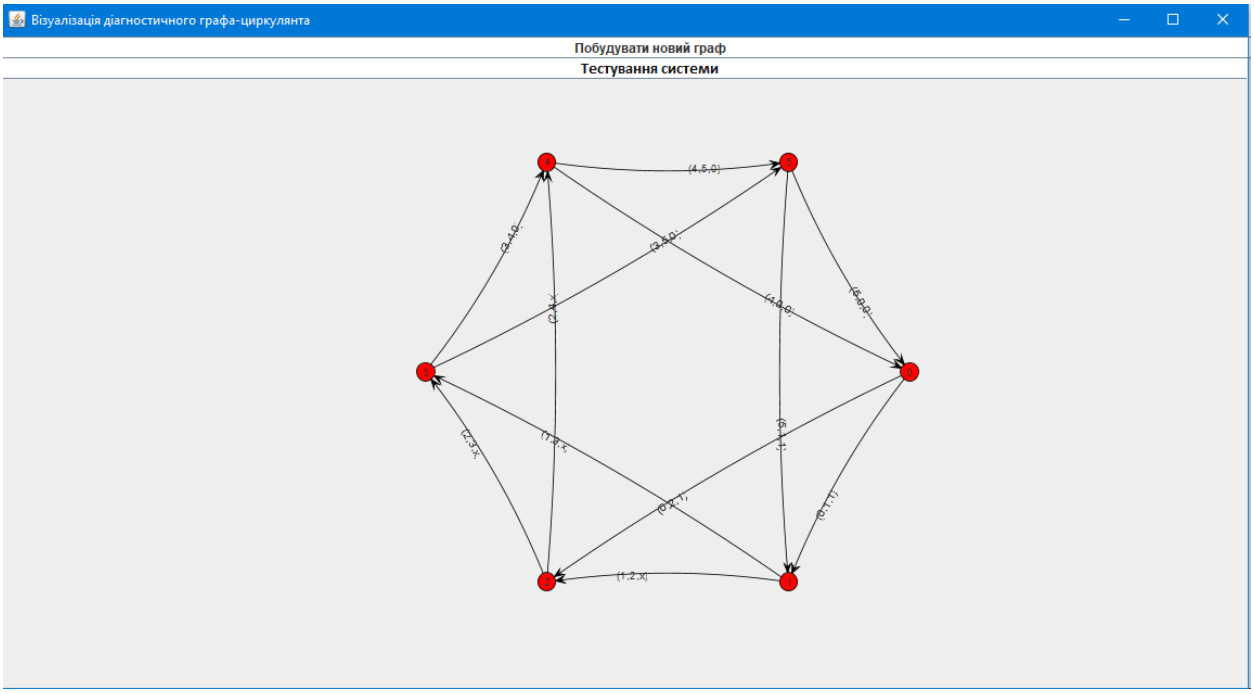


Рисунок 28 – Діагностичний граф з n=6, m=2

Зверху кожного вікна є дві кнопки. Перша з них будує новий граф-циркулянт. А при натисненні на кнопку «Тестування системи» розпочнеться робота алгоритму, результатом роботи якого є вектор з порядковими

номерами несправних процесорів. Для вище описаного графа результат роботи зображено на рисунку 29.

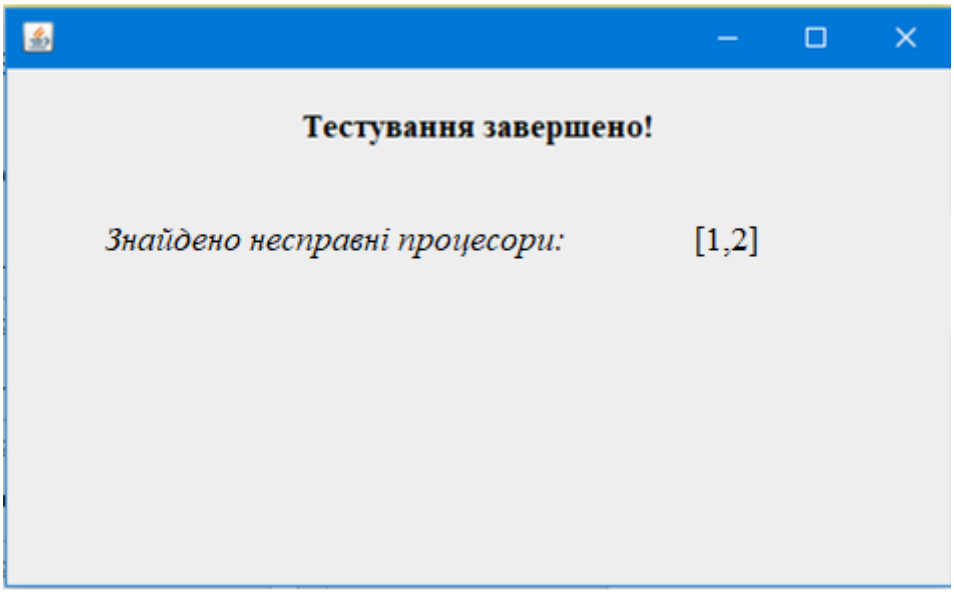


Рисунок 29 – Результат тестування системи K(6,2)

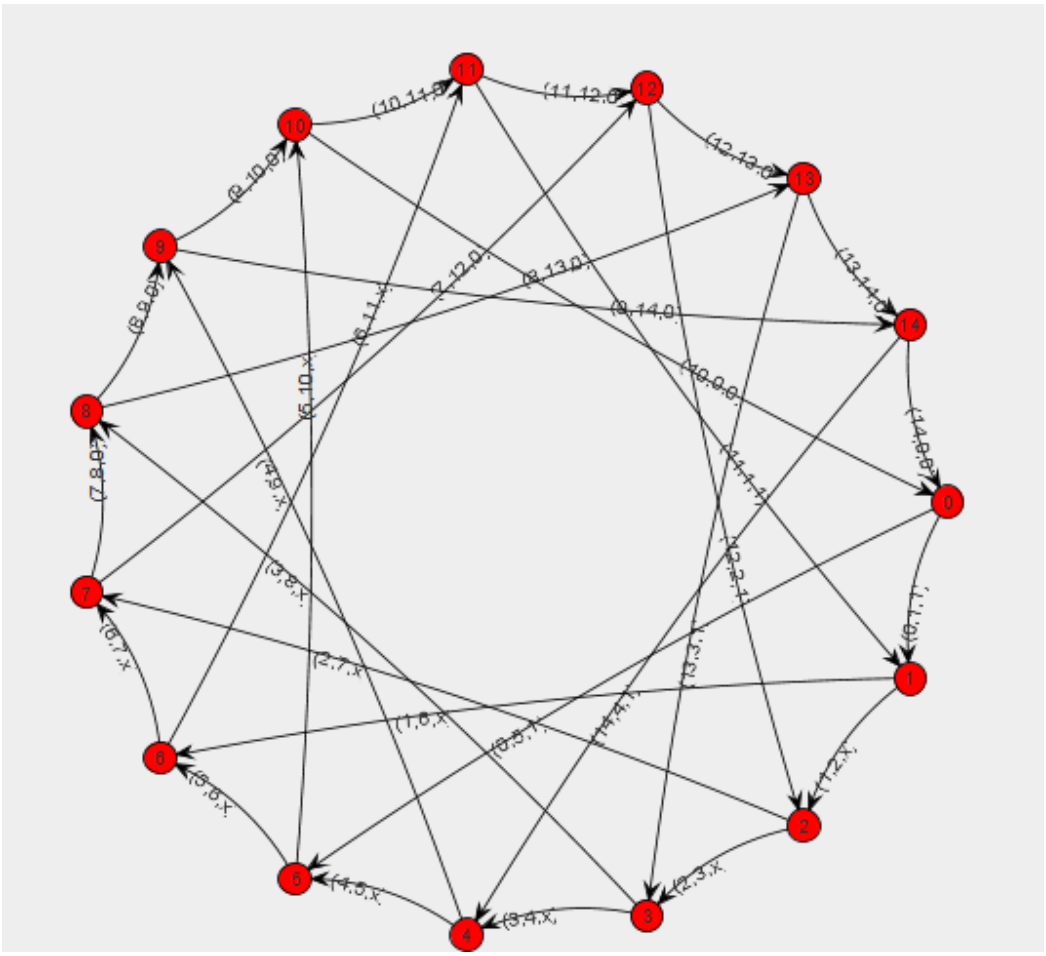


Рисунок 30 – Діагностичний граф з n=15, m=6

Проведемо тестування ще для однієї системи, діагностичний граф якої зображено на рисунку 30. Цикли були задані відстанню у 1 та 5. Результат тестування продемонстровано на рисунку 31.

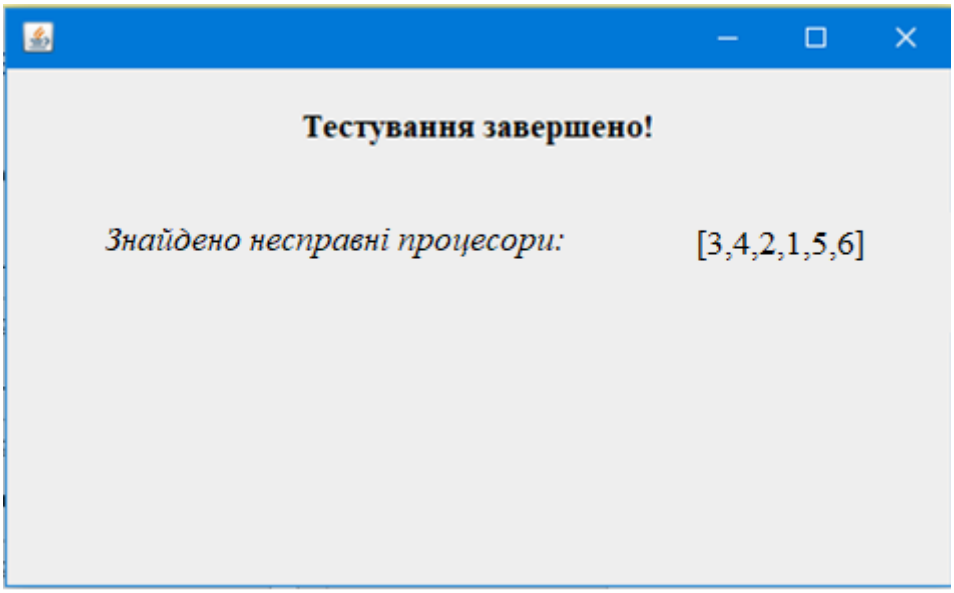


Рисунок 31 – Результат тестування системи K(15,6)

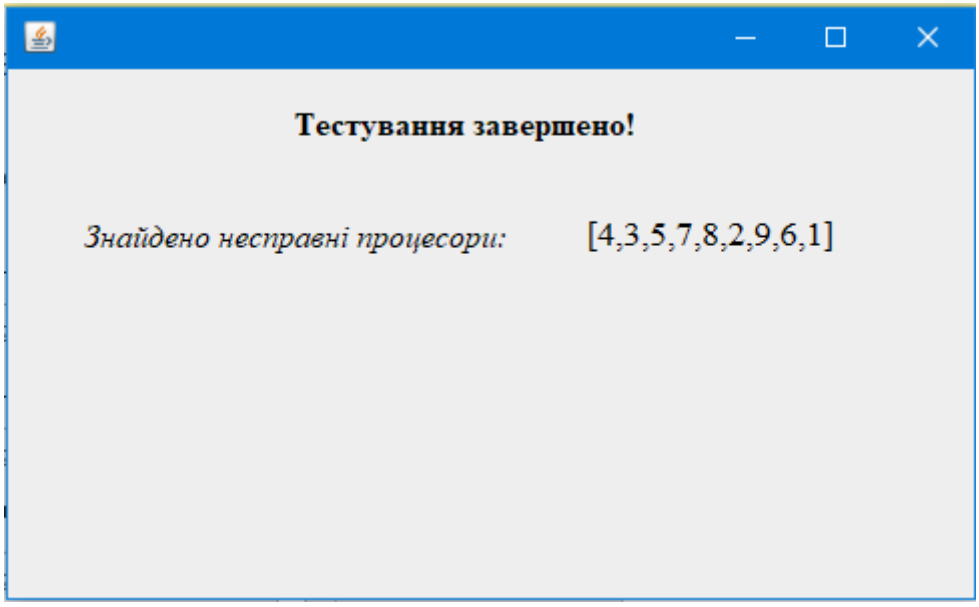


Рисунок 32 – Результат тестування системи K(25,9)

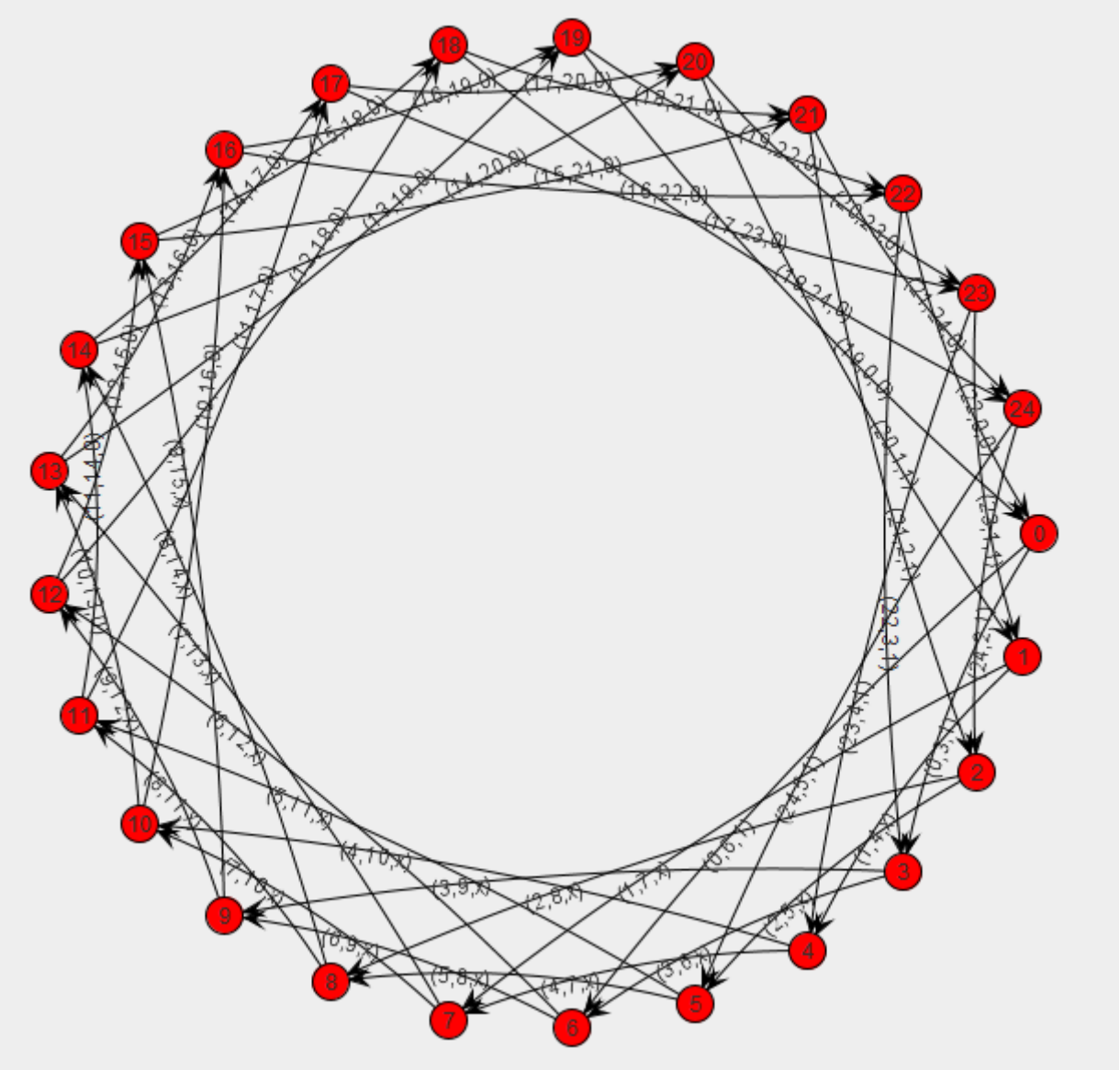


Рисунок 33 – Діагностичний граф з n=25, m=9

Наступним було побудовано граф для системи K(25,9) з стрибками циклу 3 і 6. Результат пошуку несправних процесорів відображено на рисунку 32, а сам граф на рис. 33.

І останній випадок розглянемо для таких параметрів: n=28, m=10 і стрибки циклу 4 і 5. Відповідний граф й результат тестування показано на рисунках 34 і 35.

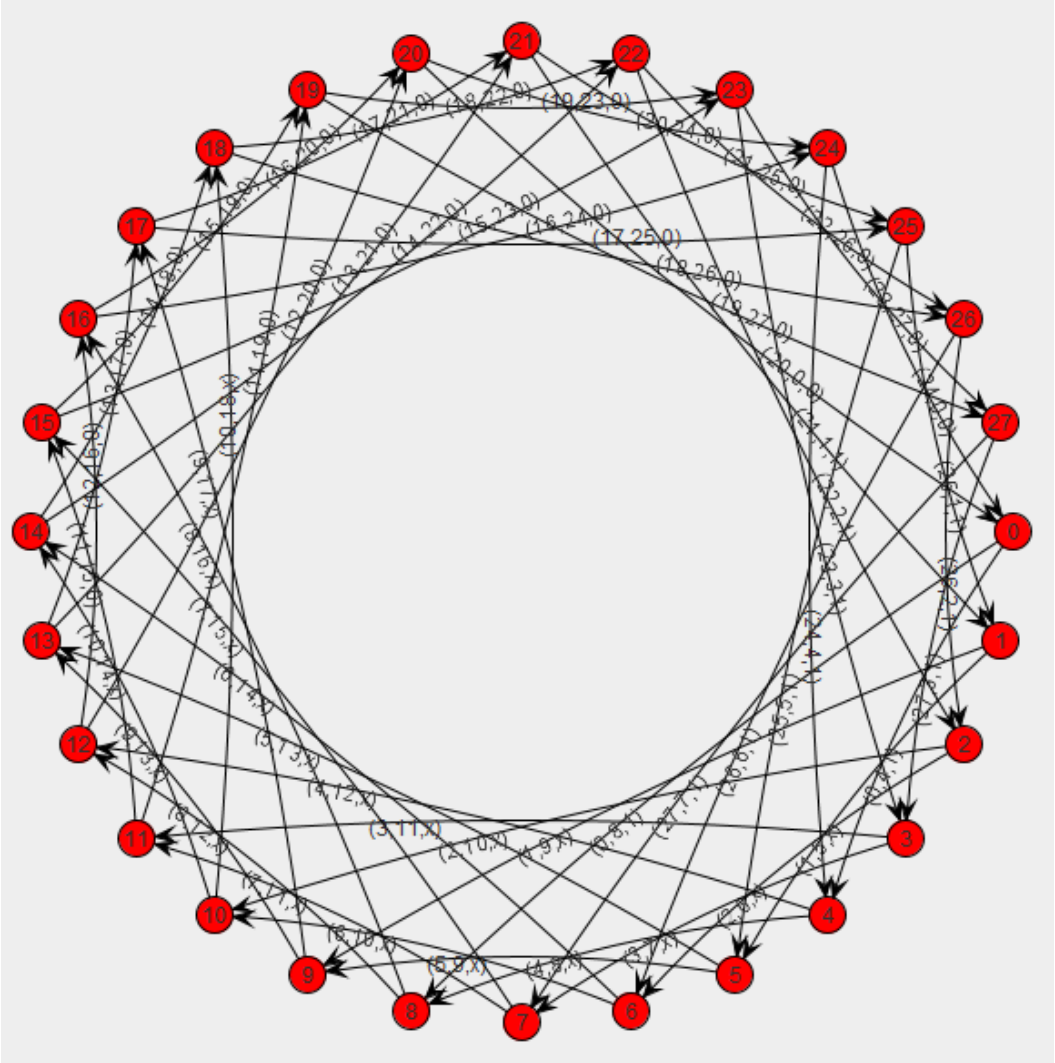


Рисунок 34 – Діагностичний граф з n=28, m=10

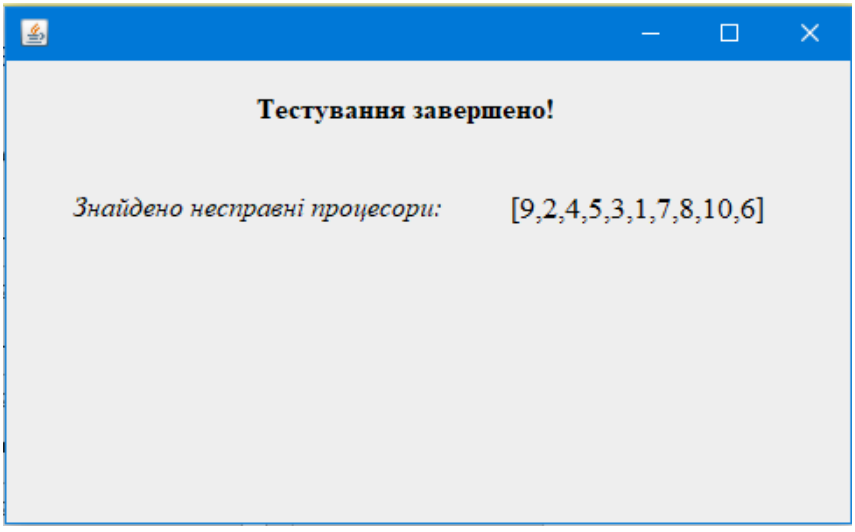


Рисунок 35 – Результат тестування системи K(28,10)

Висновки до розділу 3

Багатошинна структура складна, тому для побудови моделі необхідно розділити систему на підрівні. А потім побудувавши моделі для кожної одиниці окремо, необхідно їх об’єднати. Окремі моделі можуть обмінюватися одна з одною елементами у разі, якщо це необхідно для забезпечення роботоздатності системи в цілому.

Було розроблено програму, що забезпечує тестування багатопроцесорної системи на основі побудови ПМЧ-моделі. Також програма будує діагностичний граф-циркулянт. Процес взаємного тестування відбувається за допомогою пошуку 0-ланцюжків. Знайдені ланцюжки об’єднуються у дерево, після чого справний процесор розпочинає процес тестування.

ВИСНОВОК

У даній роботі було проведено аналіз існуючих рішень по організації тестування багатошинних систем. У результаті розроблено програму, однією з функцій якої є пошук «0-ланцюжків», для реалізації пошуку було написано алгоритм. На основі даних про наявність ланцюжків та їх довжину можна знайти справний процесор. Далі, маючи граф зв'язків між процесорами, розробник організує взаємотестування процесорів таким чином, щоб тестуючим був завжди справний процесор. Перевагами алгоритму є забезпечення можливості тестування систем з великою кількістю процесорів, адже в XXI столітті апаратні параметри дозволяють створювати системи з все більшим числом процесорів.

На сьогодні спостерігається підвищений інтерес до використання GL-моделей, особливо у випадках, коли складну систему можливо описати з використанням декількох базових моделей. Оскільки система є багатошинною, то для можливості побудови моделі систему було розділено на підсистеми так, щоб всередині кожної такої одиниці елементи були взаємозамінні, а зв'язок між рівнями здійснювався за рахунок окремо взятих моделей з урахуванням особливостей початкових умов, таких як роботоздатність кожної шини, а також наявність міжшинних зв'язків кожної пари шин. Використовуючи загальні теоретичні основи побудування графо-логічних моделей, що описують поведінку системи у потоці відмов, було побудовано модель для системи, що складається з трьох шин. Система включає 42 процесори, з яких на кожній шині по 10 процесорів, а між кожною парою шин по 4 процесори. При цьому мінімальними умовами роботоздатності системи є наявність по 8 справних процесорів на кожній шині й по 2 процесори між кожною парою шин. Якщо на шині кількість справних процесорів стає менше восьми, то дана шина може замінити недостатню кількість процесорів з міжшинних, якщо це можливо. Тобто підсистеми можуть використовувати міжшинні процесори, хоча вони

відносяться до різних підсистем. При цьому процесори, що були додані до шини, втрачають можливість повернутися на попереднє місце.

Графо-логічні моделі дозволяють розраховувати параметри надійності, що взагалі є досить непростою задачею стосовно відмовостійких багатопроцесорних систем.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Вычислительные машины, комплексы и сети: Учебник / С.Н. Беляев, Г.М. Козырева и др.; Под ред. А.П. Пятибратова. М.: Финансы и статистика, 2011. 344 с.
2. Гук Михаил. Аппаратные средства IBM PC: Энциклопедия / М.Гук. - 2-е изд. - СПб.: Питер, 2009. - 923с.
3. Иванова Е.М., Жарков С.В. Организация ЭВМ и систем: Учеб. пособие. – М.: МИЭМ, 2011. 147 с.
4. Компьютерные системы и сети: Учеб. пособие /В.П. Косарев и др. М.: Финансы и статистика, 2009. 431 с.
5. Питолин А.В, Даньшин И.Л., Крамаренко А.П. Программная реализация функций управления файловой и дисковой системами. Методические указания.
6. Питолин А.В. Организация взаимодействия устройств ЭВМ с использованием функций WIN32 API: Учеб. пособия. - Воронеж: ВГТУ, 2009.
7. Питолин А.В. Функционально-структурная организация ЭВМ и систем: Учеб. пособие. - Воронеж: ВГТУ, 2008. - 119 с.
8. Питолин А.В. Функционально-структурная организация ЭВМ и систем: Учеб. пособие. - Воронеж: ВГТУ, 2009. - 119 с.Гук М. Энциклопедия ПЭВМ. М.: Мир, 2010. 487 с.
9. Питолин А.В., Яскевич О.Г. Изучение работы с регистрами центрального процессора при помощи команд учебной ЭВМ. – Воронеж, ВГТУ, 2011. – 19.
10. Хамахер К., Вранешич З., Заки С. Организация ЭВМ. СПб.: Изд. «Питер», 2009, 541 с.
11. Микеладзе М.А. Развитие основных моделей самодиагностирования сложных технических систем //Автоматика и телемеханика.-1995.- №5.- С. 3-18.

12. Харченко В.С., Гридин Ю.В. Модель функционирования бортовых вычислительных систем с категорированием задач в условиях сбоев и отказов аппаратных и программных средств // Космічна наука і технологія.- 1999.- №5.- С.103-109.
13. Salam Salloum, Melvin A. Breuer. Fast Optimal Diagnosis Procedures for k-out-of-n:G Systems //IEEE Trans. on Reliability.- 1997.-Vol.46, №2.- P.283-290.
14. Chen R.W., Hwang F.K., Li Wen-Ching. Consecutive 2-out-of-n:F systems with node and link failures //IEEE Trans. on Reliability.- 1993.-Vol.42,№3.- P.497-502.
15. Романкевич О.М., Карачун Л.Ф., Романкевич В.О. До питання побудови моделі поведінки багатомодульних систем. // Наукові вісті НТУУ “КПІ”, 1998, №1,с.38-40.
16. Романкевич А.М., Карачун Л.Ф., Романкевич В.А. Графо-логические модели для анализа сложных отказоустойчивых вычислительных систем // ж. Электронное моделирование, 2001,т.23, №1, с.102-111.
17. Романкевич В.А., Потапова Е.Р., Ал Шбул Рабах Объединение моделей подсистем в рамках GL-моделей // Інформаційно-керуючі системи на залізничному транспорті. Тези доповідей.- 2002.- Додаток до №4,5.- С. 22.
18. Романкевич А.М., Орловский А.С. Л.Ф., Романкевич В.А. Алгоритм синтеза базовой GL-модели // Вісник НТУУ “КПІ”.- Інформатика, управління та ОТ.-2001.- №35.-С.132-135.
19. Харченко В.С., Жихарев В.Я., Илюшко В.М., Нечипорук Н.В. Многоверсионные системы, технологии, проекты // Харьков; Нац. Аэрокосм. ун-т "Харьк.авиац. ин-т", 2003. – 486с.
20. Райншке К., Ушаков И.А. Оценка надежности систем с использованием графов/ Под ред. И.А. Ушакова.– М.: Радио и связь, 1988.- –208с.
21. Романкевич А.М., Карачун Л.Ф., Романкевич В.А. Графо-логические модели для анализа сложных отказоустойчивых вычислительных систем // Электронное моделирование.- 2001.-т.23, №1.- С.102-111.

22. Романкевич В.А. Об одной модели поведения отказоустойчивой многопроцессорной системы // Радиоэлектроника и информатика.- Харьков.-1999.- №1.- С. 75-76.
23. Романкевич О.М, Романкевич В.О., Богуславський О.В., Ал Шбул Рабах. Аналіз відмовостійких багатопроцесорних систем на основі графо-логічних моделей нециклічного типу // Вісник ТУП, т. 2 "Технічні науки".- Хмельницький 2002.-С.30-33.
24. В.А. Романкевич, Рабах Мох'д Ахмад Ал Шбул, Назаренко В.В. О минимизации базовых циклических GL-моделей // Вісник ТУП, ч.1,т.2 "Технічні науки".- Хмельницький 2004.-С.42-46.
25. Романкевич А.М., Иванов В.В., Романкевич В.А. Анализ отказоустойчивых многомодульных систем со сложным распределением отказов на основе циклических GL-моделей // Электронное моделирование.-№5, т.26, 2004, с.67-81
26. S.V.Amari, H.Pham, and G. Dill Optimal Design of k-out-of-n:G Subsystems Subjected to Imperfect Fault-Coverage // IEEE Transaction on Reliability.- vol.53.- December 2004.- pp.566-575
27. H. Lui. Reliability of a load-sharing k-out-of-n:G system: Non-id components with arbitrary distributions //IEEE Trans. on Reliability.- 1998.-Vol.47, №3.- P.279-294.
28. Шилдт, Герберт Java 8. Руководство для начинающих / Герберт Шилдт. – М.: Вильямс, 2015. – 51 с.
29. Альфред, В. Ахо Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. – М.: Вильямс, 2015. – 460 с.