Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра комп'ютерної інженерії та системного програмування

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Мультикомп'ютерна система загального призначення на основі топології «тор»
Назва теми

КвРКІ.170140.17.01.09 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія»*
Назва

Виконав: студент IV курсу, група КІ–17–1 _____ О.М. Клейн
Підпис Ініціали, прізвище

Керівник _____ Д.М. Медзатий
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ С.М. Лисенко
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної
Інженерії та системного
Програмування _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«____» _червня_ 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем

Кафедра Комп'ютерної інженерії та системного програмування

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма освітня програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"11" 01 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Клейну Олександру Миколайовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Мультикомп'ютерна система загального призначення на основі топології «тор»

Керівник проекту (роботи) Медзатий Д.М., к.т.н., доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 05.02.2021 р. № 11

2. Строк подання студентом проекту (роботи) на кафедру 07.06.2021 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження предметної області та постановка задачі

Проектування мультикомп'ютерної системи загального призначення на основі топології «тор»

Програмно-апаратна реалізація та тестування мультикомп'ютерної системи загального призначення на основі топології «тор»

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Загальна архітектура SMP-системи

Процесорний блок

Модуль когерентності кеш-пам'яті

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прий
Нормоконтроль	Лисенко С.М., професор кафедри КІСП		
Антиплагіат	Нічепорук А.О., доцент кафедри КІСП		

7. Дата видачі завдання «_11_»_____01_____2021 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Приміт
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	11.01.2021	викона
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2021	викона
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2021	викона
4	Робота над розділом 2 – моделювання та проєктування мультикомп'ютерної системи	01.04.2021	викона
5	Робота над розділом 3 – програмно - апаратна реалізація мультикомп'ютерної системи	30.04.2021	викона
6	Оформлення пояснювальної записки згідно вимог	31.05.2021	викона
7	Попередній захист ВКР	02.06.2021	викона
8	Захист ВКР на засіданні ЕК	Червень 2021 року	

Студент _____ О.М. Клейн
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Д.М. Медзатий
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 170140.17.01.09 ПЗ	Пояснювальна записка	80		
			Графічні матеріали			
2		КвРКІ 170140.17.01.09 Е8	Менеджер процесів в			
			системі	1		
3		КвРКІ 170140.17.01.09 Е8	Граф-схема часових подій	1		
			в процесах системи	1		
4		КвРКІ 170140.17.01.09 Е8	Реалізація протоколу			
			в системі	1		

Зм	Ар к	№ докум	Підпис	Дата	КвРКІ 170140.17.01.09 ВП		
Розробив		Клейн			Відомість проекту	Літера: У	Аркуш: 1 / Аркушів: 1
Перевір.		Медзатий				ХНУ, КІ-17-1	
Н. контр.		Лисенко					
Затв.		Говорущенко					

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мультикомп'ютерна система загального призначення на основі топології тор».

Автор роботи: Клейн Олександр Миколайович.

Керівник роботи: Медзатий Дмитро Миколайович.

Пояснювальна записка: 80 с., 13 рис., 7 табл., 4 дод., 50 джерел.

Графічна частина: 10 презентаційних слайдів.

МУЛЬТИКОМП'ЮТЕРНА СИСТЕМА ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ НА ОСНОВІ ТОПОЛОГІЇ «ТОР»

Метою роботи є розробка мультикомп'ютерної системи загального призначення на основі топології тор.

У цій роботі була розроблена мультикомп'ютерна система загального призначення, на основі топології тор. Для розробки та моделювання роботи були використані такі компоненти як Quartus II, SOPC Builder та програмована плата Altera DE1-SoC. За основу системи було взято SMP-архітектуру, також у ході роботи було описано та запропоновано і частково розроблено рішення для усунення проблем «вузького місця», та когерентності кешу.

Підпис студента _____ Дата 23.06.2021

ЗМІСТ

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ОС – операційна система

ПЗ – програмне забезпечення

ЕОМ – електронно-обчислювальна машина

RISC – reduced instruction set computer

CPU – central processing unit (центральний процесор)

ЗСД – загальне сховище даних

МКК – модуль когерентності кеш-пам’яті

ISR – обробник переривань

АР – обчислювальні процесори

ВСТУП

Протягом усієї історії розвитку ЕОМ та обчислювальної техніки були спроби ввести загальну класифікацію, під яку попадали б усі комп'ютерні архітектури, але жодна з таких класифікацій так і не змогла охопити все різноманіття розроблювальних систем і не пройшла випробування часом. Проте ці спроби класифікації дали науці ряд широковикористовуваних термінів, які застосовуються і досі.

Будь яка ЕОМ, будь то суперкомп'ютерна система що використовується для надскладних обрахунків в сферах науки, чи звичний нам персональний комп'ютер, досягає найвищої продуктивності не лише завдяки використанню високошвидкісних елементів, а й можливості виконувати велике число операцій паралельно. І саме можливість паралельного виконання задач є основною, коли мова йде про пришвидшення роботи комп'ютера.

В сучасному світі актуальність цієї роботи дуже висока, адже сучасні технології, наука, розвиток суспільства ставить обширні та складні задачі, вирішення яких на однопроцесорних системах вимагає значних затрат часу. Сфера використання мультипроцесорних систем постійно розширюється і захоплює усе нові сфери науки, бізнесу, економіки та виробництва.

Саме тому однією з найважливіших тенденцій розвитку обчислювальної техніки полягає в активному запровадженні мультипроцесорних систем, та побудові цілих обчислювальних комплексів.

З появою багатоядерних процесорів, такі системи стали можливі навіть для використання в звичайних домашніх комп'ютерах, ноутбуках і уже навіть мобільних пристроях, хоча б з невеликою мультипроцесорною системою.

З'єднання великої кількості процесорів в одну єдину систему дозволяє досягти потужності, що в рази перевищує максимальну швидкодію класичних ЕОМ, по схемі фон Неймана.

Відповідно до того, як саме відбувається об'єднання процесорів у єдину систему і які ресурси використовуються, такі системи називаються паралельними комп'ютерами, мультикомп'ютерами або мультипроцесорними системами.

Такі ЕОМ потребують спеціальних операційних систем, в якості яких часто використовуються відомі ОС, але модифіковані, шляхом додавання до них спеціальних функцій зв’язку та синхронізації, а також більш скурпульозного підходу в програмуванні, адже саме програмне забезпечення дозволяє розкрити потужність таких систем на максимум.

Метою цієї роботи є створення саме мультипроцесорної системи загального призначення на основі топології «гіперкуб». У роботі буде проаналізовано різні архітектури, що можуть використовуватись в системах з загальною пам’яттю, проведено аналіз наявних архітектурних рішень. У другому та третьому розділах буде описано проектування та реалізацію такої системи з детальним обгрунтуванням вибору та застосування апаратно-програмної бази.

1 ЗАСОБИ ПІДТРИМКИ ІНТЕГРАЦІЇ В РОЗПОДІЛЕНИХ СИСТЕМАХ

1.1 Поняття про засоби підтримки розподілених систем

Проміжне програмне забезпечення – це програмний засіб, який підтримує взаємодію між компонентами в розподіленому обчислювальному середовищі. Проміжне програмне забезпечення надає набір послуг, що дозволяє клієнтським і серверних програмам спілкуватися один з одним через мережі та різні комп'ютерні системи. Проміжне програмне забезпечення забезпечує засіб розділення функціональності, щоб забезпечити незалежну розробку і розгортання програмних компонентів, які підтримують презентацію, бізнес-логіку і доступ до послуг, як правило, доступ до баз даних, для конкретного застосунку. Проміжне програмне забезпечення знаходиться між програмами та базовою операційною системою та мережею. На рис. 1.1 зображено місце проміжного програмного забезпечення.

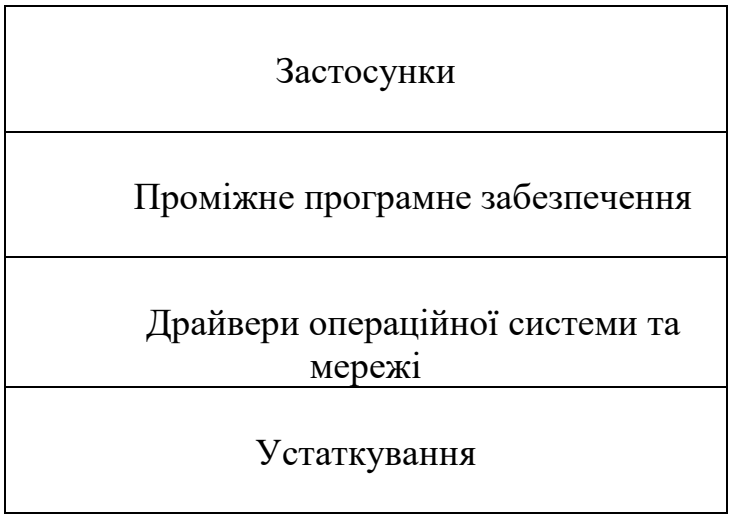

Рисунок 1.1 - Контекст проміжного програмного забезпечення

З моменту свого появи в 1980-х роках, асортимент проміжного програмного забезпечення постійно розширювався і вдосконалювався. Основними типами проміжного програмного забезпечення є синхронне процедурне RPC (Remote Procedure Call), орієнтоване на проміжне програмне забезпечення, таке як DCE-

RPC; асинхронні mom (орієнтовані на повідомлення-проміжне програмне забезпечення) продукти, такі як MQ-series IBM; і орієнтоване на транзакції проміжного програмного забезпечення, включаючи TUXEDO BEA і CICS IBM . Широко використовується об'єктна середня CORBA від OMG [OMG, 2000], DCOM корпорації Майкрософт та Java/RMI. Зовсім недавно сервіс-орієнтовані архітектури, такі як веб-сервіси Microsoft (.Net) і Java, породило проміжне програмне забезпечення на основі XML/ SOAP.

У деяких випадках проміжне програмне забезпечення базується на відкритих або узгоджених стандартах, таких як CORBA та XML/SOAP, а інші, наприклад, DCOM, TUXEDO та CICS є власними. Часто організації мають власні системи, які роблять роботу сторонніх посередників. В результаті незалежної розробки продукту постачальника, системи, засновані на одному з цих методів, як правило, не сумісні безпосередньо з системами на основі іншого. Важко викликати сервер DCOM з клієнта CORBA, хоча обидві системи засновані на об'єктній моделі. Реалізації цих об'єктних систем досить різні, і клієнт CORBA не сумісний з сервером DCOM, навіть якщо обидва працюють на платформі Windows (Intel). Необхідність системної інтеграції виникла через динаміку бізнесу та технологій, оскільки бізнес розширює, об'єднує або приєднується до партнерств чи альянсів. Впровадження нових технологій, таких як веб-технології, корпоративні обчислення та електронна комерція, сприяють створенню нових бізнес-можливостей. Деякі організації, особливо великі різноманітні організації, мають кілька продуктів і підходів проміжного програмного забезпечення. Різні частини організації, можливо, адаптували різні проміжні програми, щоб найкращими задовольнити свої потреби в минулому. У деяких випадках може бути бажано зберегти застарілу систему, або витрати на заміну можуть бути заборонені. Компанії беруть на себе або разом з іншими компаніями, і малим компаніям все частіше доводилося приєднуватися до глобальних мереж, щоб конкурувати локально (наприклад, «малі компанії, глобальні мережі»). Отримані супер-організації, як правило, включають суміш загалом несумісних IT-систем, які вимагають інтеграції якомога швидше для використання нових бізнес-структур. Проблема інтеграції систем на рівні протоколу з'єднана, якщо обидві

компанії використовують різні середовища операційної системи. Можливість підтримувати існуюче проміжне програмне забезпечення та застосунки призведе до значної економії витрат в контексті нових вимог або розгортання нових застосунків та проміжного програмного забезпечення.

Організації постійно оновлюють свою IT-інфраструктуру. Нові підходи до проміжного програмного забезпечення, наприклад, веб-служби XML / SOAP обіцяють довгострокові переваги організації, і бажано буде інтегрувати нові підходи в існуюче середовище. Інтеграція проміжного програмного забезпечення є значною проблемою для цих, як правило, великих організацій. Повідомляється, що інтеграція застарілих систем на рівні протоколу з іншими системами є серйозною проблемою без очевидних загальних рішень. Інтеграція низько рівних систем є складною, оскільки необхідно вирішувати семантику застосунків і часто потрібно низько рівне ручне налагодження даних. Vawter і Roman підтримують це в області поточних веб-систем, заявляючи, що «інтеграція спадщини часто є найскладнішим (якщо не найскладнішим) завданням, яке потрібно подолати при побудові веб-сервісу».

Поточна інтеграція проміжного програмного забезпечення вимагає значної кількості програмування. Кодування або на замовлення організації за значні кошти, або надається поза полицею постачальником, як "роз'єм для проміжного програмного забезпечення". В обох випадках рішення є одноразовим і обмежується конкретною проблемою інтеграції проміжного програмного забезпечення. Ця теза описує вдосконалений механізм інтеграції широкого спектру проміжніх програм. Мета полягає в тому, щоб дозволити організаціям легко мати клієнта, написаний для одного протоколу проміжного пз (наприклад, клієнт CORBA), отримати доступ до сервера, реалізованого в іншому, наприклад, веб-служби, клієнта .Net або навіть застарілої системної служби. Цей підхід продовжить термін служби клієнтського та серверного програмного забезпечення та дозволить організаціям легше приймати нове проміжне програмне забезпечення, зберігаючи існуючу інфраструктуру та застосунки. Він робить це, надаючи декларативну мову визначення протоколу проміжного програмного забезпечення (MPDL), яка описує проміжне програмне забезпечення, і рушій часу

виконання під назвою The Ubiquitous Broker Environment (TUBE), який приймає складені описи протоколів та виконує запити в різних протоколах проміжного ПЗ. Цей проект вирішує два широких питання, спрямовані на забезпечення основи загального та вдосконаленого підходу до інтеграції проміжного програмного забезпечення, заснованого на описі протоколів, а не програмування. Це дослідження дає докази того, що така мова і рушій можуть бути розроблені, для здійснення всіх основних поточних інтеграції проміжного програмного забезпечення. Такий підхід буде мати справу з новими проміжним програмним забезпеченням, без модифікації або з відносно простими модифікації.

1.2 Проміжне програмне забезпечення

Ядром функціональності проміжного програмного забезпечення є підтримка клієнт-серверної парадигми комп'ютерних обчислень, де застосунки базуються на програмі (клієнті), що робить запит іншої програми (сервера або служби), і сервера, що надає відповідь на цей запит. Клієнт і сервер незалежні і підключені тільки через інтерфейс сервера. Явне зв'язування або прив'язування певного сервера до клієнтського запиту зазвичай виконується під час виконання і є частиною ролі проміжного програмного забезпечення.

Клієнтські обчислення сервера дозволяють створювати програми в ряді незалежних шарів кожен з рядом різних компонентів програмного забезпечення, можливо, що знаходяться на різних платформах. Ця притаманна гнучкість має перевагу в тому, що дозволяє змінювати різні компоненти без компіляції компонентів, що не впливають.

У клієнт-серверних системах проміжний рівень виконує наступні основні завдання високого рівня. Він повинен налаштувати взаємодію і провести необхідну ініціалізацію. Він повинен керувати сеансом взаємодії, включаючи відправку повідомлень, отримання повідомлень, моніторинг з'єднань та обробку помилок та відновлення. Надсилання та отримання повідомлень вимагає перетворення повідомлень (запитів клієнтів та відповідей на сервери) у формати, які можна передавати через мережі та операційні системи на цільові сервери.

Виклики проміжного програмного забезпечення можуть бути здійснені проксі-сервером. Як правило, проміжне програмне забезпечення використовує IDL для полегшення прив'язки часу виконання до серверів. IDL визначає інтерфейси серверів і дані, що обмінюються між клієнтом і сервером. Перш, ніж цей обмін може відбутися, посередник повинен визначити адресу сервера (тобто, як зв'язатися з сервером). Це те, що називається кінцевою точкою. Кінцева точка складається з специфічної для протоколу інформації, такої як IP-адреса та номер порту брокера з запитів на об'єкт (ORB). Кожен тип проміжного пз має свою власну схему визначення, і, або збереження цієї інформації адресування. Ця адреса може бути, наприклад, IOR (Inter-operable Object Reference) для сервера CORBA, або визначення черги для MQ-series. Це надає необхідну інформацію для надсилання повідомлення або зв'язку з визначеним інтерфейсом за допомогою певного протоколу. Посередник повинен зберігати ці визначення проти кожного інтерфейсу, визначеного для конкретного протоколу проміжного програмного забезпечення. Це полегшує можливість змінювати конфігурації проміжного програмного забезпечення на основі інтерфейсу. Немає необхідності виконувати всі дії за допомогою одного типу проміжного ПЗ. З цього випливає, що якщо клієнт може отримати доступ до послуги в різних проміжних програмах, то може бути більше одного впровадження послуги, доступної через різні проміжні програми. Звичайне проміжне програмне забезпечення зазнає невдачі, якщо воно не може виконати запит, навіть якщо альтернатива може задовольнити запит. Проміжне програмне забезпечення визначає одного постачальника послуг і маршалів виклику належним чином, і onus знаходиться на розробнику програми для підтримки будь-яких невдач. Проміжне програмне забезпечення розуміє лише власні формати повідомлень і схеми адресування. Наприклад, за замовчуванням може бути CORBA, і виклики будуть націлені на кінцеві точки CORBA (наприклад, сумісне посилання на об'єкт); однак, доступна альтернатива може бути доступна через MQ-Series. Не буде спроб отримати доступ до цієї служби, якщо запит на обслуговування CORBA. Це пов'язано з тим, що проміжне програмне забезпечення CORBA не знає конфігурації MQ і не знає, як зв'язатися з кінцевою точкою; в цьому випадку черга MQ-Series. Можливість динамічно

змінювати проміжне програмне забезпечення надає можливість доступу до різних серверів, щоб задовольнити один і той же запит.

Проміжне програмне забезпечення базується на двох основних моделях: моделі віддаленого виклику процедур (RPC) та орієнтованої на повідомлення моделі проміжного програмного забезпечення (MOM). Модель RPC є моделлю дистанційного виклику процедури, в якій є абстрагування розташування сервера та способу зв'язку з клієнтською програмою. Клієнт використовує локальний модуль заглушки для обміну даними із сервером. Клієнт не знає про місцезнаходження серверів; він може бути локальним (перебуває на одному комп'ютері) або віддаленим. Заглушка містить механізм, необхідний для маршальських і немотиваційних параметрів для запитів і відповідей і зв'язку з сервером. Заглушка виробляється компілятором IDL (Interface Definition Language) і тісно пов'язана з бібліотеками часу виконання RPC. У цьому синхронному режимі взаємодії клієнт очікує відповіді від сервера в обмежений проміжок часу. Якщо відповідь не повернуто протягом указаного періоду, виникне час очікування. Модель RPC аналогічна лінійній телефонній системі, в тому, що сторони залишаються підключеними протягом усього терміну обміну. На схемі рис. 1.2 зображено базову модель RPC.

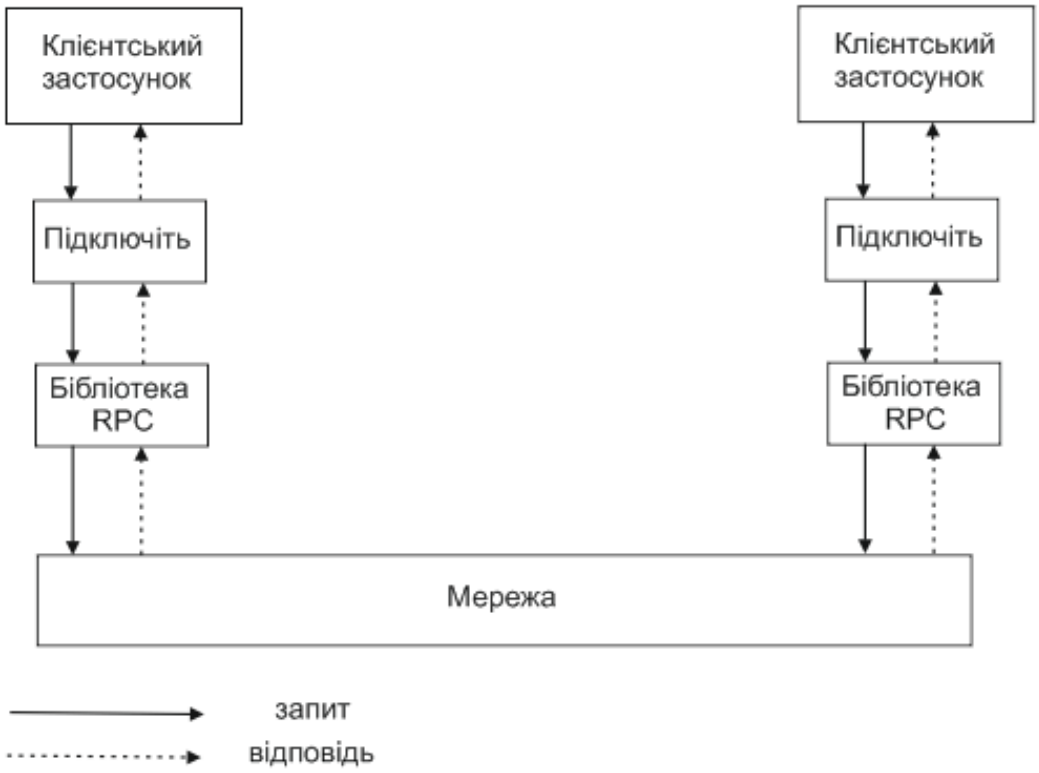

Рисунок 2.1 - Модель дистанційного виклику процедури

Це нове об'єктне проміжне програмне забезпечення все ще слідує цій базовій моделі. CORBA, COM і Java RMI використовують компілятори IDL для створення клієнтських заглушок і надають бібліотеки підтримки виконання, які виконують фактичний зв'язок. Орієнтована на повідомлення модель проміжного програмного забезпечення (MOM) на відміну від синхронної моделі на основі RPC, орієнтована на повідомлення модель (як правило) не використовує компілятор IDL для створення заглушок. Застосунки здійснюють дзвінки до конкретних модулів API для здійснення взаємодії. Це передбачає розміщення повідомлень у чергах та моніторинг черг для повідомлень відповідей. Ця модель асинхронна, а це означає, що відповідь може надходити в будь-який час. Як і для моделі RPC, час очікування може бути вказаний, хоча вони, як правило, мають більшу тривалість. Модель MOM аналогічна вітчизняній поштовій системі, де лист відправляється з одного місця в інше і відповідь може або не може прийти. Це відповідальність особи, яка розмістила лист для моніторингу та подальшого спостереження за будь-якою очікуваною відповіддю, якщо вона не надійде. У

моделі МОМ це відповідальність програми. Розширенням моделі МОМ є метод публікації-підписки. Це забезпечує можливість однієї заявки (абонента) зареєструвати інтерес до (підписатися) послуг, пропонованих іншою заявою (видавцем). Абонент може підписатися на події, що представляють інтерес за предметом. Наприклад, застосунок може зацікавитися, коли ціна певних акцій зміниться; ця програма підпишеться на таку тему, як "ціна-рух". Цей тип взаємодії використовується в сучасних системах EAI/workflow, таких як TIBCO Businessworks. Базовий протокол TIBCO Rendezvous базується на обміні повідомленнями на основі теми.

Зазвичай проміжне програмне забезпечення працює з повідомленнями у певних форматах, як текстових, так і двійкових. Текстові протоколи включають XML, HTTP і SOAP. Двійкові протоколи включають об'єктні протоколи (наприклад, CORBA, COM і Java-RMI) та інші, такі як DCE-RPC, які не є об'єктно-об'єктами. Існують також орієнтовані на повідомлення протоколи проміжного програмного забезпечення (MOM), такі як MQ-серія та JMS. Кожен протокол інкапсулює фактичний вміст повідомлення різними способами. SOAP, наприклад, загортає вміст у структуру, яка називається тілом SOAP, а потім загортає його в іншу структуру, відому як SOAP-конверт. Конверт також має необов'язкову структуру, яка називається заголовком. Вміст повідомлення, згаданий вище, є специфічним для програми текстом повідомлення, визначеним для інтерфейсу.

IDL визначає формат і типи даних параметрів, що передаються між клієнтами і серверами. Сумісність із проміжним програмним забезпеченням повинна мати можливість перетворювати повідомлення між різними форматами проміжного програмного забезпечення або надавати метод, який спрямовує цільовий інтерфейс у потрібний протокол, заснований на різних визначеннях проміжного програмного забезпечення. Дані користувача, що знаходиться в тілі повідомлення і формують параметри виклику, можуть бути комбінацією native-типів або складних типів, оголошених і визначених для кожної програми. Існує обмежена кількість власних (власних) типів даних: цілочисельний, довгий, короткий, символьний/байтовий, подвійний та бітовий. Всі інші типи даних

побудовані з цих фундаментальних типів. Після того, як буде визначено механізм зчитування і запису цих типів, можуть бути оброблені структури будь-якої складності. Протокол визначає правила і механізми читання і запису цих основних типів. Після цього ці правила оброблятимуть усі повідомлення за цим протоколом. Наприклад, CORBA використовує кодування, відоме як CDR (Загальне представлення даних) для читання та запису основних типів даних. Як тільки визначено правила CDR або бібліотеку, що піддається виклику, яка реалізує правила CDR, можна обробляти повідомлення на основі CDR (CORBA).

Загалом, повідомлення між клієнтами та серверами, незалежно від режиму зв'язку, складаються з двох типів даних, внутрішніх даних та корисного навантаження. Внутрішні дані надають необхідну інформацію для пошуку запитуваної послуги та відправки відповідей назад до запитаювача. Корисне навантаження - це специфічні для програми дані, необхідні для виконання функціональності програми. Внутрішні (контрольні) дані можна додатково розділити на загальні дані, тобто дані, які потрібні для всіх протоколів проміжного ПЗ, і специфічні для протоколу дані. Загальні дані можуть містити різні значення в залежності від протоколу, або з'являтися в різних місцях повідомлення в різних протоколах; однак вони потрібні для будь-якого обміну повідомленнями незалежно від протоколу. Ці змінні відстежують такі речі (серед інших), як; довжина повідомлень, порядковий номер, і чи має система справу з запитом або відповіддю. Ці змінні детально описані в главі 4. У наступному обговоренні ці обов'язкові змінні називаються загальними внутрішніми змінними проміжного пз. З іншого боку, специфічні для протоколу дані є змінними, які мають значення лише для певного протоколу. Наприклад, об'єкт CORBA, який ідентифікує об'єкт для створення екземпляра (або виклику) в кінцевому кінці запиту CORBA.

Проміжне програмне забезпечення підтримує два основних режими взаємодії або зв'язку, синхронний і асинхронний. За допомогою синхронного зв'язку клієнтська програма робить запит серверної програми, а потім чекає відповіді. Клієнт залишається заблокованим або в стані очікування, поки не отримає відповідь від запитаної служби. Прикладами проміжного ПЗ, які

підтримують цей режим взаємодії, є ті, що базуються на віддалених викликах процедур (RPC), і включають CORBA, COM, Java-RMI, SOAP (веб-служби) та DCE-RPC. При асинхронному спілкуванні клієнт розміщує повідомлення в шині повідомлень (або черзі) якийсь і відновлює обробку. Запит може потребувати відповіді, але клієнт може продовжувати виконувати інші завдання, очікуючи відповіді. Такий підхід набирав протоколи проміжного програмного забезпечення \(MOM), орієнтовані на повідомлення, такі як MQ-series та JMS.

Як правило, взаємодія обмежується одним режимом. Справжня сумісність між протоколами проміжного ПЗ означатиме, що виклики повинні бути зроблені не тільки в однакових режимах, але і в середньому посередині, використовуючи різні режими взаємодії, в деяких випадках режими, які безпосередньо не підтримуються цим типом проміжного ПЗ. Розглянемо наступний приклад. Клієнт знає лише, як зробити строго синхронний запит на сервері, S1 за допомогою деякого синхронного протоколу. Припустимо, що організація представляє новий асинхронний продукт проміжного та серверу S2, і бажає, щоб клієнт мав доступ до цього нового сервера. Клієнт не знає, що реалізацію сервера було змінено на використання асинхронної черзі. Синхронний сеанс повинен бути проведений з клієнтом, який чекає відповіді і, таким чином, заблокований. У той же час, черга на стороні сервера повинна контролюватися, поки відповідь, яка може прийти в будь-який час. Коли надходить відповідь, вона надсилається клієнту, що чекає. В ідеалі це потрібно робити без зміни або компіляції клієнта.

У синхронному та асинхронному режимах взаємодії існує два підтипи: розмовний та транзакційний. У розмовному режимі клієнт і сервер мають "розмову", тобто взаємодія складається з декількох типів обміну повідомленнями. Як правило, в цьому типі обміну клієнт, сервер або іноді обидва повинні підтримувати стан обміну. Один або обидва повинні знати, до якого моменту взаємодії вони до у випадку, якщо обмін несподівано припинено. У транзакційному режимі клієнт і сервер беруть участь в "транзакції", що означає, що вони виконують (як правило) єдиний обмін запитом і відповіддю. Зв'язок, як правило, припиняється після того, як клієнт отримує відповідь. Іноді ці режими об'єднуються, наприклад, розмовний обмін може бути використані в контексті

однієї транзакції. У цьому випадку, зазвичай, надсилаються спеціальні повідомлення, щоб позначити початок і кінець операції. Розмова відбувається між цими маркерами обміну. Це більше функція програми, ніж проміжне програмне забезпечення.

Крім того, проміжний рівень також може мати деяку семантику високого рівня вище стандартних синхронних і асинхронних режимів зв'язку. Наприклад, повідомлення-відповідь CORBA LOCATION-FORWARD (див. OMG, 2000) виконує завдання перенаправлення, наказуючи клієнту повторно надіслати початковий запит до нової кінцевої точки призначення, і можливість обробки цих завдань повинна бути включена в будь-який загальний підхід до сумісності з проміжним програмним забезпеченням. Ряд підходів до проміжного програмного забезпечення описані в літературі, і ряд був реалізований як комерційна продукція. Типовим автоматизованим підходом є те, що пропонується Dashofy та ін., які досліджували використання різних поза полицею продуктів проміжного програмного забезпечення для побудови мостів або роз'ємів для розподілених систем. Вони представляють свої погляди з точки зору архітектури програмного забезпечення, обмеженої архітектурною моделлю C2. Вони створили програмні роз'єми, специфічні для чотирьох пакетів середнього програмного забезпечення, Q, Polylith, RMI та ILU. Це означає, що для додавання підтримки іншого пакету середнього ПЗ необхідно буде створити новий специфічний роз'єм (програму). Комерційні інструменти, такі як ті, що надаються комерційними продуктами інтеграції корпоративних додатків (EAI), аналогічно обмежені. Більшість підходів вимагають, щоб програмне забезпечення було переписати для спілкування з кожною іншою реалізацією середнього ПЗ. На відміну від цього, цей проект спрямований на більш простий і гнучкий підхід, при якому правила та характеристики проміжного програмного забезпечення вказані в репозиторії, для брокера забезпечити з'єднання та трансформацію для різних протоколів середнього програмного забезпечення, а також для застарілих систем.

Архітектурні підходи забезпечують в основному високий рівень моделювання систем, і не мають великої практичної користі для низько рівня

системного інтегратора. Звичайно, вони не дозволяють ніякої автоматизованої інтеграції на рівні протоколів.

Специфічний підхід до проміжного програмного забезпечення, де такі системи, як ASTER надають API, який дозволяє перекладати різні протоколи на CORBA (тобто ASTER спирається на одне середнє програмне забезпечення, CORBA, для надання всіх віддалених (RPC) та компонентних послуг).

Декларативні та напівавтоматичні підходи, що надають опис протоколу та рушія або набору процедур та сховищ даних, які можуть перетворити будь-який протокол на будь-який інший.

Об'єкт дослідження вписується в категорію декларативних і напівавтоматичних підходів. Перевагою такого підходу є те, що програмування мінімізовано, а протокол потрібно визначити лише один раз. Недоліком є те, що MPDL, який може обробляти широкий спектр протоколів, обов'язково складний, як і двигун, який здійснює інтеграцію під час виконання.

Очевидно, що такі підходи мають комерційну цінність, і частина відповідної роботи з'являється в сфері патентів, а не в опублікованій літературі. Підхід з використанням мови визначення протоколу описаний в патенті США, проведеному Гольцманом. Описана мова дуже низько рівні, схожа на збірку, і не призначена для легкого розуміння широким спектром розробників. Як клієнтські, так і серверні вузли, що беруть участь в обміні повідомленнями, повинні реалізувати протокол, який визначає мову. Першим повідомленням в кожному обміні є опис протоколу, який сторони будуть використовувати для спілкування. Наприклад, якщо обидва вузли хотіли б використовувати IIOP (Інтернет-протокол між orb), ініціюючий вузол повинен був би надіслати опис IIOP на вузол отримання, перш ніж будь-які фактичні дані можуть бути обміняні. З опублікованого патентного документа незрозуміло, чи може цей механізм обробляти протокол, такий складний, як IIOP. Крім того, вузли, що беруть участь у обміні повідомленнями, повинні мати можливість реалізувати обидва перетворення повідомлень. Наприклад, якщо вузол ініціалізації використовує програму IIOP, а вузол отримання даних використовує протокол SOAP (HTTP), обидва вони повинні мати змогу виконати потрібне перетворення.

Кузнєцов описує метод генерації перекладу повідомлень під час виконання. Це досягається шляхом створення XSL відображення вихідного та цільового інтерфейсів. Це зіставлення використовується під час обробки повідомлень для створення власного коду об'єкта, який перетворює повідомлення з вихідного інтерфейсу на представлення цільового інтерфейсу. Це, здається, задовольнити більше для відображення інтерфейсу, а не різні формати проміжного програмного забезпечення. Підхід орієнтований на XML/XSLT. Існує згадка про аналізатори для різних типів визначень (наприклад, ASN.1, C++), які описують структури даних інтерфейсу, а не характеристики протоколів.

Хоча переклад протоколу визнається важливою проблемою, немає загальноприйнятого єдиного способу зробити це, і існує значна сфера для вдосконалення поточних підходів. Інші підходи, які забезпечують перенастроєні фреймворки проміжного програмного забезпечення, безпосередньо не адресують переклад протоколу. Переклад протоколу обслуговується архітектурою на основі плагіна, шикуючи, що користувачу фреймворка потрібно буде або джерело (від третьої сторони), або запрограмувати необхідний плагін для виконання перекладу. Загальний брокер проміжного програмного забезпечення повинен працювати з низкою існуючих протоколів проміжного пз і легко адаптуватися до будь-якого нового протоколу проміжного ПЗ. Він повинен мінімізувати розширення програмування до інструменту інтеграції проміжного програмного забезпечення при додаванні нових протоколів проміжного ПЗ, бажано можливість описувати протоколи Проміжне програмне забезпечення повністю декларативним чином, і мовою, якою може легко скористатися широке коло розробників. Характеристики вдосконаленого та загального брокера проміжного посуду можуть бути узагальнені в наступних п'яти основних вимогах.

TUBE спрямована на вдосконалення сучасних автоматизованих підходів, особливо тих, які були окреслені Гольцманом та Кузнєтовим. Він забезпечить MPDL, який є розширенням IDL і, таким чином, легко розуміється, він буде здатний описувати складні протоколи, такі як IIOP (CORBA). Протоколи та інтерфейси не залежить один від одного. Тобто інтерфейси визначаються в IDL і протоколи визначаються в MPDL. MPDL визначає всі характеристики

проміжного програмного забезпечення, включаючи кінцеві точки та будь-які API або бібліотеки, необхідні.

TUBE дозволить вузлам, що беруть участь в обміні повідомленнями, використовувати будь-який протокол, який найкраще відповідає їхнім потребам, без будь-яких вимог для будь-якого кінця для реалізації певного протоколу. Наприклад, якщо ініціюючий вузол використовує IIOP, а вузол отримання даних використовує SOAP (HTTP), не потрібно знати про протокол інших користувачів. Відправник просто відправляє повідомлення в IIOP, і одержувач отримає його в SOAP через HTTP. Брокер відображає протоколи на одному або обох кінцях.

Зберігаючи визначення інтерфейсу окремо від визначення кодування TUBE дозволяє задовольняти будь-який протокол та комбінацію інтерфейсу. TUBE не обмежує переклад таких підходів, як XML за допомогою XSLT.

2 ПРОЄКТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

2.1 Топологія тор

Топологічні багатовиди зображено на рис. 2.1.

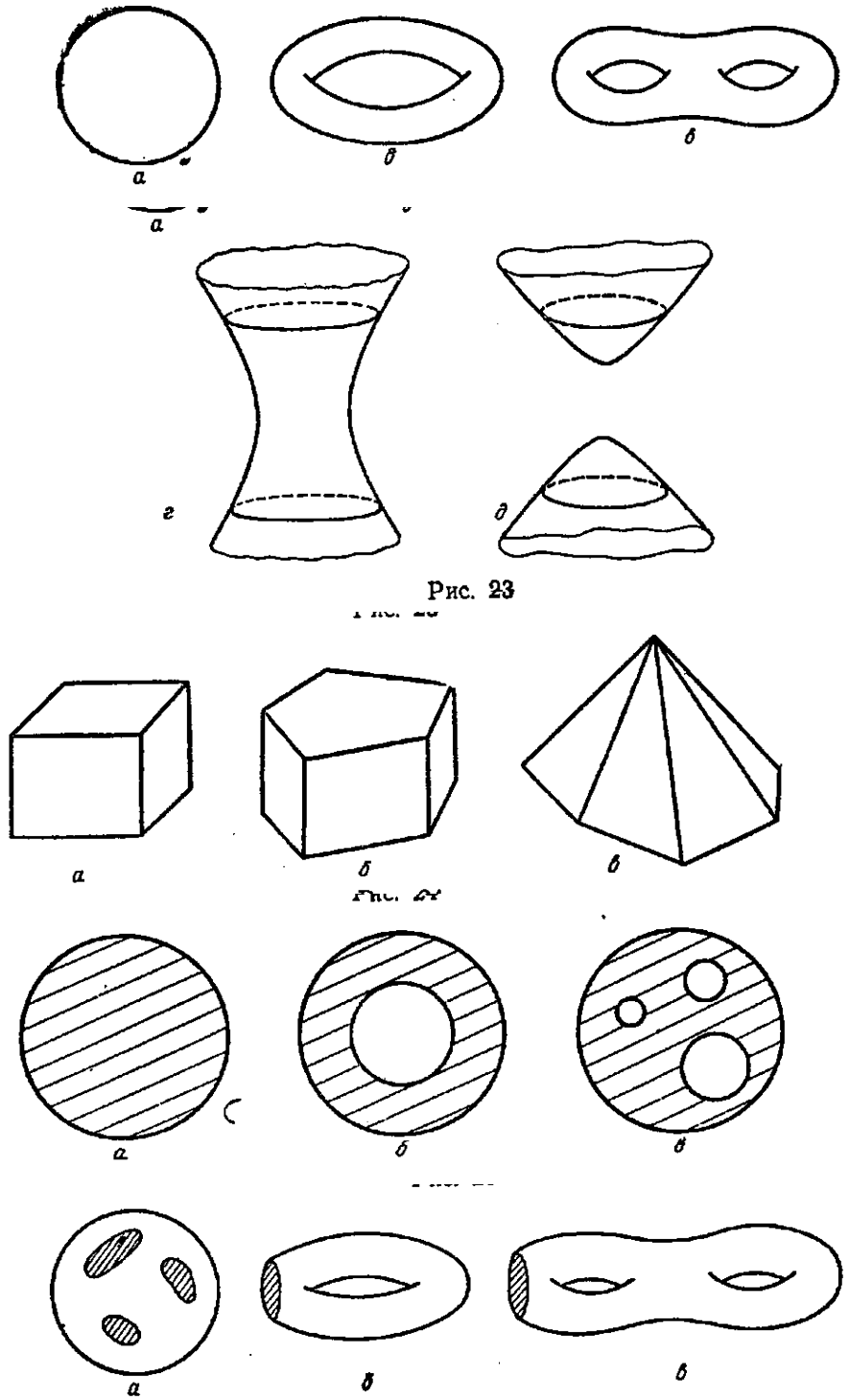

Рисунок 2.1 – Топологічні багатовиди

Топологія - це наука, що вивчає деформації. Так, наприклад, чашка і тор це різні речі, але ми можемо перевести одну в іншу безперервної деформацією. Тіла, які можна перевести один в одного безперервної деформацією, називаються гомеоморфними (мають однакову форму); Згідно відомому жарті, тополог - це людина, не здатна відрізнити кавову чашку від бублика!

Тор - поверхня обертання, що отримується обертанням утворює кола навколо осі, що лежить в площині цієї кола.Гіперкуб (N-куб) – це мережа з 2^N вузлів, розташованих в вершинах n-мірного куба. Гіперкуб це узагальнення звичайного куба тривимірної форми.

Вісь тора може лежати поза кола або торкатися її.

Точка на торі параметризується двома кутовими координатами: одна описує стан точки над обертаючим колом, а інша кут, на який коло потрібно повернути. Можна помітити аналогію з широтою і довготою.

Рівняння тора з відстанню від центру утворює кола до осі обертання R і з радіусом утворює кола r може бути задано в параметричному у вигляді:

$$\begin{cases} x(\phi,\psi) = & (R + r\cos\phi)\cos\psi \\ y(\phi,\psi) = & (R + r\cos\phi)\sin\psi \\ z(\phi,\psi) = & r\sin\phi \end{cases} \qquad \phi,\psi \in [0, 2\pi)$$

Непараметричне рівняння в тих же координатах і з тими ж радіусами має четверту ступінь:

$$\left(x^2 + y^2 + z^2 + R^2 - r^2\right)^2 - 4R^2\left(x^2 + y^2\right) = 0$$

Зокрема, тор є поверхнею четвертого порядку.

Площа поверхні тора як наслідок з першої теореми Гульдін:

$$S = 4\pi^2 Rr.$$

Об'єм тіла, обмежуваного тором, як наслідок з другої теореми Гульдіна:

$$V = 2\pi^2 R r^2.$$

На рисунку r позначений як b.

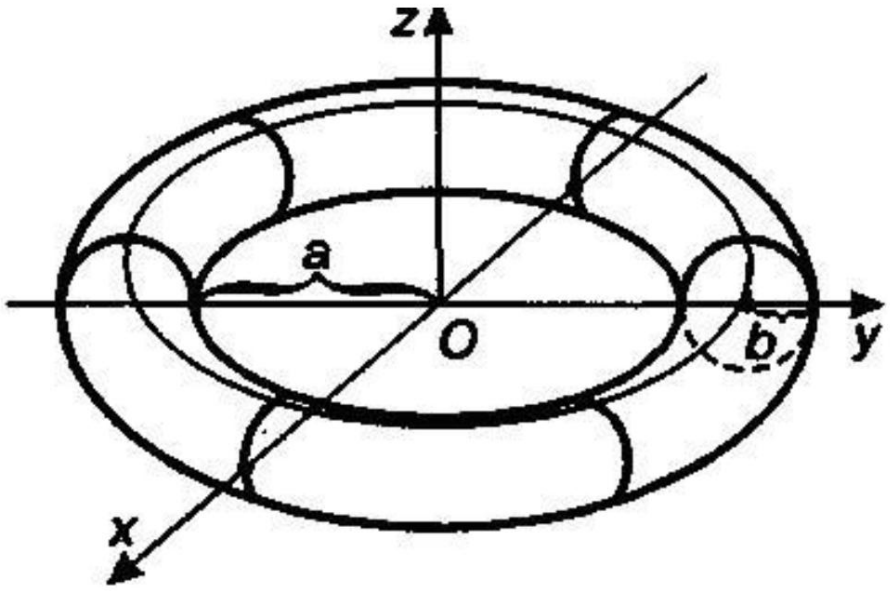

Рисунок 2.2 – Тор

На торі обертання добре видно два сімейства кіл: меридіани (сині лінії) і паралелі (червоні).

Коли розглядають сферу, можна було легко сказати, що меридіани відрізняються від паралелей тим, що кожен з них проходить через обидва полюси, проте на торі ніяких полюсів, та й взагалі точок перетину меридіанів немає. Тому існує угода називати сині лінії меридіанами, тому що площині, в яких вони лежать, містять вісь обертання, а червоні лінії - паралелями, так як площині, їх містять, перпендикулярні осі обертання.

Застосування тороїдально-решітчастих комунікаційних мереж в масивно-паралельних мультипроцесорних комп'ютерних системах. Серед паралельних обчислювальних систем (суперкомп'ютерів) найбільш масштабними та потужними є масивно-паралельні КС. В них - від «історичних» до найсучасніших - широко застосовуються КМ, топології яких утворюють клас тороїдально-решітчастих структур - *n*-розмірні тори на основі прямокутних *n*-решіток або гіпертори (булеві гіперкуби або *n*-куби також можуть розглядатися як

найпростіші тороїдально-решітчасті ТС змінної розмірності) [1-4]. Це обумовлено наступними властивостями ТС даного класу [1,5,6]:

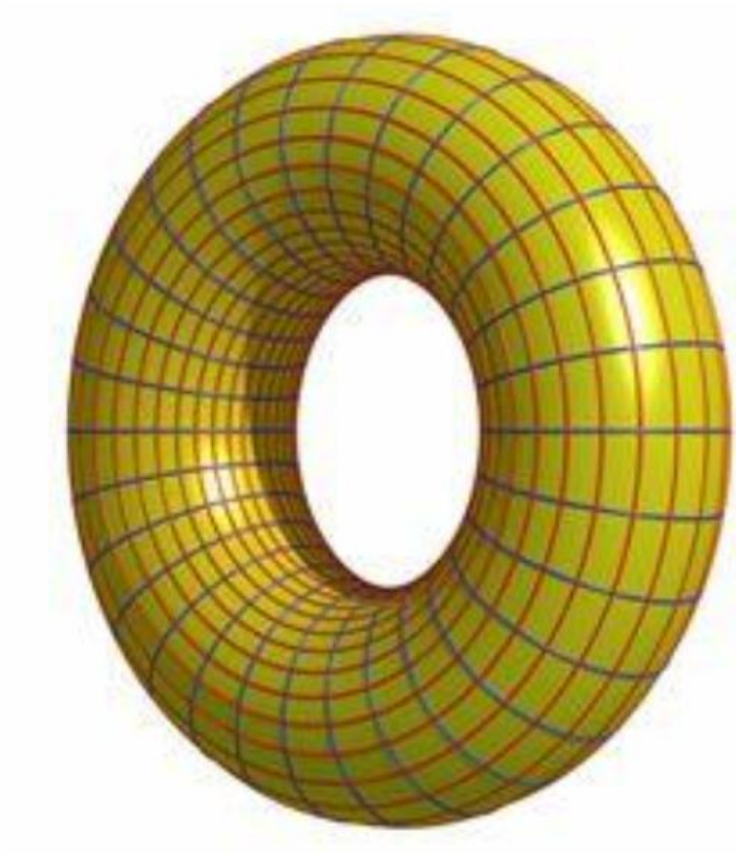

Рисунок 2.3 – Меридіани і паралелі на торі

- конфігурація решітчастих (або стільникових) ТС відповідає специфіці науково-технічних завдань, що вимагають оброблення великих масивів різної розмірності;

- неунівалентні *n*-решітки легко перетворюються на унівалентні *n*-тори шляхом введення в їх структуру додаткових зв'язків [6];

- тороїдально-решітчасті КМ надають широкі можливості їх оптимізації (вибору співвідношення між топологічною вартістю КМ та значеннями метрик, що характеризують її надійність і максимальну затримку передачі повідомлень);

- маршрутизація повідомлень в тороїдально-решітчастих КМ здійснюється на основі простої кубічної функції, що припускає схему адресації вузлів за допомогою рефлексивного коду Грея [1]. Відповідно, алгоритм маршрутизації

ґрунтується на операції порозрядного порівнянні адрес вузла-відправника та вузла-приймача за допомогою логічної операцій XOR (виключне АБО, сума за модулем 2), що легко реалізується апаратним шляхом.

Головна перевага простих решітчастих ТС - масштабованість (велика матриця може бути отримана з матриці меншого розміру простим додаванням вузлів без переустановлення існуючих зв'язків) [1, 2, 4, 6, 8]. Двомірні решітчасті ТС застосовувалися в багатьох КС наприклад, Intel Paragon та Intel Touchstone Delta, Goodyear Aerospace, J-Machine Масачусетського технологічного інституту [8]. Але, прості решітки, як вже вказувалося раніше, є неунівалентними, що й обумовило поступову відмову від їх використання та перехід до тороїдально-решітчастих ТС. ILLIAC IV - суперкомп'ютер сімейства ILLIAC (ILLInois Automated Computer) 1965 р. випуску. Був побудований університетом Іллінойсу спільно з корпорацією Burroughs за замовленням NASA. Архітектурно ILLIAC IV являв собою МПКС с деякими характерними рисами систолічних матриць та масово-паралельних систем. Топологія КМ ILLIAC рис. 2.4) - двомірний тор, де операція скручування застосована тільки до рядків матриці [6,8].

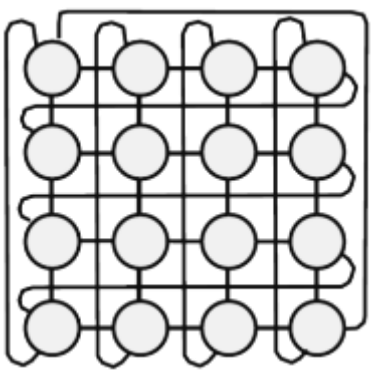

Рисунок 2.4 - Топологія КМ ILLIAC

Двомірний тор реалізований, наприклад, в КС iWarp. КМ топології трьохмірний тор - в КС Cray T3D. КМ лінійки суперкомп'ютерів IBM Blue Gene (Blue Gene/L, Blue Gene/P) також побудовані на основі топології «тривимірний тор» [6,8]. Blue Gene/L є першим із названої лінійки. Максимальна конфігурація КМ системи, що може містити від 1024 до 65536 обчислювальних вузлів, -

трьохмірний тор розміром 64x32x32. КС SpiNNaker, розроблена в Манчестерському університеті у Великобританії, призначена для моделювання дуже великих, біологічно реалістичних нейронних мереж в режимі реального часу. КС має властивості масового паралелізму та високої відмовостійкості [18]. Система включає до свого складу 65 536 ідентичних 18-ядерних процесорів (всього 1 179 648 ядер). Кожен процесор реалізований у вигляді спеціально розробленої мікросхеми (має власний маршрутизатор та 128 Мб пам'яті для зберігання синаптичних даних), підключений до шести сусідів, утворюючи тороїдальную мережу. Принцип побудови КМ КС SpiNNaker пояснюється рис. 2.5.

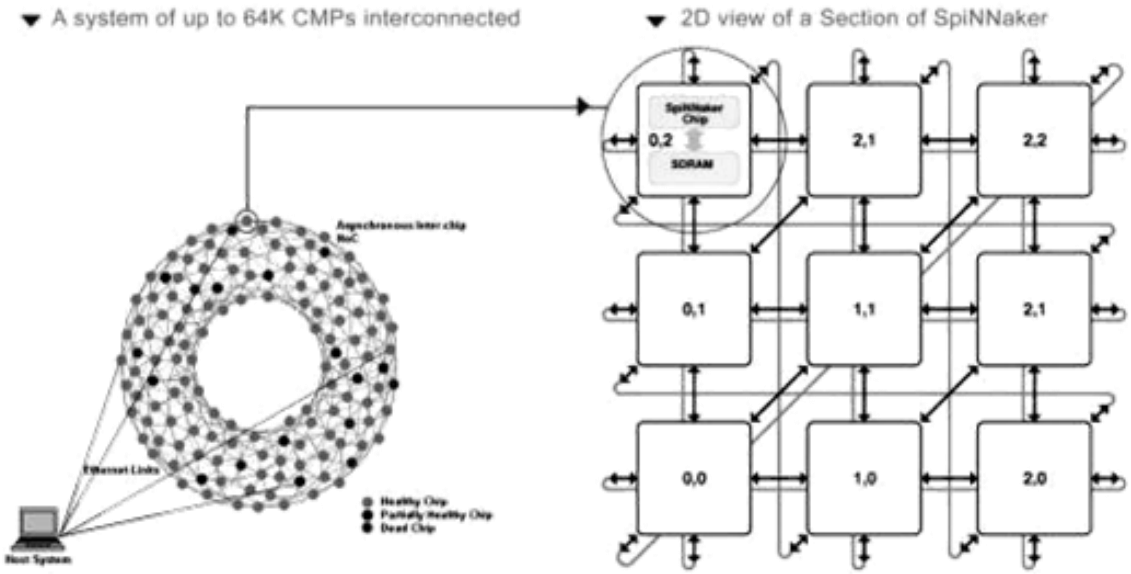

Рисунок 2.5 - Схематичне представлення топології КМ БПКС SpinNaker [11]

Взагалі в сучасних БПКС спостерігається тенденція щодо збільшення розмірності КМ тороїдально-решітчастих топологій. Наприклад, КС Blue Gene/Q використовує вже п'ятивимірний тор. Мікропроцесорні гіперкуби уперше почала створювати компанія IMS Associates. Спочатку розповсюдженню гіперкубічних КМ суттєво заважала їх основна негативна риса - стрімке зростання порядку вузлів зі збільшенням розміру мережі. Перші розробники гіперкубічних КМ намагалися зменшити обсяг комунікаційного обладнання, модифікуючи гіперкуб шляхом введення у нього ієрархічних структур, але ці спроби не виходили за межі університетських лабораторій з причини неготовності елементної бази і, як

наслідок, занадто великого обсягу устаткування, необхідного для реалізації практично значимих КМ. Ситуація стала змінюватися на початку вісімдесятих, коли з'явилися потужні 16/32-розрядні процесори і мікросхеми пам'яті з ємністю від сотні кілобайт і вище. Тільки тоді в 1983 році в Каліфорнійському технологічному інституті вдалося створити справжній працюючий гіперкуб Cosmic Cube розміром 64 вузли. Поява Cosmic Cube стимулювала і Intel до створення в 1985-му «персонального суперкомп'ютера» iPSC/1 (Intel Personal Supercomputer) з кількістю вузлів до 128. Подібну КС System/14 представила компанія Ametek, теж на 8086/87, але з удвічі більшою кількістю вузлів. В 1985 році з'явилося кілька КС (Caltech/JPL Mark III, Connection Machine, Intel iPSC-VX, Floating Point Systems T Series), КМ яких мала гіперкубічну топологію. Гіперкубічна КМ також була реалізована в КС Intel iPSC. Великих успіхів в свій час досягла компанія nCube. КС nCube/ten (1985 р.) має у своєму складі десятирозмірний гіперкуб (1024 вузли). КМ nCube-2 (1989 р.) вже була побудована на основі гіперкуба розмірністю 13 та містила 8192 процесора. Розмірність гіперкуба в останній версії системи nCube -3 (1995 р.) досягла 16, кількість процесорів, відповідно - 65536. Одним із останнім досягненням в цій сфері став збудований в кінці 2011 року японський суперкомп'ютер - K Computer компанії Fujitsu, що мав 88128 процесорів (705024 ядер). Продуктивність системи на тесті Linpack досягла 10,51 Пфлопс. Пікова швидкодія комплексу досягає 11,28 квадрильйона операцій з плаваючою комою в секунду. На сьогоднішній день Китай став лідером у всесвітній гонці розрахункової потужності суперкомп'ютерів. Його три суперкомп'ютери Tianhe-1A, Tianhe-2 та Sunway TaihuLight стали самими потужними в своєму роді та в свій час. Tianhe-1A був запущений в кінці 2010 року, мав 7168 графічних процесорів Nvidia Tesla M2050 і 14336 серверних процесорів Intel Xeon, швидкість його розрахунків дорівнює 2,57 Пфлопс. В 2013 році був запущений Tianhe-2, в своєму складі він мав 3,12 мільйонів ядер(384 тисячі процесорів Ivy Bridge і 2736 тисячі Intel Xeon Phi), практична швидкість розрахунків - 33,86 Пфлопс, теоретична - 54,9 Пфлопс. Самим останнім та на даний час найпотужнішим суперкомп'ютером в світі став Sunway TaihuLight. Він був запущений в кінці 2016 року, має 40960 процесорів

SW26010 власного виробництва, тобто всього комплекс має в своєму розпорядженні 10,6496 мільйонів ядер. Його практична продуктивність становить 93 Пфлопс, за тестом LINPACK, а теоретична пікова швидкість - 125,43 Пфлопс.

Таким чином, топологія тор може бути перспективною для вирішення багатьох задач предметної області і на відміну від топології гіперкуб бути побудованою за мінімальної кількості компонентів.

2.2 Кеш-пам’ять у комп’ютерних розподілених системах

Подібно до NUMAв системі COMA кожен процесор має частину спільної пам'яті (рис. 2.6). Проте в даному випадку спільна пам'ять є кеш пам'яттю. Система COMA вимагає, щоб дані мігрували до процесора, що запросив їх.

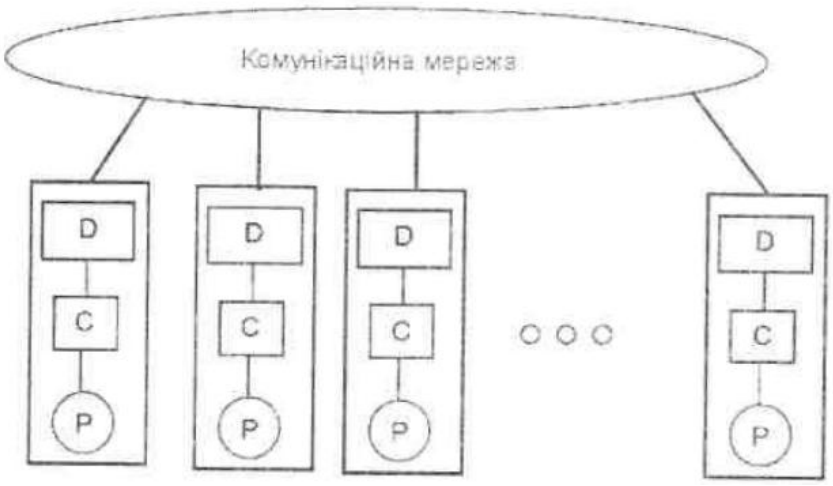

Рисунок 2.6 – Система із спільною пам’яттю

Тут відсутня ієрархія пам'яті, а адресний простір визначається ємністю кеш пам'яті. Крім того, тут наявна директорія Dкеш пам'яті, яка допомагає у віддаленому доступі до кеш пам'яті. Прикладом цієї архітектури є система KSR-1 фірми Kendall Square Recearch.

Системи передачі повідомлень - клас багатопроцесорних систем, в яких кожен процесор має доступ до своєї локальної пам'яті. На відміну від систем із спільною пам'яттю, комунікації в системі передачі повідомлень виконуються

шляхом посилання та отримання повідомлень. Вузол в такій системі складається з процесора і його локальної пам'яті (рис. 2.7).

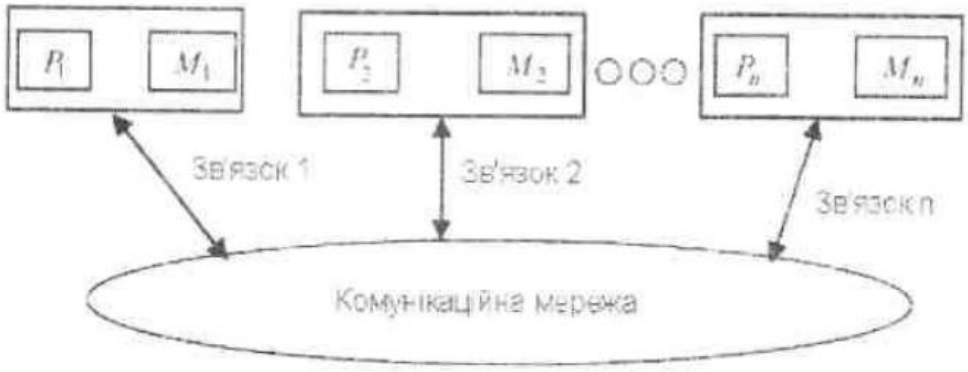

Рисунок 2.7 – Система з розподілено пам’яттю

Вузли зазвичай запам'ятовують повідомлення в буферах (тимчасові комірки пам'яті, де повідомлення чекають до того часу, коли вони можуть бути послані або одержані), і виконують посилання-отримання повідомлення під час обробки даних. Паралельна обробка повідомлення і поточного обчислення здійснюється під керуванням операційної системи. Процесори не розділяють спільну пам'ять і кожен процесор має доступ до його власного адресного простору. Вузли в системі передачі повідомлень можуть бути з'єднані різноманітними способами - від спеціалізованих комутаторів до географічно розподілених мереж.

Підхід передачі повідомлень є масштабованим до великих пропорцій, тобто число процесорів може бути збільшено без істотного зменшення ефективності їх взаємодії.

Для організації зв'язку системи передачі повідомлень використовують статичні мережі, зокрема гіперкубічні мережі, які були досить популярні протягом багатьох років. Двовимірні і тривимірні решітчасті мережі так само широко використовувались в системах передачі повідомлень. Два важливі чинники повинні розглядатися в проектуванні комунікаційної мережі для систем передачі повідомлень. Це - пропускна здатність і мережний час очікування. Пропускна здатність визначена як кількість бітів інформації, переданих за одиницю часу (біт/сек). Мережний час очікування визначений як час, потрібний

для передачі повідомлення. В 1987 році була введена блокова маршрутизація як альтернатива традиційній маршрутизації з проміжним зберіганням для того, щоб скоротити розмір необхідних буферів і щоб зменшити час очікування повідомлення. У блоковій маршрутизації пакет ділиться на менші блоки, які називають блоками керування потоком даних, так, що ці блоки рухаються конвеєрним методом разом з блоком заголовка до вузла призначення. Коли блок заголовка блокується через мережний затор, наступні за ним блоки блокуються також.

В основі архітектури будь-якої багатопроцесорної комп'ютерної системи лежить здатність до обміну даними між її компонентами. Це забезпечується комунікаційною мережею (КММ), яка з'єднує між собою вузли комп'ютерної системи за допомогою каналів передачі даних (каналів зв'язку). В ролі вузлів можуть виступати процесори, модулі пам'яті, пристрої введення-виведення, комутатори або декілька перерахованих елементів, об'єднаних у функціональний пристрій. Організація внутрішніх комунікацій комп'ютерної системи називається топологією.

Топологію комунікаційної мережі визначає множина вузлів, які об'єднані множиною каналів. Зв'язок між вузлами зазвичай реалізується по двоточковій схемі (point-to-point). Будь-які два вузли, зв'язані каналом зв'язку, називають суміжними вузлами або сусідами. Кожен канал з'єднує один вузол-джерело з одним вузлом-приймачем. Канал характеризується кількістю сигнальних ліній, частотою або швидкістю передачі бітів по кожній сигнальній лінії, затримкою - часом пересилання біта з одного вузла до іншого. Для більшості каналів затримка знаходиться в прямій залежності від фізичної довжини лінії зв'язку та швидкості, розповсюдження сигналу.

Вузол у мережі може бути термінальним, тобто джерелом або приймачем даних, комутатором, що пересилає інформацію з вхідного порту на вихідний, або суміщати обидві ролі. У мережах із прямими зв'язками кожен вузол одночасно є як термінальним вузлом, так і комутатором, і повідомлення пересилаються між термінальними вузлами безпосередньо. У мережах з непрямими зв'язками вузол може бути або термінальним, або комутатором, але не одночасно, тому

повідомлення передаються опосередковано, за допомогою виділених комутуючих вузлів. Існують також такі топології, які не можна однозначно зарахувати ні до прямих, ні до непрямих. Будь-яку пряму КММ можна зобразити у вигляді непрямої, розділивши кожен вузол на двотермінальний вузол і вузол комутації. Сучасні прямі мережі реалізуються саме таким чином - комутатор від діляється від термінального вузла і поміщається у виділений маршрутизатор. Основна перевага прямих КММ полягає в тому, що комутатор може використовувати ресурси термінальної частини свого вузла. Це стає істотним, якщо врахувати, що, як правило, останній включає комп'ютер або процесор.

Комунікаційні мережі багатопроцесорних систем можуть бути поділені на наступні групи: залежно від того, чи залишається конфігурація взаємозв'язків незмінною, принаймні поки виконується певне завдання, розрізняють мережі із статичною і динамічною топологіями; залежно від способу функціонування: синхронні та асинхронні; залежно від стратегії керування: централізовані та децентралізовані; залежно від способу перемикання: схемні та пакетні.

З'єднання в статичній мережі є фіксованими, тоді як в динамічній мережі вони можуть змінюватися в процесі роботи мережі за допомогою програмних засобів. При цьому статичні комунікаційні мережі в свою чергу ділять на одно-, двовимірні та гіперкубічні. Динамічні комунікаційні мережі ділять на шинні (одно та багатошинні) та комутуючі (одноярусні, багатоярусні та координатні), як це показано на рис. 2.8.

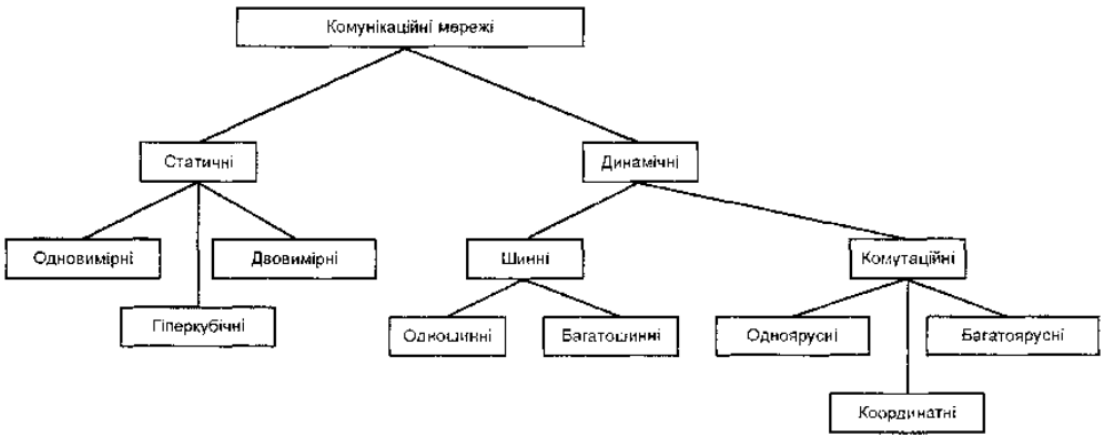

Рисунок 2.8 – Топології мереж

2.3 Вибір програмно-апаратного забезпечення

Розглянемо проведення підбору апаратних засобів для побудови мультипроцесорної системи загального призначення на основі топології тор.

Дві можливі стратегії операцій взаємодії в мережі - це синхронна й асинхронна. У синхронних КММ всі дії жорстко узгоджені в часі, що забезпечується за рахунок єдиного генератора тактових імпульсів (ГТІ), сигнали якого одночасно транслюються у всі вузли. В асинхронних мережах єдиного генератора немає, а функції синхронізації розподілені по всій системі, причому в різних частинах мережі часто використовуються локальні ГТІ.

Залежно від вибраної стратегії комутації розрізняють мережі з комутацією з'єднань і мережі з комутацією пакетів. Як у першому, так і в другому варіанті інформація пересилається у вигляді пакету. Пакет є групою бітів, для позначення якої застосовують також термін повідомлення.

У мережах з комутацією з'єднань шляхом відповідного встановлення комутуючих елементів мережі формується канал від вузла-джерела до вузла-приймача, що зберіга ється, поки весь пакет не досягне пункту призначення. Пересилання повідомлень між певною парою вузлів проводиться завжди поодинці і за тим же маршрутом. В мережі з комутацією пакетів приймається, що повідомлення самостійно знаходить свій шлях до місця призначення. На відміну від мереж із комутацією з'єднань, маршрут від початкового пункту до пункту призначення кожного разу може бути іншим. Пакет послідовно проходить через вузли мережі. Черговий вузол запам'ятовує прийнятий пакет у своєму буфері тимчасового зберігання, аналізує його і робить висновки, що з ним робити далі. Залежно від завантаженості мережі ухвалюється рішення про можливість негайного пересилання пакету до наступного вузла і про подальший маршрут проходження пакету на шляху до пункту призначення. Якщо всі можливі канали для переміщення пакету до чергового вузла зайняті, в буфері вузла формується черга пакетів, яка "розсмоктується" у міру звільнення ліній зв'язку між вузлами (якщо черга також насичується, то відповідно до однієї із стратегій маршрутизації

може відбутися так зване "скидання хвоста", тобто відмова від пакетів, що знову поступають). У схемах із децентралізованим керуванням функції керування розподілені по вузлах мережі. Варіант із централізованим керуванням простіше реалізується, але розширення мережі в цьому випадку пов'язане із значними труднощами. У ряді мереж зв'язок між вузлами забезпечується за допомогою деякої множини комутаторів, але існують також мережі з одним комутатором. Наявність великого числа комутаторів веде до збільшення часу передачі повідомлення, але дозволяє використовувати прості комутуючі елементи. Подібні мережі зазвичай будують як багатоярусні.

2.3.1 Порівняльна характеристика процесорів

Багатоядерні процесори - це процесори, що містять на одному процесорному кристалі або в одному корпусі два і більше обчислювальних ядер. Багатоядерність, як спосіб підвищення продуктивності процесорів, використовується з відносно недавнього часу, але визнана найперспективнішим напрямом їх розвитку. Для домашніх комп'ютерів вже існують процесори з 8 ядрами. Для серверів на ринку є 12-ядерні пропозиції (Opteron 6100). Розроблені прототипи процесорів, що містять близько 100 ядер. Ефективність обчислювальних ядер різних моделей процесорів відрізняється. Але у будь-якому випадку, чим їх (ядер) більше, тим процесор продуктивніший.

Чим більше потоків - тим краще. Кількість потоків не завжди співпадає з кількістю ядер процесора. Так, завдяки технології Hyper-Threading, 4-ядерний процесор Intel Core i7 - 3820 працює у 8 потоків і багато в чому випереджає 6-тиядерних конкурентів.

Розмір кеша 2 і 3 рівнів. Кеш - це дуже швидка внутрішня пам'ять процесора, яка використовується ним як буфер для тимчасового зберігання інформації, що обробляється в конкретний момент часу. Чим кеш більший - тим краще. Структура не усіх сучасних процесорів передбачає наявність кеша 3 рівня, хоча це не є критичним моментом. Так, за результатами багатьох тестів продуктивність процесорів Intel Core 2 Quadro, що випускалися з 2007 р. по 2011

р. і не мають кеша 3 рівня, навіть зараз виглядає гідно. Правда, кеш 2 рівня у них досить великий. Частота процесора. Чим вища частота процесора, тим він продуктивніший. Швидкість шини процесора (FSB, HyperTransport або QPI). Через цю шину центральний процесор взаємодіє з материнською платою. Її швидкість (частота) вимірюється в мегагерцах і чим вона вища - тим краще. Поняття техпроцесу розглядалося в попередньому пункті цієї статті. Чим тонший використано техпроцес, тим більше процесор містить транзисторів, менше споживає електроенергії і менше гріється. Від техпроцесу багато в чому залежить ще одна важлива характеристика процесора - TDP. Termal Design Point - показник, що відображає енергоспоживання процесора, а також кількість тепла, що виділяється ним в процесі роботи. Одиниці виміру - Вати (Вт). TDP залежить від багатьох чинників, серед яких головними є кількість ядер, техпроцес виготовлення і частота роботи процесора. Окрім інших переваг, "холодні" процесори (з TDP до 100 Вт) краще піддаються розгону, коли користувач змінює деякі налаштування системи, внаслідок чого збільшується частота процесора. Розгон дозволяє без додаткових фінансових вкладень збільшити продуктивність процесора на 15 - 25 %, але це вже окрема тема. В той же час, проблему з високим TDP завжди можна вирішити придбанням ефективної системи охолодження. Наявність і продуктивність відеоядра. Останні технічні досягнення дозволили виробникам, окрім обчислювальних ядер, включати до складу процесорів ще і ядра графічні. Такі процесори, окрім вирішення своїх основних завдань, можуть виконувати роль відеокарти. Можливостей деяких з них цілком вистачає для гри в комп'ютерні ігри, не кажучи вже про перегляд фільмів, роботу з текстом і вирішення інших завдань. Якщо відеоігри - не головне призначення комп'ютера, процесор з вбудованим графічним ядром дозволить заощадити на придбанні окремого графічного адаптера. Тип і максимальна швидкість підтримуваної оперативної пам'яті. Ці характеристики процесора необхідно враховувати при виборі оперативної пам'яті, з якою він буде використовуватися. Немає сенсу переплачувати за швидкісні модулі ОЗП, якщо процесор не зможе реалізувати усі їх переваги.

Для мультипроцесорної системи загального призначення, розроблюваної в цій роботі я обрала саме Nios 3.0, оскільки в ньому найкраще збалансовані показники вартості та якості, окрім цього він оптимізований для PLD Altera та має вбудовану кеш-пам’ять.

2.3.2 Nios 3.0. Вбудований кеш

Nios 3.0 має п’ятиступінчастий конвеєр зі зменшеним набором команд, побудований на RISC-архітектурі. На рисунку 2.9 зображена блок-схема ядра обраного процесора.

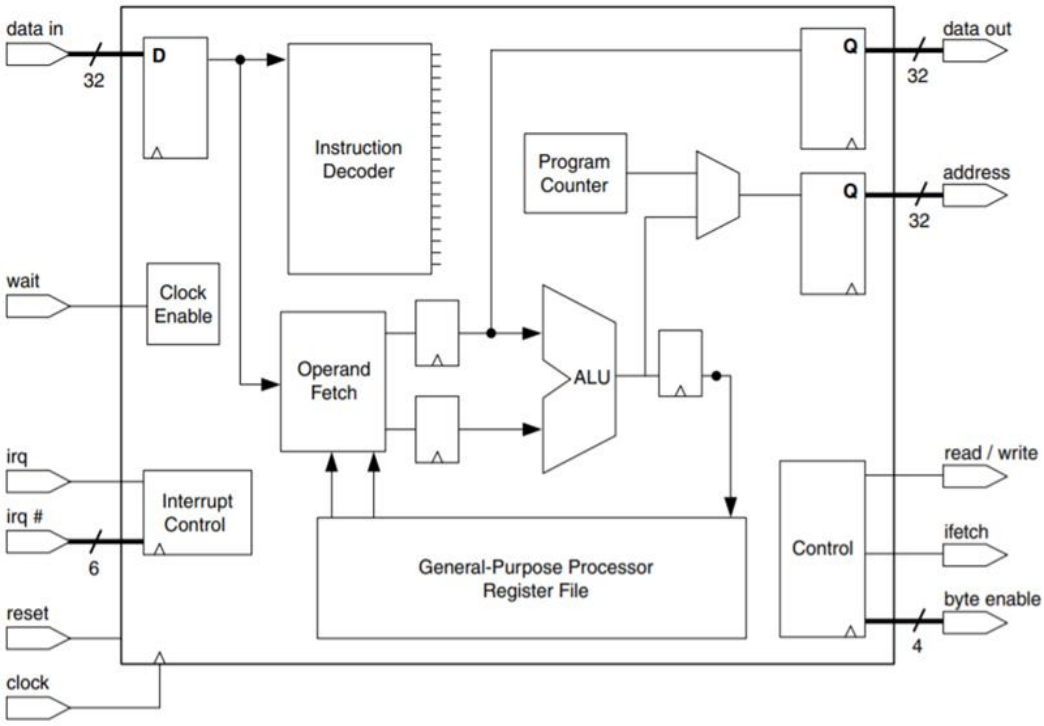

Рисунок 2.9. – Структурна схема ядра Nios 3.0

Настроювані користувачем кеші, версія 3.0 вбудованого процесора Nios включає настроювані користувачем інструкції першого рівня (L1) та кеші даних, які використовують велику двопортову пам’ять на недорогих пристроях Cyclone та Stratix для підвищення системних вимог. Покращений контролер SDRAM, розробники вбудованих процесорів Nios можуть отримати доступ за один цикл до недорогих пристроїв SDRAM зі швидкістю вище 100 МГц.

2.3.3 Шина Avalon

Сьогодні можна говорити про те, що кластерні системи успішно застосовуються для всіх завдань суперкомпьютінга - від розрахунків для науки і промисловості до управління базами даних. Практично будь-які додатки, що вимагають високопродуктивних обчислень, мають зараз паралельні версії, які дозволяють розбивати завдання на фрагменти і обраховувати її паралельно на багатьох вузлах кластеру. Наприклад, для інженерних розрахунків (розрахунки на міцність, аеромеханіка, гідро-та газодинаміка) традиційно застосовуються так звані сіткові методи, коли область обчислень розбивається на клітинки, кожна з яких стає окремою одиницею обчислень. Ці осередки обраховуються незалежно на різних вузлах кластера, а для одержання загальної картини на кожному кроці обчислень відбувається обмін даними, поширеними в прикордонних областях. Для практичних розрахунків (3D-анімація, креш-тести, розвідка нафтових і газових родовищ, прогнозування погоди) зазвичай використовуються кластери з 10-200 вузлів. При цьому основне завдання - забезпечення ефективної роботи кластера з конкретним додатком. Архітектура кластеру повинна забезпечувати масштабованість ПЗ при збільшенні кількості вузлів, тобто приріст продуктивності при додаванні нових обчислювальних модулів. Для цього важливо правильно вибрати конфігурацію кластеру в залежності від профілю обміну даними між екземплярами програми, запущеними на різних вузлах. Тут потрібно брати до уваги загальний обсяг даних, що пересилаються, розподіл довжин повідомлень, використання групових операцій і т. п. Сьогодні навіть ті завдання, для вирішення яких традиційно застосовувалися багатопроцесорні системи зі спільною пам'яттю, такі, як управління великими базами даних, успішно вирішуються на кластерах. Поява на ринку таких продуктів, як, наприклад, Oracle RAC (Real Applications Cluster), дало можливість застосовувати кластерні системи в області баз даних, а нова версія СУБД Oracle 10g, побудована на базі GRID-технологій, забезпечує максимально ефективне використання кластерної архітектури для вирішення цих завдань. Таким чином, завдяки доступності

кластерних рішень число підприємств, які можуть дозволити собі кардинально спростити і прискорити роботу з корпоративною базою даних, істотно збільшується. Кластерні рішення - найбільш економічно обгрунтований вибір. На відміну від більшості серверних систем зі спільною пам'яттю кластерні рішення легко масштабуються до систем більшої продуктивності. Таким чином, при жорсткості вимог замовника до продуктивності необов'язково купувати нову систему - можна додати стандартні обчислювальні вузли і легко наростити стару. Причому діапазон масштабованості дуже широкий: наприклад, 288-вузловий кластер «СКІФ К-1000» з піковою продуктивністю 2,5 TFlops можна наростити до системи потужністю 30 TFlops шляхом об'єднання стандартних обчислювальних модулів.

Кластерні рішення володіють найкращим на сьогодні співвідношенням ціна / проізводітелиюсть і мають істотно більш низьку сукупну вартість володіння. Це досягається завдяки масштабованості та використання стандартних загальнодоступних компонентів, ціна яких постійно знижується. Два кластерних двопроцесорних вузла в середньому на 35% дешевше, ніж один чотирипроцесорних SMP-сервер, причому із зростанням кількості процесорів перевагу кластерних рішень по цьому параметру збільшується. Крім того, кластерна архітектура забезпечує прекрасну відмовостійкість системи: при виході з ладу одного або декількох обчислювальних модулів (або вузлів) кластер не втрачає працездатності і нові завдання можуть бути запущені на меншій кількості вузлів. Що вийшов з ладу вузол легко і швидко виймається з стійки і замінюється новим, який відразу ж включається в роботу. Це можливо завдяки комутованій топології сучасних системних мереж, «коли обмін повідомленнями між двома вузлами може йти багатьма шляхами. У ході експлуатації система типу «СКІФ К-1000» передбачає можливий вихід з ладу не більше 2 вузлів на рік.

Шина Avalon - це архітектура шини, яка була розроблена, для мережі взаємозв'язку для SOPC. З цією метою шина Avalon має простий інтерфейс, який визначає сигнали між головним та веденим портами. Окрім простоти, шина

Avalon була розроблена для використання мінімальних логічних ресурсів в PLD та синхронної роботи, щоб уникнути складних проблем з аналізом часу.

В даний час кластер складається з обчислювальних вузлів на базі стандартних процесорів, з'єднаних високошвидкісний системної мережею (інтерконнектом), а також, як правило, допоміжного та сервісної мережами. Більшість кластерних систем списку Top500 виконані на процесорах Intel (Intel Xeon, Intel Xeon EM64T, Intel Itanium 2). Часто використовуються процесори Power і PowerPC компанії IBM. Останнім часом популярністю користуються процесори AMD (особливо AMD Opteron і його нещодавно вийшла двоядерний версія). В якості обчислювальних вузлів найчастіше використовуються двопроцесорні SMP-сервери в корпусі від 1U до 4U, зібрані в 19-дюйм стійки. Компактні пристрої дозволяють створювати високопродуктивні рішення з максимальною питомою щільністю, більші - недорогі системи. Іноді провідні виробники пропонують власний формфактор: наприклад, IBM, Verari, LinuxNetworx і інші компанії пропонують обчислювальні вузли на основі блейд-технологій, які забезпечують високу щільність установки, але здорожують рішення. На російському ринку блейд-рішення поки мало затребувані із-за їх високої вартості. Кожен вузол працює під керуванням своєї копії стандартної операційної системи, в більшості випадків - Linux. Склад і потужність вузлів можуть бути різними в рамках одного кластера, однак частіше будуються однорідні кластери. Вибір конкретної комунікаційного середовища (інтерконнекту) визначається багатьма чинниками: особливостями розв'язуваних задач, доступним фінансуванням, вимогами до масштабованості і т. п. У кластерних рішеннях застосовуються такі технології інтерконекту, як Gigabit Ethernet, SCI, Myrinet, QsNet, InfiniBand. Кластер - це складний програмно-апаратний комплекс, і завдання побудови кластеру не обмежується об'єднанням великої кількості процесорів в один сегмент. Для того щоб кластер швидко і правильно вважав завдання, всі комплектуючі повинні бути ретельно підібрані один до одного з урахуванням вимог програмного забезпечення, тому що продуктивність кластерного ПЗ сильно залежить від архітектури кластера, характеристик процесорів, системної шини, пам'яті і

інтерконекту. Використання тих чи інших компонентів сильно залежить від завдання, для якої будується кластер. Для деяких добре розпаралелюваних завдань (таких, як рендеринг незалежних сюжетів у відеофрагменті) основний фактор швидкодії - потужний процесор, і продуктивність інтерконекту не грає основної ролі. У той же час для задач гідро-і аеродинаміки, розрахунку креш-тестів важлива продуктивність системної мережі, інакше збільшення числа вузлів в кластері буде мало впливати на швидкість вирішення завдання. Системна мережу, або високошвидкісна комунікаційне середовище, виконує завдання забезпечення ефективності обчислень. Gigabit Ethernet - найбільш доступний тип комунікаційного середовища, оптимальне рішення для завдань, які потребують інтенсивних обмінів даними (наприклад, візуалізація тривимірних сцен або обробка геофізичних даних). Ця мережа забезпечує пропускну здатність на рівні MPI * (близько 70 Мбайт / с) і затримку (час між відправленням і отриманням пакету з даними) приблизно 50 мкс. Myrinet - найбільш поширений тип комунікаційного середовища з пропускною здатністю до 250 Мбайт / с і затримкою 7 мкс, а нове, нещодавно анонсований ПЗ для цієї мережі дозволяє скоротити цю цифру в два рази. Мережа SCI відрізняється невеликими затримками - менше 3 мкс на рівні MPI - і забезпечує пропускну здатність на рівні MPI від 200 до 325 Мбайт / с. QsNet - дуже продуктивне і дороге устаткування, що забезпечує затримку менше 2 мкс і пропускну здатність до 900 Мбайт / с. Найбільш перспективна на сьогодні технологія системної мережі - InfiniBand. Її поточна реалізація має пропускну здатність на рівні MPI до 1900 Мбайт / с і час затримки від 3 до 7 мкс. Один з найбільш цікавих продуктів, що з'явилися останнім часом, - високошвидкісний адаптер компанії PathScale, який реалізує стандартні комутатори та кабельну структуру InfiniBand, використовуючи власний транспортний протокол. Це дозволило досягти рекордно низького часу затримки - 1,3 мкс.

Зараз існують два способи внутрішнього устрою стандартних системних мереж. Наприклад, мережа SCI має топологію двох-або тривимірного тора і не вимагає застосування комутаторів, що зменшує вартість системи. Проте ця

технологія має істотні обмеження по масштабованості. Решта загальнодоступні високошвидкісні технології системних мереж Myrinet, QsNet, InfiniBand використовують комутовану топологію Fat Tree. Обчислювальні вузли кластера з'єднуються кабелями з комутаторами нижнього рівня (leaf, або edge switches), які в свою чергу об'єднуються через комутатори верхнього рівня (core, або spine switches). При такій топології є багато шляхів передачі повідомлень між вузлами, що дозволяє підвищити ефективність передачі повідомлень завдяки розподілу завантаження при використанні різних маршрутів. Крім того, за допомогою Fat Tree можна об'єднати практично необмежену кількість вузлів, зберігши при цьому хорошу масштабованість додатків. Завдання ефективного доступу вузлів до даних (наприклад, до зовнішнього сховища) найчастіше вирішується за допомогою допоміжної мережі (як правило, Gigabit Ethernet). Іноді для цього застосовують канали Fibre Channel (це значно збільшує вартість системи) або системну мережу (наприклад, InfiniBand у кластерах баз даних). Допоміжна (або сервісна) мережа також відповідає за розподіл завдань між вузлами кластеру та управління роботою завдань. Вона використовується для файлового обміну, мережевий завантаження ОС вузлів і управління вузлами на рівні ОС, в тому числі моніторингу температурного режиму та інших параметрів роботи вузлів. Сервісна мережа застосовується і для так званого керування вузлами out-of-band, тобто без участі операційної системи. До нього відносяться «плавне», послідовне включення і виключення вузлів вр уникнути великого стрибка напруги, апаратне скидання вузла і доступ до його консолі на всіх етапах роботи, що дозволяє діагностувати поломки в недоступних вузлах, змінювати налаштування ОС та ін Провідні виробники суперкомп'ютерів, такі , як IBM, SUN, HP, вводять до складу вузла спеціальні плати, що дозволяють здійснювати управління out-of-band, які в перерахунку на весь кластер досить дорогі. На щастя, є набагато більш доступне російське рішення з аналогічною функціональністю - мережа ServNet, розроблена в Інституті програмних систем РАН і успішно застосовується у вітчизняних кластерних системах, зокрема в кластерах «СКІФ». Компактна плата ServNet (всього 66х33 мм) легко вбудовується в обчислювальний вузол і дозволяє, крім усього перерахованого вище, змінювати параметри BIOS вузла, вибирати

завантажувану ОС, змінювати параметри завантаження ядра Linux, контролювати критичні повідомлення ОС і проводити «посмертне» читання (з енергонезалежної пам'яті плати ServNET) кількох останніх повідомлень ОС. Суперкомп'ютери - це завжди дуже великі потужності. У ситуації, що склалася вже неможливо розглядати високопродуктивні обчислювальні системи окремо від систем їх розміщення, охолодження та електроживлення. Наприклад, «СКІФ К-1000» споживає понад 89 кВт, і практично все йде в тепло. Такої потужності було б достатньо для обігріву невеликого будинку, але все 288 вузлів формфактора 1U стоять у восьми стійках, і без продуманого теплового дизайну не обійтися. У перших суперкомп'ютерах використовувалося рідинне охолодження, але такі охолоджувальні системи нерідко виходили з ладу. У сучасних суперкомп'ютерах застосовують повітряне охолодження, і необхідний температурний режим забезпечується двома факторами. По-перше, продуманим тепловим дизайном обчислювального вузла: стандартні шасі необхідно модернізувати для того, щоб повітряний потік, створюваний внутрішніми вентиляторами, максимально ефективно охолоджував процесори. По-друге, підтриманням робочої температури в приміщенні: гаряче повітря повинен бути або відведений від вузлів і кондиціонований, або виїхали за межі приміщення. Оптимізація енергоспоживання - не менш серйозне завдання. На думку світових експертів, при сучасних темпах зростання продуктивності систем і збереженні характеристик їх енергоспоживання вже до 2010 р. самі потужні суперкомп'ютери будуть споживати стільки енергії, що забезпечити її подачу і відведення тепла буде неможливо. Проте проблема забезпечення безперебійного живлення існує і для систем з середньою продуктивністю, і кожен виробник вирішує її по-своєму.

Nios використовує порти вводу-виводу для доступу до пам'яті та периферійних пристроїв на шині Avalon (процесор Nios, пов'язані з ним підлеглі периферійні пристрої та шина Avalon спільно називаються системним модулем).

2.4 Висновки

У цьому розділі було розглянуто топологію «тор» при використанні її для мультипроцесорних систем загального призначення. Визначено її основні характеристики та покази ефективності роботи.

Було висвітлено роботу симетричної системи зі спільною пам’ятю, піднято і пояснено проблему когерентності кеш-пам’яті та проілюстровано роботу кешу у розроблювальній системі. Представлено важливий аспект – забезпечення коректності виконання коду та актуалізація даних при розробці та виконанні паралельних та багатопотокових програм.

Проведено порівняльну характеристику декількох процесорів фірми Altera, описано основні особливості обраного процесора Nios 3.0 та інтерфейсу шини Avalon.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНО-ТЕХНІЧНОГО ЗАСОБУ

3.1 Засоби для проектування мультипроцесорної системи

3.1.1 Засоби візуалізації роботи – макетна плата DE1-SoC

Якого рівня продуктивність потрібна сьогодні на мультипроцесорних системах, скільки процесорів можуть налічувати у своєму складі мультипроцесорні системи і які галузі є основними споживачими високопродуктивних обчислень, можна оцінити хоча б з наступних прикладів: Комп'ютерне моделювання задач газо- і гідродинаміки характеризується великими обсягами обчислень. Наприклад, гідродинамічні процеси при видобутку нафти, за умови розгляду області моделювання протягом 103 кроків моделювання і обсягом 106 дискретних точок, в кожній з яких оцінюється значення 10 функцій, кожна з яких вимагає порядку 103 операцій, матимуть обчислювальну складність порядка T=103 *106 *103 *10=1013 операцій з плаваючою точкою. З цього можна зробити висновок, що виконання цього завдання у " реальному часі" за 10 сек вимагатиме продуктивності паралельної системи порядка 1 Тфлопс. Така продуктивність досяжна на сьогодні тільки для мультипроцесорних систем, що складаються з тисяч процесорів. Інший приклад − менш чисельної за складом мультипроцерної системи. Для обслуговування клієнтів системи резервування авіаквитків Amadeus у США з 180 000 терміналів і 60 млн. запитів за добу за умови, щоб одна операція резервування квитка виконувалась у середньому кільканадцять секунд (оцінки 2000 року), знадобилось два сервери від Hewlett-Packard по 12 процесорів у кожного.

На сьогодні найбільш розповсюдженими архітектурами паралельних обчислювальних систем є:

1) векторно-конвейерні;

2) симетричної мультиобробки (SMP);

3) масивно паралельні;

4) кластерні;

5) метакомп′ютерні.

Ці класи архітектур у “ чистому” вигляді зазвичай не використовуються і існують лише у різних комбінаціях між собою, до яких вдаються виробники мультипроцесорних систем.

Яскравим представником цього класу архітетктур є мультипроцесорні системи фірми Cray. Під керівництвом засновника компанії С. Крея у 1976 р. було розроблено першу з серії машин Cray-1, з якою пов’язано походження терміну “суперкомп’ютер” Як не дивно, це була однопроцесорна машина, але досить складної побудови. ЇЇ спеціально сконструйований процесор мав тактову частоту 12.5 нс. і налічував 12 конвейерних функціональних пристроїв. Пристрій оперативної пам’яті мав час доступу 50 нс. і являв собою 106 64 розрядних слів, поділених на 16 банків. Cray-1 демонстрував середню продуктивність 130 Мфлопс при піковій потужності 160 Мфлопс, що було абсолютним рекордом на той час серед всіх ЕОМ, включно з мультипроцесорними. Принциповим моментом архітетктури Cray-1 були 8 векторних регістрів по 64 64-розрядних слів для зберігання чисел з плаваючою точкою (ЗПТ). Швидкість обміну з пристроєм оперативної пам’яті (ОЗП) складав 320 Мслів/сек. У 80-ті роки фірма розширила цю архітектуру до мультипроцесорної з спільною пам′ятью в системах Cray X-MP та Cray Y-MP. Але головні архітектурні ідеї для високої продуктивності залишилися тими ж: розшарована память; швидкий процесор (9,5 нс. тактова частота); велика кількість регістрів скалерних; 12 функціональних пристроїв, які складають чотири групи − адресні, скалярні, векторні, обробки ЗПТ, і працюють паралельно. У 90-і роки модель Cray-YMP C90 вже налічує 16 процесорів (з тактовою частотою 4,1 нс.), де паралелізм обчислень досягається на чотирьох рівнях. Перший рівень − конвейерезація виконань операцій. Всі операції виконання команд: звернення до основної пам′яті, обробка адрес, власне виконання команди − є конвейерними за рахунок буферних регістрів між ОЗП і функціональними пристроями. На другому рівні паралелізм реалізується паралельними обчисленнями на різних функціональних пристроях. Це дозволяє виконувати розпаралелювання обчислень виразів на зразок: x=a*(в+c)/(d+e). Як

очевидно з аналізу інформаційної залежності, підвирази a*(в+c) і (d+e) можуть обчислюватись незалежно, а значить, при наявності додаткового обладнання, і паралельно.

Третій рівень складає векторна обробка масивів. Саме на цьому рівні забезечується найбільше прискорення обчислень − час виконання обробки може бути на порядок меншим, ніж для відповідних послідовних операцій. Виграш одержується за рахунок групування даних і операцій.Крім того, векторні операції, що використовують різні функціональні пристрої − теж виконуються паралельно.

DE1-SoC - комплект розробки, представляє собою потужну апаратну платформу, побудовану на основі ALTERA System-on Chip (SoC) FPGA, яка є комбінацією новітнього вбудованого двоядерного процесора Cortex-A9 і провідною в галузі програмованої логіки. Тепер розробники можуть використовувати потужність конфігурованної системи в поєднанні з високою продуктивністю і низьким споживанням процесора.

3.2 Створення класичного вигляду SMP-системи

На сучасному етапі метакомп′ютерні системи розвиваються у напрямку динамічних мережних утворень (аж до глобальних) з змінною конфігурацією та асинхронною роботою компонент, націленої на виконання задач зіслабозв′язаними частинами, наприклад переборного чи пошукового типу.Метакомп′ютерна система забезпечує прозорий контрольований доступ користувачів до системи через Internet та прозоре підключення не використовуваних (idle) обчислювальних ресурсів до мета-комп′ютера. Відомими прикладами систем є Globus і Legion [9]. Одним з перших прикладів задач, для розв’язування яких знадобились обчислювальні ресурси у глобальному масштабі стала RSA Challenges − розшифровка кодів секретності. Поточна версія задачі з розшифровки тексту, закодованого 64-бітним ключем має справу з 264 комбінаціями. У проекті залучено понад 200тис. учасників; досягнута продуктивність 75 млрд.ключів/ сек. В іншому проекті SETI@home (Search for Exraterrestial Intelligence) − пошук позаземних цивілізацій − зареєстровано

близько 1 млн. учасників. Серед прикладів подібних масштабних проектів − проект Оксфордського університету та фірми Intel (http:/www.intel.com/cure) зі створенню Інтернет-суперкомпютера IPPPP (Intel Philantropie Peer-to-Peer Program) для пошуку нових ліків від лейкемії то ракових захворювань.

Системи паралельних обчислень є багаторівневими системами. Традиційно виділяють три головні рівні: аппаратне забезпечення, системне програмне забезпечення (ПЗ) і прикладне ПЗ. В останній час між системним і прикладним ПЗ виділяють ще й проміжне ПЗ (middleware), а в його складі − системне проміжне ПЗ, що складається з ядра ОС та системних сервісів. Ядро реалізує функції нижнього рівня, що є базовими (в цілому − це управління ресурсами). Системні сервіси − це абстрактне бачення операційної системи з боку прикладної програми (абстрактна машина). Пара <менеджер ресурсів, абстрактна машина> власне і визначає те, що називається операційним середовищем.

Хронологічно першим у розвитку паралельного ПЗ був етап централізованих систем (ЦС), що були тісноз′язаними у апаратному і програмному розумінні. За хронологією другим виник етап паралельних операційних середовищ у мережах − мережних систем (МС), що характеризуються слабкозв′язаністю у обох розумінях. На подальших етапах виникали розподілені системи (РС)− слабкозв′язані в апаратному і тісно зв′язані в програмному розумінні, і нарешті, кооперативно-автономні системи (КАС)− що характеризуються послабленням централізованості розподілених систем. Отже ЦС → РС → КАС → МС − логічна лінія розвитку паралельних систем у напрямку їх децентралізації.

Ядро ОС є монолітним програмним утворенням, що реалізує низькорівневі функції управління. Принцип мінімального ядра зменшує до мінімуму машинно-залежний код. Універсальне мінімальне ядро називається мікроядром (microkernel). Архітектуру мікроядра утворюють пластформно-залежне мінімальне ядро та апаратно-незалежні виконавчі модулі (executives) з їх прикладними прграмними інтерфейсами. Значення мікроядра полягає у його універсальності, що дає необхідні і достатні умови для побудови будь-якої операційної системи чи підсистеми з мінімальними затратами. Прикладами

відомих мікроядер є IBM Microkernel та Microsoft Windous NT. Мікроядро, на відміну від загального випадку ядра, структурується відповідно до особливостей цільових архітектур. Тому є проміжний шар апаратної абстракції HAC (Hardware Abstraction Layer), що полегшує перенесення (мобільність) коду на різні платформи.

3.3 Створення модуля когерентності кешу

Когерентність кешу (англ. Cache coherence) - це властивість кешу, що означає цілісність даних, що зберігаються в локальних кешах для роздільного ресурсу. Когерентність кешу - окремий випадок когерентності пам'яті.

Коли клієнти в системі використовують кешування загальних ресурсів, наприклад, пам'яті, можуть виникнути проблеми з суперечливістю даних. Це особливо справедливо по відношенню до процесорів в багатопроцесорній системі. На малюнку «Кеш для ресурсу пам'яті» зображено, що клієнт у верхній частині має копію блоку пам'яті з попереднього читання, а нижній клієнт змінює блок пам'яті, копія даних в кеші верхнього клієнта стає застарілою, якщо не використовуються будь-які повідомлення про зміну або перевірки змін. Когерентність кешу призначена для управління такими конфліктами і підтримкою відповідності між різними кешами.

Когерентність визначає поведінку читань і записів в одному і тому ж місці пам'яті. Кеш називається когерентним, якщо виконуються наступні умови:

1. Якщо процесор P записує значення в змінну X, то при наступному зчитуванні X він повинен отримати раніше записане значення, якщо між записом і читанням X інший процесор не здійснював запис в X. Це умова пов'язана із збереженням порядку виконання програми, це повинно виконуватися і для однопоточної архітектури.

2. Операція читання X процесором P 1 {\displaystyle P_{1}} , наступна після того, як інший процесор P 2 {\displaystyle P_{2}} здійснив запис в X, повинна повернути записане значення, якщо інші процесори не змінювали X між двома операціями. Ця умова визначає поняття когерентної видимості пам'яті.
3. Записи в одну і ту ж комірку пам'яті повинні бути послідовними. Іншими словами, якщо два процесори записують у змінну X два значення: A, потім B - не повинно трапитися так, щоб при зчитуванні процесор спочатку отримував значення B, а потім A.

3.3.1. Архітектура МКК

МПС на основі МП частіше за все використовуються в якості вбудованих систем для вирішення завдань управління об’єктом. Важливою особливістю даного застосування є робота в реальному часі, тобто забезпечення реакції на зовнішні події протягом певного часового інтервалу. Такі вбудовувані вузькоспеціалізовані керуючі МПС, виконані у вигляді окремих мікросхем, які працюють у реальному масштабі часу, називають мікроконтролерами.Мікроконтролер (МК) – це функціонально закінчена МПС, виготовлена на одній НВІС (надвеликій інтегральній схемі). Мікроконтролер містить у собі: процесор, ОЗП, ПЗП, порти вводу-виводу для підключення зовнішніх пристроїв, модулі вводу аналогового сигналу АЦП, таймери, контролери переривання, контролери різних інтерфейсів і т.д. Найпростіший МК представляє собою ВІС площею не більше 1 см2 і всього з вісьмома виходами. Контролери, як правило, створюються для рішення якоїсь окремої задачі або групи близьких задач. Вони зазвичай не мають можливостей підключення додаткових вузлів і пристроїв, наприклад, великої пам’яті, засобів вводу/виводу. Їх системна шина часто недоступна користувачеві. Структура контролера проста і оптимізована під максимальну швидкодію. У більшості

випадків програми, що виконуються, зберігаються в постійній пам’яті і не змінюються. Конструктивно контролери випускаються в одноплатному варіанті.

Розглянемо основні типи архітектури та системи команд процесорів МК. У сучасних МК використовуються такі типи системи команд процесорів:

- RISC – (Reduce Instruction Set Commands) архітектура зі скороченим набором команд.

- CISC – (Complex Instruction Set Commands) традиційна архітектура з розширеним набором команд.

- ARM – (Advanced RISC – machine) вдосконалена RISC архітектура. Головне завдання RISC архітектури забезпечення найвищої продуктивності процесора. Її характерними ознаками є:
- Мала кількість команд процесора (кілька десятків);
- Кожна команда виконується за мінімальний час (1-2 машинних цикла, такта).

- Максимально можлива кількість регістрів загального призначення процесора (кілька тисяч); - Збільшена розрядність процесора (12,14,16 біт). Сучасна RISC архітектура включає, як правило, тільки останні 3 пункти, тому що за рахунок щільності компонування ВІС стало можливим реалізувати велику кількість команд. У сучасних 32-розрядних МК використовують ARM архітектуру (розширена RISC архітектура з суперскороченням команд THUMB).

До основних архітектур організації мікроконтролерів входять:

1. Фон-Нейманівська (Прінстонська);

2. Гарвардська архітектура;

3. Модифікована (оптимізована) Гарвардська CISC архітектура. Основні переваги Фон-Нейманівської архітектури (рис. 5.1):

- Простота апаратної реалізації;

- Універсальність виконання команд. Головні ознаками Гарвардської архітектури організації контролера:

- Реалізація пам'яті у вигляді різних пристроїв – різних фізичних типів памяті (для програм ПП і пам'яті для даних ПД);

- Використання двох паралельно працюючих незалежних шин для читання даних і команд.

Особливе місце займають на сьогодні мікропроцесорні системи на основі персонального комп'ютера. Персональний комп'ютер має розвинені засоби програмування, що істотно спрощує завдання розробника. До того ж він може забезпечити найскладніші алгоритми обробки інформації. VHDL – мова опису апаратури інтегрованих схем. VHDL є базовою мовою при розробці апаратури сучасних обчислюваних систем. Був розроблений в 1983 р. по замовленню Міністерства оборони США з метою формального опису логічних схем для всіх етапів розробки електронних систем, починаючи з модулів мікросхем і закінчуючи великими обчислювальними системами. Програма на VHDL складається із одного чи декількох файлів. В одному файлі розміщується одна чи декілька пар елементів "об’єкт проекту – архітектура об’єкту". Об’єкт проекту – це частина інтерфейсу, в якій вказана інформація про те, як ввімкнути даний об’єкт в середині іншого об’єкту, який знаходиться на більш високому рівні ієрархії. А в архітектурі об’єкту описується алгоритм його функціонування. У одного об’єкт може містити декілька архітектур, відповідаючи різним алгоритмам функціонування, проектам на різних етапах проектування. В прикладах цього і подальшого коду курсивним шрифтом вказані коментарі, напівжирним – ключові слова даної мови. Ключове слово port відкриває опис входів-виходів (портів) об’єкти. Слово inвказує на вхід, a out– на вихід. BIT– це тип порту, який, згідно визначенню цього типу в пакеті standard, приймають значення 0 і 1. Розділ архітектури складається із декларативної і описової частин. В декларативний частині оголошуються використані в середині сигналу, константи спеціальні (власні) типи, атрибути, процедури і функції. Описова частина складається із списку паралельних операторів. Всі паралельні оператори виконуються одночасно, їх порядок в списку не має значення. Verilog*, Verilog HDL – ще одна мова описання архітектури, що використовується для описання і моделювання електронних систем. Verilog була створена в 1984 році фірмою Automated Integrated Design Systems (пізніше перейменований на GatewayDesignAutomation) як власний засіб моделювання. Після поглинання останньої фірмової Caddence, мова почала отримувати все більший попит серед

розробників і стала не менш популярною, чим VHDL. Синтаксис Verilog дуже схожий га синтаксис мови С, що спрощує його освоєння. Препроцесор Verilog дуже схожий на препроцесор мови С, і основні управляючі конструкції if, while та інші подібні однойменні конструкції мови С. Програма на Verilog зазвичай складається із декількох модулів. У заголовку модуля описується список портів. У тілі модуля описується його функціональність. У проекті, особливо в складному, буває багато модулів, з'єднаних між собою. Перш за все, потрібно помітить, що зазвичай в проекті завжди є один модуль самого верхнього рівня (top level). Він складається із декількох інших модулів. Ті в свою чергу можуть містити ще модулі і так далі. Необов'язково, щоб всі модулі були написані на одній мові описання апаратури. Зовсім навпаки. Досить зручно і наочно мати модуль самого верхнього рівня виконаним у вигляді схеми, що складається із модулів більш низького рівня. Ці модулі можуть бути написані різними розробниками,на різних мовах (Verilog, VHDL) і навіть виконанні у вигляді схеми. System Verilog являє собою надмножину мови Verilog, але з великими можливостями для верифікації і моделювання розробок. Примітка – при називанні програмних об'єктів (проектів, інтерфейсів, архітектур, пакетів, сигналів, змінних, констант, компонентів і т.д.) в коді потрібно притримуватись правил. В спільному випадку дешифратор (декодер) – це пристрій, що перетворює цифровий сигнал в якому-небудь кодуванні в іншу, не закодовану форму. Класичний дешифратор представляє собою пристрій для перетворення N-розрядного позиційного двійкового коду в одиничний вихідний сигнал на одному із 2^N виходів. При кожній вихідній комбінації сигналів на одному із виходів з'являється логічна 1. Таким чином, по одиничному сигналів на одному із виходів можна оцінити вхідну кодову комбінацію. Наприклад, пристрій повинен мати чотири виходи. Для кожного виходу записуємо логічний вираз. За цією системою виразів не складно побудувати схему потрібного дешифратора.

Процесор Nios II - це 32-розрядна процесорна RISC архітектура для застосування у вбудованих системах (soft processor або програмний процесор), розроблена спеціально для ПЛІС фірми Altera. Nios II є розвитком архітектури

Nios і знаходить застосування в різних вбудованих додатках - від систем цифрової обробки сигналів до різноманітних пристроїв управління.

Існують 3 різні версії конфігурації процесора.

• Nios II / f (fast) - версія, призначена для досягнення максимальної продуктивності. Конфігурація має широкий набір опцій для оптимізації процесора по продуктивносі.

• Nios II / s (standart) - стандартна версія. Дана конфігурація вимагає менше ресурсів для реалізації і характеризується меншою продуктивністю.

• Nios II / e (economy) - економічна версія. Дана конфігурація вимагає найменшу кількість ресурсів для реалізації, але володіє обмеженим набором можливостей.

Таким чином, забезпечення узгодженості кеш-пам’ятки є поєднанням апаратного та програмного рішень.

3.3.2. Реалізація модуля когерентності кешу у Quartus II

Процесор Nios II має RISC архітектуру, в якій всі арифметичні і логічні операції виконуються над операндами, що зберігаються в регістрах загального призначення. Обмін інформацієюміж регістрами і пам'яттю здійснюється шляхом виконання спеціальних команд «Load» і «Store». Довжина машинного слова процесора Nios II становить 32 біта. Таку ж розрядність мають і його регістри. процесор використовує роздільні шини для команд і даних, тобто побудований за гарвардської архітектури. Структурна схема процесора Nios II представлена на рис. 3.1.

Процесор може функціонувати в двох режимах - в режимі супервізора (процесор перемикається в даний режим після надання сигналу скидання), де йому дозволяється виконувати всі інструкції і здійснювати будь-які функції, і в режимі користувача, в якому обмежена можливість виконання

певних інструкцій системного призначення (доступний тільки при наявності модуля управління пам'яттю MMU або модуля захисту пам'яті MPU).

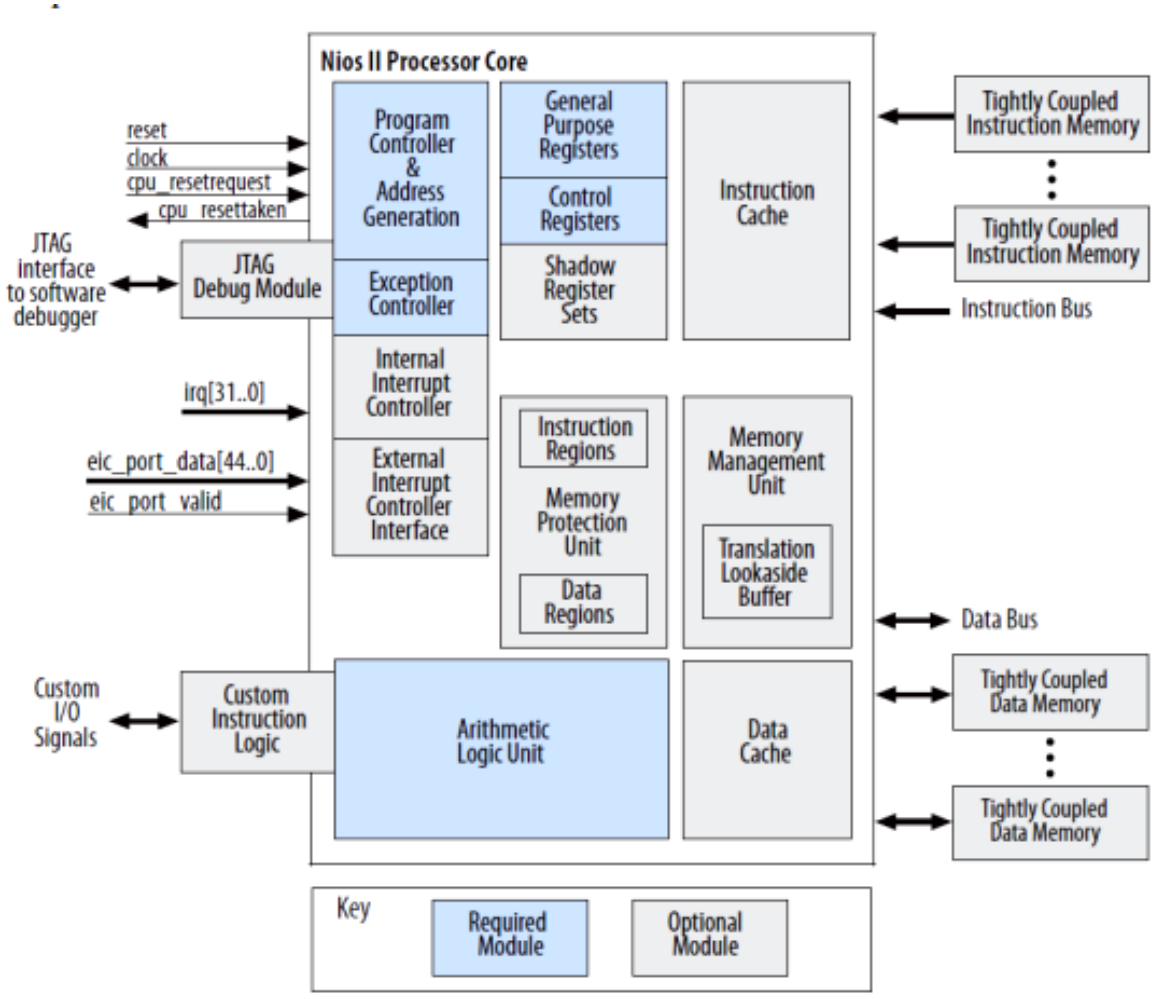

Рис. 3.1 - Структурна схема процесора Nios II

Архітектурою процесора Nios II підтримується однорідний регістровий файл, що складається з тридцяти двох регістрів загального назначення (РОН) і до тридцяти двох керуючих регістрів. Всі регістри мають імена, які розпізнаються асемблером. Регістри загального призначення і керуючі регістри, представлені в таблицях 6.1 і 6.2, відповідно. До складу процесора може входити до 32 управляючих регістрів. Їх загальна кількість залежить від наявності модулів захисту пам'яті або управління пам'яттю. Опціонально, для прискорення контекстних перемикань, процесор може мати до 63 наборів тіньових (shadows) регістрів. Для роботи з ними

процесор має дві спеціальні інструкції, що дозволяють переміщати дані між наборами регістрів.

Таблиця 3.1

Регістри загального призначення процесора Nios II

Регістр	Ім’я	Призначення
r0	zero	Регістр нуля, завжди містить значення 0x00000000
r1	at	Часовий регістр, використовується асемблером
r2, r3	v0, v1	Повертання значення
r4..r7	a0, a3	Регістр аргумента
r8..r15	t0..t7	Часові регістри
r16..r23	s0..s7	Збережні регістри
r24	et	Часовий регістр для обробки винятків
r25	bt	Часовий регістр, використовуваний при налагодженні
r26	gp	Глобальний покажчик
r27	sp	Покажчик стека
r28	fp	Покажчик кадра
r29	ea	Адреса повернення з винятків (недоступний в режимі користувача)
r30	ba	Повернення з контрольної точки (використовується тільки модулем налагодження JTAG)
r31	ra	Адреса повернення при виклику підпрограм

Поточний використовуваний набір тіньових регістрів відображає поле CRS статусного регістра. Для адресації процесор Nios II використовує 32-бітову адресу з побайтовою адресацією (порядок Little-endian). За допомогою відповідних команд можна записувати або зчитувати слова (32 біта), півслова (16 біта) і байти даних (8 біт). У процесорі Nios II визначені кілька способів адресації. Безпосередня адресація - в команді присутня 16-бітний операнд (він може доповнюватися до 32 розрядів при виконанні арифметичних операцій); реєстрова адресація - операнди знаходяться в регістрах процесора; відносна реєстрова адресація- ефективний адреса операнда обчислюється підсумовуванням вмісту регістра і знакового 16-розрядного числа, що знаходиться в самій команді, яка визначає зміщення; непряма регістроваадресація - вміст регістра є ефективним адресом операнда (цей спосіб еквівалентний попередньому при нульовому зміщенні), абсолютна адресація - 16-бітний абсолютна адреса операнда визначено зміщенням щодо регістра нуля.

3.3.3. Процес обробки переривань

Опрацьовувач переривань (англ. interrupt handler), також відомий як процедура опрацювання переривань (англ. interrupt service routine, ISR) або обробник переривань, це функція зворотного виклику в операційній системі або драйвері пристрою, виконання якої спричиняється отриманням переривання. Опрацьовувачі переривань мають безліч функцій, які різняться відповідно до причини виникнення переривання і швидкодії опрацьовувача. Опрацьовувач переривань є низькорівневим двійником опрацьовувача подій (повідомлень). Ці опрацьовувачі ініціюються або апаратними перериваннями або інструкціями переривання на програмному рівні, і використовуються для обслуговування апаратних пристроїв і переходів між захищеними режимами операцій такими як системні виклики.

В деяких ОС - Linux, Microsoft Windows і деяких інших, опрацьовувачі переривань поділені на дві частини: Опрацьовувачі переривань першого рівня (англ. First-Level Interrupt Handler, FLIH) і Опрацьовувачі переривань другого рівня (англ. Second-Level Interrupt Handlers, SLIH). FLIH також відомі як тверді/швидкі опрацьовувачі переривань (англ. hard/fast interrupt handlers), а SLIH також відомі як м'які/повільні опрацьовувачі переривань (англ. soft/slow interrupt handlers), відкладений виклик процедури. FLIH виконує щонайменше платформо-залежну обробку переривання. У відповідь перериванню відбувається перемикання контексту і завантажується та виконується код для переривання. Завданням для FLIH є швидка обробка переривання, або запис платформо-залежних критичних даних, які доступні лише під час переривання, і планування виконяння SLIH для подальшого опрацювання переривання. FLIH спричиняють тремтіння в виконанні процесу. FLIH також маскують переривання. Зменшення тремтіння найважливіше для операційних систем реального часу, бо вони мають відповідати вимозі виконання певного коду в узгоджений відтинок часу. Для зменшення тремтіння і зменшення ймовірності втрати даних через приховані переривання, програмісти намагаються зменшити час виконання FLIH, виносячи весь можливий код у SLIH. Зі швидкістю сучасних комп'ютерів, FLIH можуть виконати усе пристрій і платформо-залежне опрацювання, і використати SLIH для подальшого платформо-незалежного довго-тривалого опрацювання. FLIH, що обслуговують апаратну частину зазвичай маскують своє переривання до завершення свого виконання. Особливий FLIH, який викриває пов'язане з ним переривання до свого завершення називається повтороновикористовним обробником переривань. Повторновикористовний обробник переривань може спричинити переповнення стека через багаторазові вкладені викликі. SLIH завершує завдання довгої обробки переривань подібно до процесу. SLIH або має виділений ядром потік для кожного опрацьовувача, або виконується в пулі робочих потоків ядра. SLIH може мати великий час виконання, і тому, зазвичай, планується подібно до потоків і процесів. В різних системах FLIH і SLIH іменуються по різному. В Windows FLIH називається опрацьовувачем переривання, а SLIH — відкладеним викликом процедури. В Linux, FLIH

називається upper half, а SLIH - lower half або bottom half. Це різниться від іменування використовного в інших Unix-подібних системах, де обидва є частиною bottom half. При виникненні переривання мікроконтролер зберігає в стеку вмісту лічильника команд і завантажує в нього адресу відповідного вектора переривання, в якому, як правило, міститься команда безумовного переходу до підпрограми обробки переривання. Останньою командою підпрограми-обробника переривань має бути команда повернення reti, яка забезпечує повернення в основну програму шляхом відновлення значення заздалегідь збереженого лічильника команд. Обробка переривань відбувається відповідно до загальних принципів обробки переривань у мікропроцесорній техніці. Модуль переривань приймає запити переривання й організовує перехід до виконання визначеної програми. Запити переривання можуть надходити як від зовнішніх джерел, так і від внутрішніх джерел модулів мікроконтроллера. Для прийому запитів від зовнішніх джерел використовуються виводи портів вводу/виводу, для яких ця функція є альтернативною. Джерелами запитів зовнішніх переривань також можуть бути будь-які зміни зовнішніх сигналів на деяких спеціально виділених лініях портів вводу/виводу.

Системні програми, адреси яких зберігаються у векторах переривань, в більшості своїй є всього лише диспетчерами, що відкривають доступ до великих груп програм, що реалізовують системні функції. Так, відеодрайвер **BIOS** (вектор 10h) включає програми зміни відеорежиму, управління курсором, завдання колірної палітри, завантаження шрифтів і багато інших. Особливо характерний в цьому відношенні вектор 21h, через який здійснюється виклик практично всіх функцій **DOS**: введення з клавіатури і виводу на екран, обслуговування файлів, каталогів і дисків, управління пам'яттю і процесами, служби часу і так далі Для виклику необхідної функції треба не тільки виконати команду int з відповідним номером, але і вказати системі в одному з регістрів (для цієї мети завжди використовується регістр AH) номер функції, що викликається. Іноді для "багатофункціональних" функцій доводиться указувати ще і номер підфункції (у регістрі AL).

Таким чином, важливо, щоб розробник системи забезпечив, що під час налаштування системи, МКК отримує номер переривання 16.

3.4 Вимоги до програмного забезпечення

Вимоги до програмного забезпечення для підтримки когерентності кеш-пам'яті, як правило, передбачають створення точки бар'єру ініціалізації після ініціалізації процесора для всіх AP (процесори, не позначені як BSP). Це змушує APs чекати, поки не завершиться глобальна ініціалізація. Це схоже на Intel SMP-метод утримання AP в скиданні, поки BSP не завершить ініціалізацію. У цьому випадку глобальна ініціалізація складається в основному з встановлення ISR та включення МКК.

Вимоги до програмного забезпечення — набір вимог щодо властивостей, якості та функцій програмного забезпечення, що буде розроблено, або знаходиться у розробці. Вимоги визначаються в процесі аналізу вимог та фіксуються в специфікації вимог, діаграмах прецедентів та інших артефактах процесу аналізу та розробки вимог. Розробка вимог до програмної системи може бути розділена на декілька етапів: знаходження вимог (збір, визначення потреб зацікавлених осіб та систем); аналіз вимог (перевірка цілісності та закінченості); специфікація (документування вимог); тестування вимог. В рамках Уніфікованого процесу розробки вимоги представляються у вигляді кількох необов'язкових документів: модель випадків використання (прецедентів) (англ. Use Case Model) – набір типових сценаріїв використання системи, описує функціональні (поведінкові) вимоги; додаткова специфікація (англ. Supplementary Specification), містить нефункціональні вимоги, такі як вимоги до надійності, продуктивності, документування, підтримки, ліцензування тощо; словник термінів (англ. Glossary), визначає важливі терміни і визначення, може включати концепцію словника даних, який фіксує вимоги, пов'язані з даними, такими як правила верифікації, прийнятні значення тощо, бачення (англ. Vision), узагальнює найважливіші високорівневі ідеї та вимоги, покладені в основу розробки системи. Це короткий оглядовий документ для швидкого ознайомлення з проектом, бізнес-

правила (англ. Business Rules). Бізнес-правила або правила предметної області описують вимоги або політики, які виходять за рамки одного проекту, наприклад, політика компанії, організація бухобліку, державні норми оподаткування, закони. Можуть бути представлені у додатковій специфікації або окремим артефактом.

3.5 Тестування

3.5.1 Тест спільної пам’яті

Спільна пам'ять (англ. Shared memory) — регіон комп'ютерної пам'яті, до якої мають доступ кілька програм водночас. Такий доступ може організовуватись з метою зв'язку або передачі даних між програмами (чи їх потоками виконання), коли зайве копіювання даних небажане. Залежно від контексту, програми можуть запускатись як на одному процесорі, так і на кількох.

Стосовно апаратної реалізації комп'ютера, термін спільна пам'ять означає блок оперативної пам'яті, до якого мають доступ кілька центральних процесорів (ЦП) у багатопроцесорних комп'ютерних системах.

Існує кілька різновидів архітектури систем зі спільною пам'яттю:

- однорідний доступ до пам'яті (англ. uniform memory access): рівномірний доступ до пам'яті всіма процесорами;
- неоднорідний доступ до пам'яті (англ. non-uniform memory access): час доступу до пам'яті залежить від розташування пам'яті стосовно процесора;
- архітектура пам'яті із використанням лише кешу (англ. cache-only memory architecture): локальні блоки пам'яті для процесорів у кожному вузлі використовуються як кеш-пам’ять, а не як основна.

Системи із розподіленою пам'яттю відносно прості для програм так як усі процеси розподіляють єдине представлення даних і комунікація між процесорами може бути такою ж швидкою як доступ до пам'яті у одному місці. Проблема із

системами із розподіленою пам'яттю полягає у тому, що багато центральних процесорів потребують швидкого доступу до пам'яті та швидше за все, доступу до кешу процесора, тому з'являються такі ускладнення:

- збільшення часу доступу: коли кілька процесорів намагаються отримати доступ до одного блоку пам'яті, це викликає незгоду. Системи із розподіленою пам'яттю не можуть добре маштабуватись. Більшість із них мають 10, або менше процесорів;

- відсутність узгодженості даних: всякий час, коли один кеш оновлюється інформацією що може бути використана іншими процесорами, зміна повинна відбутись у інших процесорах інакше, різні процесори будуть працювати із різними даними. Така когерантність кешу, коли вона добре працює, може забезпечити надзвичайно високопродуктивний доступ до розподіленої між багатьма процесорами інформації. З іншої сторони, інколи вони можуть бути перевантажені і стати надто вузьким місцем для продуктивності.

У випадку з Heterogeneous System Architecture (архітектура процесорів, у якій різні типи процесорів, такі як (GPU), або (CPU) поєднуються за допомогою розподіленої пам'яті), CPU-модуль керування пам'яттю (MMU) та GPU-блок управління пам'яттю для вводу/виводу (IOMMU) повинні ділити певні характеристики, як загальний адресний простір. Альтернативами до розподіленої пам'яті є distributed memory і distributed shared memory, які мають той же список проблем.

3.5.2 Тест мультизапису

У програмному забезпеченні, розподілена пам'ять представляє собою:

- метод взаємодії між процесами, тобто шлях обміну даними між програмами які працюють одночасно. Один процес створює область у RAM-пам'яті до якої мають доступ інші процеси;

- метод збереження простору пам'яті завдяки направленню доступу до того, що є копіями даних замість одиничних екземплярів, використовуючи планування віртуальної пам'яті. Найчастіше це використовується для розподілених бібліотек та для execute in place-систем.

Оскільки обидва процеси можуть отримати доступ до області розподіленої пам'яті як для звичайної робочої пам'яті, це дуже швидкий шлях до комунікації. З іншої сторони, це менш маштабовано, для прикладу, взаємодіючі процеси повинні бути запущені на одній машині, і необхідно бути обережним, щоби уникнути проблем, у випадку, якщо процеси, що використовують розподілену пам'ять, виконуються на окремих про ЦПУ і архітектура не є кеш-когерентною.

POSIX забезпечує стандартизоване API для використання розподіленої пам'яті, POSIX Shared Memory. Воно використовує функцію shm_open із sys/mman.h. міжпроцесорна комунікація POSIX (частина POSIX:XSI розширення) функції для роботи із розподіленою пам'яттю : shmat, shmctl, shmdt and shmget.

Unix System V чудово забезпечує API для розподіленої пам'яті. Воно використовує shmget з sys/shm.h. BSD-системи забезпечують "anonymous mapped memory" яка може бути використана кількома процесорами. Розподілена пам'ять, створена за допомогою shm_open є стійкою. Вона залишається у процесі, поки не буде явно видалена за допомогою процесу. У цьому є недолік, який полягає у тому, що якщо процес вийде з ладу і не вдається очистити загальну пам'ять він буде залишатись активним до завершення роботи системи. POSIX також забезпечує mmap API для відображення файлів у пам'ять; відображення може бути розподіленим, дозволяючи вміст пам'яті, в ролі розподіленої. Дистрибутиви Linux які базуються на 2.6 та новіших версіях kernel пропонують /dev/shm як розподілену пам'ять у формі RAM-диску, конкретніше, як каталог, доступний для запису (директорія, у якій кожен користувач системи може створювати файли) що зберігається у пам'яті. Дистрибутиви, які базуються на RedHat та Debian включають це за умовчанням. Підтримка цього типу RAM-пам'яті не є обов'язковим у файлі конфігурації kernel.

3.5.3 Результати тестування

Програма тестування спільної пам'яті виконувалася двічі в кожній системі, один раз із увімкненим модулем когерентності і один раз із вимкненим(щоб гарантувати, що очікувана різниця в результатах була обумовлена роботою МКК). Тестованими системами були однопроцесорна та 4-, 8-, 16-процесорні (гуперкуби 2, 3 та 4 степенів) системи SMPOPC двох класів: без МКК, з прототипом МКК.

У таблиці 3.1 наведено результати тестування. Як можна бачити, результати збігаються з очікуваними, вказуючи на те, що розроблений модуль не здатний повністю забезпечити узгодженість кеш-пам'яті в системі.

Таблиця 3.1 – Результати тестування

Кількість процесорів	Тест спільної пам’яті		Тест мультизапису	
	Без МКК	З МКК	Без МКК	З МКК
1	Pass	Pass	Pass	Pass
4	Fail	Pass	Pass	Pass
8	Fail	Pass	Pass	Fail
16	Fail	Pass	Fail	Fail

Систе́мне тестува́ння є одним з рівнів тестування програмного забезпечення. Системне тестування тестує інтегровану систему для перевірки відповідності всім вимогам. Перевірка повноти та правильності документації користувача є важливою частиною системного тестування. Всі тестові комбінації повинні розроблятися тільки з використанням документації користувача.

3.6 Висновки

У цьому розділі було розроблено мультипроцесорну систему загального призначення на основі топології «гіперкуб». Розглянуто програмно-апаратні засоби що дозволили розробити та змоделювати роботу такої системи.

Для масштабованості розробили окремий процесорний модуль, що дозволив гнучко змінювати кількість процесорів в системі і будувати різностепеневі гіперкуби.

Також, була описана проблема когерентності кеш-пам’яті у таких системах, для вирішення цієї проблеми був створений модуль когерентності кеш-пам’яті. Було здійснено тестування розробленої мультипроцесорної системи як з модулем так і без.

ВИСНОВКИ

В роботі представлено розробку мультипроцесорної системи загального призначення на основі топології «тор».

У ході роботи було описано важливість створення такої системи, розглянуто основні типи архітектур мультипроцесорних систем. Шляхом порівняння було обрано симетричну архітектуру, адже вона давала ширші можливості як для розробки так і для масштабування системи. Також розглянуто уже існуючі рішення, розглянуто основні проблеми готових систем, внаслідок чого сформульовано вимоги до розроблювальної системи.

Тор – це топологія що дає одні з найкращих характеристик швидкості та продуктивності, значною її перевагою є те, що вона підтримує велику кількість вузлів і легко масштабується, тому аналіз ведеться, фактично для трьох варіацій систем з 4-, 8- та 16- процесорами. При аналізі можливого програмно-апаратного забезпечення вибір впав на продукцію фірми Altera, як і в питаннях середовища розробки, так і в питаннях апаратного забезпечення та засобів візуалізації.

Спочатку було розроблено класичну модель SMP-системи, але вона є дуже проблемною у функціонуванні при використанні великої кількості процесорів, адже шина стає вузьким місцем. Для вирішення цієї проблеми було додано апаратний кеш кожному з процесорів, але це породило нову проблему – проблему когерентності кеш-пам’яті, тому було розроблено модуль когерентності кеш-пам’яті.

Для підтвердження правильності функціонування системи провелось тестування двома видами тестів – тестом спільної пам’яті та тестом мультизапису. Це показало, що розроблений модуль має певні недоліки у своїй архітектурі і може бути покращений. Сама ж система працює досить коректно та може використовуватись для виконання різних задач.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Vincke B., Elouardi A., Lambert A. Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems. *J Embedded Systems*. 2012. № 5. С. 28-34. DOI: https://doi.org/10.1186/1687-3963-2012-5

2. Tanveer M., Iqbal M. A., Azam, F. Using Symmetric Multiprocessor Architectures for High Performance Computing Environments. *International Journal of Computer Applications*, 2011, URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.4182&rep=rep1&type=pdf

3. Lv F., Cui, HM., Wang L. et al. Dynamic I/O-Aware Scheduling for Batch-Mode Applications on Chip Multiprocessor Systems of Cluster Platforms. *J. Comput. Sci. Technol*. 2014. № 29. С. 21–37. DOI: https://doi.org/10.1007/s11390-013-1409-2

4. V. Schwambach, S. Cleyet-Merle, A. Issard and S. Mancini, "Fast parallel application and multiprocessor design space exploration from sequential code," *International Conference on Hardware/Software Codesign and System Synthesis* (CODES+ISSS), 2015, pp. 163-172, DOI: 10.1109/CODESISSS.2015.7331379.

5. A. Mello, I. Maia, A. Greiner and F. Pecheux, "Parallel simulation of systemC TLM 2.0 compliant MPSoC on SMP workstations," *Design, Automation & Test in Europe Conference & Exhibition* (DATE 2010), 2010, pp. 606-609, DOI: 10.1109/DATE.2010.5457136.

6. C. Su, D. Li, D. S. Nikolopoulos, K. W. Cameron, B. R. d. Supinski and E. A. León, "Model-based, memory-centric performance and power optimization on NUMA multiprocessors," *IEEE International Symposium on Workload Characterization* (IISWC), 2012, pp. 164-173, DOI: 10.1109/IISWC.2012.6402921.

7. Peter Tröger, Andreas Polze. Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium. Potsdam : *Universitätsverlag Potsdam* 2012 p.57

8. Xu Zhou, Kai Lu, Xiaoping Wang, Xu Li, Exploiting parallelism in deterministic shared memory multiprocessing,Journal of Parallel and Distributed

Computing,2012,Volume 72, Issue 5,Pages 716-727, DOI: https://doi.org/10.1016/j.jpdc.2012.02.008.

9. K. Yu, D. Han, C. Youn, S. Hwang and J. Lee, "Power-aware task scheduling for big.LITTLE mobile processor," *International SoC Design Conference (ISOCC),* 2013, pp. 208-212, DOI: 10.1109/ISOCC.2013.6864009.

10. Tabatabaee, H., Akbarzadeh-T, M.R. & Pariz, N. Dynamic task scheduling modeling in unstructured heterogeneous multiprocessor systems. *J. Zhejiang Univ. - Sci*. 2014. № 15. С. 423–434. DOI: https://doi.org/10.1631/jzus.C1300204

11. H. Yun, G. Yao, R. Pellizzoni, M. Caccamo and L. Sha, "Memory Access Control in Multiprocessor for Real-Time Systems with Mixed Criticality," *24th Euromicro Conference on Real-Time Systems*, 2012, pp. 299-308, DOI: 10.1109/ECRTS.2012.32.

12. Otoom, M., Paul, J.M. Workload Mode Identification for Chip Heterogeneous Multiprocessors. Int *J Parallel Prog*. 2012. № 40. С. 184–224. DOI: https://doi.org/10.1007/s10766-011-0175-4

13. Cannella, E., Stefanov, T.P. Energy efficient semi-partitioned scheduling for embedded multiprocessor streaming systems. *Des Autom Embed Syst*. 2016. № 20. С. 239–266. DOI: https://doi.org/10.1007/s10617-016-9176-2

14. ALTERA Nios II Classic Processor Reference Guide, San Jose, 2016, URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2cpu_nii5v1.pdf

15. Du, J., Wang, Y., Zhuge, Q. et al. Efficient Loop Scheduling for Chip Multiprocessors with Non-Volatile Main Memory. *J Sign Process Syst*. 2013. № 71. С. 261–273. DOI: https://doi.org/10.1007/s11265-012-0703-5

16. Maruf, M.A., Azim, A. Requirements-preserving design automation for multiprocessor embedded system applications. *J Ambient Intell Human Comput* 2021. № 12. С. 821–833. DOI: https://doi.org/10.1007/s12652-020-02086-9

17. Kryzhanovsky, M.V., Malsagov, M.Y. Neuron network methods of task assignment in multiprocessing system. *Opt. Mem. Neural Networks*, 2010, 19, 213–219. DOI: https://doi.org/10.3103/S1060992X10030021

18. Dai Bui, Edward Lee, Isaac Liu, Hiren Patel, and Jan Reineke. Temporal isolation on multiprocessing architectures. *In Proceedings of the 48th Design Automation Conference. Association for Computing Machinery*, 2011, New York, NY, USA, 274–279. DOI:https://doi.org/10.1145/2024724.2024787

19. Awasthi, M., Nellans, D., Sudan, K. et al. Managing Data Placement in Memory Systems with Multiple Memory Controllers. Int *J Parallel Prog*. 2012. № 40. C. 57–83. DOI: https://doi.org/10.1007/s10766-011-0178-1

20. Kumar, B.K., Thanikachalam, A., Kanakasabapathi, V. et al. Performance analysis of a multiprogramming–multiprocessor retrial queueing system with orderly reattempts. Ann Oper Res 2016. № 247. C. 319–364. DOI: https://doi.org/10.1007/s10479-015-2005-3

21. Asad, A., Dorostkar, A. & Mohammadi, F. A novel power model for future heterogeneous 3D chip-multiprocessors in the dark silicon age. *J Embedded Systems*. 2018. T. 2018, № 3. DOI: https://doi.org/10.1186/s13639-018-0086-1

22. Karavai, M.F., Podlazov, V.S. Distributed full switch as an ideal system area network for multiprocessor computers. *Autom Remote Control* 2013. № 74. C. 710–724. DOI: https://doi.org/10.1134/S0005117913040140

23. Zhao, J., Xu, C., Zhang, T. *et al.* BACH: A Bandwidth-Aware Hybrid Cache Hierarchy Design with Nonvolatile Memories. *J. Comput. Sci. Technol.* 2016. № 31. C. 20–35. DOI: https://doi.org/10.1007/s11390-016-1609-7

24. Regnier, P., Lima, G., Massa, E. et al. Multiprocessor scheduling by reduction to uniprocessor: an original optimal approach. Real-Time Syst 2013. № 49. C. 436–474. DOI: https://doi.org/10.1007/s11241-012-9165-x

25. Vincke, B., Elouardi, A. & Lambert, A. Real time simultaneous localization and mapping: towards low-cost multiprocessor embedded systems. *J Embedded Systems*. 2012. № 5. C. 2012. DOI: https://doi.org/10.1186/1687-3963-2012-5

26. J. Happe, H. Groenda, M. Hauck and R. H. Reussner, "A Prediction Model for Software Performance in Symmetric Multiprocessing Environments," *Seventh International Conference on the Quantitative Evaluation of Systems*, 2010, pp. 59-68, doi: 10.1109/QEST.2010.15.

27. Furugyan, M.G. Computation planning in multiprocessor real time automated control systems with an additional resource. *Autom Remote Control* 2015. № 76. С. 487–492. DOI: https://doi.org/10.1134/S0005117915030121

28. Nunez-Yanez, J., Amiri, S., Hosseinabady, M. et al. Simultaneous multiprocessing in a software-defined heterogeneous FPGA. *J Supercomput* 2019, 75, 4078–4095 . DOI: https://doi.org/10.1007/s11227-018-2367-9

29. Karavay, M.F., Podlazov, V.S. An extended generalized hypercube as a fault-tolerant system area network for multiprocessor systems. *Autom Remote Control* 2015. № 76. С. 336–352. DOI: https://doi.org/10.1134/S0005117915020137

30. Zoltan Majo and Thomas R. Gross. Memory system performance in a NUMA multicore multiprocessor. In Proceedings of the 4th Annual International Conference on Systems and Storage. *Association for Computing Machinery*, 2011, New York, NY, USA, Article 12, 1–10. DOI:https://doi.org/10.1145/1987816.1987832

31. Wang, HC., Woungang, I., Yao, CW. et al. Energy-efficient tasks scheduling algorithm for real-time multiprocessor embedded systems. *J Supercomput* 2012. № 62. С. 967–988. DOI: https://doi.org/10.1007/s11227-012-0771-0

32. Smirnov, G.S., Stegailov, V.V. Efficiency of classical molecular dynamics algorithms on supercomputers. Math Models Comput Simul. 2016. № 8. С. 734–743. DOI: https://doi.org/10.1134/S2070048216060156

33. Summit. Oak Ridge National Laboratory, 2015 URL: https://www.olcf.ornl.gov/summit/

34. Kao, CC., Lin, YC. Designs of Low Power Snoop for Multiprocessor System on Chip. *J Sign Process Syst* 2017. № 88. С. 83–89. DOI: https://doi.org/10.1007/s11265-016-1135-4

35. Jiang Y., Tian K., Shen X. Combining Locality Analysis with Online Proactive Job Co-scheduling in Chip Multiprocessors. Lecture Notes in Computer Science, vol 5952. Springer, Berlin, Heidelberg, 2010, DOI: https://doi.org/10.1007/978-3-642-11515-8_16

36. Furugyan, M.G. Some algorithms for analysis and synthesis of real-time multiprocessor computing systems. Program Comput Soft 2014. № 40. С. 21–27. DOI: https://doi.org/10.1134/S0361768814010034

37. Josephson, J., Ramesh, R. A novel algorithm for real time task scheduling in multiprocessor environment. Cluster Comput. 2019. № 22. С. 13761–13771. DOI: https://doi.org/10.1007/s10586-018-2083-5

38. Zhang, D., Lynch, B. & Dechev, D. Queue-Based and Adaptive Lock Algorithms for Scalable Resource Allocation on Shared-Memory Multiprocessors. *Int J Parallel Prog.* 2015. № 43. С. 721–751. DOI: https://doi.org/10.1007/s10766-014-0317-6

39. Garcia, V., Rico, A., Villavieja, C. et al. Adaptive Runtime-Assisted Block Prefetching on Chip-Multiprocessors. *Int J Parallel Prog*. 2017. № 45. С. 530–550. DOI: https://doi.org/10.1007/s10766-016-0431-8

40. Xu Zhou, Kai Lu, Xiaoping Wang, Xu Li, Exploiting parallelism in deterministic shared memory multiprocessing, *Journal of Parallel and Distributed Computing*, 2012, Volume 72, Issue 5, Pages 716-727,DOI: https://doi.org/10.1016/j.jpdc.2012.02.008.

41. Wang, H., Wang, R., Luan, Z. et al. Improving multiprocessor performance with fine-grain coherence bypass. Sci. China Inf. Sci. 2015. № 58. С. 1–15. DOI: https://doi.org/10.1007/s11432-014-5175-8

42. Furugyan, M.G. Synthesizing a Multiprocessor System for Scheduling with Interruptions and Execution Intervals. *J. Comput. Syst. Sci. Int*. 2019. № 58. С. 194–199. DOI: https://doi.org/10.1134/S1064230719020072

43. Chi Zhang, Xin Yuan and A. Srinivasan, "Processor affinity and MPI performance on SMP-CMP clusters," IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010, pp. 1-8, DOI: 10.1109/IPDPSW.2010.5470774.

44. Demaine, E.D., Ghodsi, M., Hajiaghayi, M. et al. Scheduling to minimize gaps and power consumption. *J Sched.* 2013. № 16. С. 151–160. DOI: https://doi.org/10.1007/s10951-012-0309-6

45. S. Jin, Z. Huang, R. Diao, D. Wu and Y. Chen, "Comparative Implementation of High Performance Computing for Power System Dynamic Simulations," in IEEE Transactions on Smart Grid, May 2017, vol. 8, no. 3, pp. 1387-1395, DOI: 10.1109/TSG.2016.2647220.

46. Anuradha, Dande. Simulation of Multiprocessor System Scheduling. MS thesis. 2014, URL: https://trepo.tuni.fi/handle/123456789/22144

47. A. Bastoni, B. B. Brandenburg and J. H. Anderson, "An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor EDF Schedulers," 31st IEEE Real-Time Systems Symposium, 2010, pp. 14-24, DOI: 10.1109/RTSS.2010.23.

48. Dimitriev, Y.K. Necessary and sufficient conditions for t-diagnosability of multiprocessor computer systems for various models of nonreliable testing established using the system graph-theoretical model. Autom Remote Control 2016. № 77. С. 1060–1070. DOI: https://doi.org/10.1134/S0005117916060096

49. Using LC's Sierra Systems, Lawrence Livermore National Laboratory, 2018 URL: https://hpc.llnl.gov/hardware/platforms/sierra

50. Mitropol’skii, Y.I. Electronic components and architecture of future supercomputers. Russ Microelectron. 2015. № 44. С. 139–153. DOI: https://doi.org/10.1134/S1063739715030063

UNICHECK

User name:
Кафедра кибербезпеки

Check ID:
1008355916

Check date:
24.06.2021 15:46:00 EEST

Check type:
Doc vs Internet

Report date:
24.06.2021 15:46:23 EEST

User ID:
100005590

File name: **Клейн Диплом**

Page count: **68** Word count: **15976** Character count: **119092** File size: **1.65 MB** File ID: **1008425556**

4.36% Matches

Highest match: **0.43%** with Internet source (**https://link.springer.com/article/10.1186/1687-3963-2012-5**)

4.36% Internet sources **207** ..... Page 70

No Library search was conducted

0% Quotes

Exclusion of quotes is off

Exclusion of references is off

0% Exclusions

No exclusions

Anti-Plagiarism v-15.257

Максимальное совпадение с одним документом 30.0%

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 13%**

ID: 95689 Название: Мультикомп'ютерна система загального призначення на основі топології «тор» Добавлено в БД: 2021-08-11 Авторы: Клейн О. Руководители: Медзатий Д.М. Консультанты: Опоненты:	Документ		Суммарное совпадение по Базе Данных	
	Символы	Лексемы	Символы	Лексемы
	111491	778	40908 (37%)	320 (41%)

Источник плагиата

ID	Описание	Наличие плагиата в документе	
		Символы	Лексемы
95356	Название: Мультикомп'ютерна система загального призначення на основі топології «тор» Добавлено в БД: 2021-06-24 Авторы: Клейн О.М. Руководители: Медзатий Д.М, Консультанты: Опоненты:	34002 (30.0%)	262 (34.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник Клейн Олександр Миколайович
Тема Мультикомп'ютерна система загального призначення на основі топології «тор»
Спеціальність 123 Комп'ютерна інженерія

Обсяг кваліфікаційної роботи:

кількість листів креслень 3; кількість сторінок записки 80

1. Короткий зміст КП та прийнятих рішень В рамках кваліфікаційної роботи було розроблено мультикомп'ютерну систему на основі топології «тор». За базу взято SMP-архітектуру, для покращення роботи системи було додано процесорний кеш та розроблено модуль когерентності кеш-пам'яті для надежного її функціонування.

2. Висновок про відповідність КП дипломному завданню Кваліфікаційна робота у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому, теоретичному, розділі кваліфікаційної роботи якісно та в повній мірі розглянуті методи вирішення поставленої задачі, був проаналізований кожен аспект, який стосується теми кваліфікаційної роботи. У наступному розділі було проведено порівняльну характеристику модулів, що можуть використовуватись в системі. Рішення про використовувані модулі обгрунтовано. Також були розглянуті переваги та недоліки топології «тор» У основній проектній частині роботи була реалізована сучасними методами та рішеннями мультикомп'ютерна система загального призначення на основі топології «тор». За проведеним у попередніх розділах аналізом, систему було покращено шляхом додавання кеш-пам'яті та створення модуля когерентності кеш-пам'яті. Було проведено тестування системи у двох варіантах – класичного вигляду, та покращеного. В загальному усі розділи відповідають завданню та містять сучасні методи вирішення поставлених завдань.

4. Позитивні сторони проекту Кваліфікаційна робота відповідає сучасним вимогам до проектування мультикомп'ютерних систем. Для проектування системи були вукористані сучасні програмно-апаратні рішення. Окремо можна виділити підняття питання про «вузьке місце» таких систем, та спроектовано і протестовано рішення для усунення такого недоліку.

5. Негативні сторони проекту Надмірна деталізація в плані питання вибору процесора. Добре було б детальніше розглянути питання масштабованості та організації системи. Вказаний недолік не зменшує позитивне враження від роботи.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконане відповідно до суті кваліфікаційної роботи. У першому листі відображено загальну схему мультикомп'ютерної системи загального призначення на основі, обраної, SMP-архітектури. У другому листі детально показано об'єднання процесорів та їх кешу один між одним та у топологію «тор». Третій лист представляє собою схему когерентності кеш-пам'яті. В загальному графічне оформлення виконане на належному рівні. Пояснювальна записка відповідає задекларованим нормам для її оформлення.

7. Відгук про роботу в цілому В загальному кваліфікаційна робота заслуговує схвальних відгуків. Весь матеріал роботи структурований, чіткий та послідовний. Усі розділи кваліфікаційної роботи йдуть у вірній послідовності, що дозволяє чітко розуміти викладений матеріал в рамках даної роботи. Графічний матеріал дозволяє наочно побачити принцип побудови, та методи покращення мультикомп'ютерної системи на основі топології «тор».

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Розглянувши позитивні та негативні сторони представленого дипломного проекту, можна зробити висновок, що він заслуговує оцінку «відмінно»

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи) доцент кафедри автоматизації, комп'ютерно-інтегрованих технологій та телекомунікацій, к.т.н. Федула Микола Васильович

«_____» _____ 2021 р. _____ (підпис)

ХНУ. Зам. 54, тир. 6000, 2007.

Завідувачу кафедри КІСП
д-р.техн.наук, проф. Говорущенко Т. О.

Клейн Олександр Миколайович
ПІБ здобувача вищої освіти

ФПКТС, 4 курсу, групи КІ-17-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 29.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіатоповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

23.06.2021
дата

підпис

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА СИСТЕМНОГО ПРОГРАМУВАННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Мультикомп'ютерна система загального призначення на основі топології «тор»

Автор: Клейн Олександр Миколайович

Спеціальність: 123 – Компютерна інженерія та програмування

Освітня програма: освітньо-професійна

Науковий керівник: Медзатий Д.М., к.т.н, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з більш ніж 10 джерелами на один фрагмент речення;
4) серед запозичень знаходяться загальновідомі терміни, скорочення та визначення.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає ____ і адресується до ____ першоджерела, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ Д. М. Медзатий

Гарант ОП _____ С.М.Лисенко

Завідувач кафедри КІСП _____ Т. О. Говорущенко