Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи візуалізації роботи багатоядерних платформ”

Виконав здобувач вищої освіти
IV курсу, групи КІ-19
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Федюк Я.В.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Коваленко А.С.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *123 “*Комп’ютерна інженерія*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Комп’ютерна інженерія*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Федюку Ярославу В’ячеславовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи візуалізації роботи багатоядерних платформ*

2. Керівник роботи *Коваленко Анна Степанівна, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 7-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи візуалізації роботи багатоядерних платформ*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Коваленко А.С.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Федюк Я.В.
(прізвище та ініціали)

АНОТАЦІЯ

Федюк Я.В. Програмне забезпечення системи візуалізації роботи багатоядерних платформ. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи візуалізації роботи багатоядерних платформ.

Метою розробки є програмне забезпечення системи візуалізації роботи багатоядерних платформ.

Результат роботи – програмна реалізація системи візуалізації роботи багатоядерних платформ.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Delphi 10.

Ключові слова: комп'ютерна інженерія, візуалізація, багатоядерні платформи

ABSTRACT

Fediuk Y.V. Software for visualization of the work of multi-core platforms. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this final qualification work for the first (bachelor) level of higher education, software is developed, which is intended for the system of visualization of the work of multi-core platforms.

The goal of the development is the software of the visualization system of multi-core platforms.

The result of the work is a software implementation of a system for visualizing the operation of multi-core platforms.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Delphi 10 environment.

Keywords: computer engineering, visualization, multicore platforms

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ОС	–	операційна система
ПЗ	–	Програмне забезпечення
АМР	–	асиметрична багатопроцесорна обробка
ВМР	–	виняткова багатопроцесорна обробка
CLI	–	інтерфейс командного рядка
DPM	–	цифровий панельний контрольно-вимірювальний пристрій
IPC	–	механізм міжзадачної взаємодії
HT	–	технологія Hyper-Threading
MTA	–	багатопотокова архітектура
HT	–	двонаправлена послідовно/паралельна комп'ютерна шина
OAM	–	експлуатація, адміністрування, підтримка
SMP	–	симетрична багатопроцесорна обробка
SMT	–	одночасне виконання потоків
SRQ	–	черга системних запитів

ВСТУП

Актуальність теми. Продуктивність двоядерного процесора може бути майже у два рази вище, ніж в одноядерного, а його вартість набагато нижче, ніж двох процесорів, що мають по одному ядру. При розміщенні двох процесорів на одному кристалі швидкість обміну інформацією між ними зростає, а спільне використання кеш-пам'яті може ще більше підвищити ефективність обробки даних. Крім того, двоядерні процесори займають менше місця, споживають менше енергії й розсіюють менше тепла, ніж окремі процесори.

Процесори на базі декількох ядер добре підходять для обробки транзакцій, а також для обслуговування баз даних і наукових застосувань.

Розвиток багатоядерних систем – це шлях до повсемісного використання паралельних обчислень. Як відомо, найпоширенішим способом підвищення продуктивності є саме розпаралелювання потоку команд або потоку даних. Розпаралелювання даних – це застосування однієї операції відразу до декількох елементів масиву даних. Паралелізм задач передбачає розбивку обчислювального процесу на декілька підзадач (процесів, потоків), кожна з яких виконується на своєму ядрі процесора. Багатоядерні системи відносять до класу MIMD (Multiple Instruction, Multiple Data). У них кілька програмних віток виконуються одночасно й незалежно, але в певні моменти вони обмінюються даними.

Зміна архітектури призводить до зміни алгоритмів програмування. Тож як студенти так і досвідчені програмісти тепер повинні вивчити багато нового.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи візуалізації роботи багатоядерних платформ.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем візуалізації роботи багатоядерних платформ.

– Дослідження системи візуалізації роботи багатоядерних платформ.

– Програмна реалізація системи візуалізації роботи багатоядерних платформ.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі візуалізації роботи багатоядерних платформ.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи візуалізації роботи багатоядерних платформ, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Багатоядерний процесор – центральний процесор, що містить два та більше обчислювальних ядра на одному процесорному кристалі або в одному корпусі (рисунок 1.1).

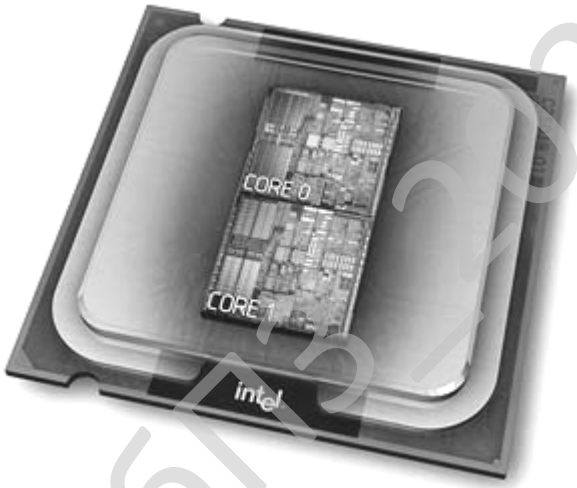

Рисунок 1.1 – Двоядерний процесор фірми Intel – Core 2 Duo

У всіх існуючих на сьогоднішній день багатоядерних процесорах кеш-пам'ять першого рівня в кожного ядра своя, а кеш 2-го рівня існує в декількох варіантах:

– поділювана – кеш розташований на одному кристалі з ядрами й доступний кожному з них у повному обсязі. Використовується в процесорах сімейств Intel Core.

– індивідуальна – окремі кеші рівного обсягу, інтегровані в кожне з ядер.

Обмін даними з кешем L2 між ядрами здійснюється через контролер пам'яті – інтегрований (Athlon 64 X2) або зовнішній (Pentium D).

Закон Амдала, що ілюструє обмеження росту продуктивності обчислювальної системи зі збільшенням кількості обчислювачів, говорить про те,

що приріст продуктивності (S) системи залежить від кількості процесорів (N) і частки послідовних операцій (c) у програмі:

$$S=1/(c+(1-c)/N) \quad (1.1)$$

Граничні значення с відповідають повністю паралельним (c=0) і повністю послідовним (c=1) програмам. Якщо лише 1/10 частина програми виконується послідовно, то в принципі неможливе прискорення в десять разів поза залежністю від числа використовуваних процесорів (ядер). Важливий наслідок закону Амдала полягає в тому, що максимальний ріст продуктивності (в N разів при N процесорах) недосяжний. У противному випадку частина програми, яка виконується послідовно, повинна бути рівною нулю, що неможливо. Ще один наслідок закону такий: чим менше частка частини, що виконується послідовно, програми, тим більше приріст продуктивності (рисунок 1.2).

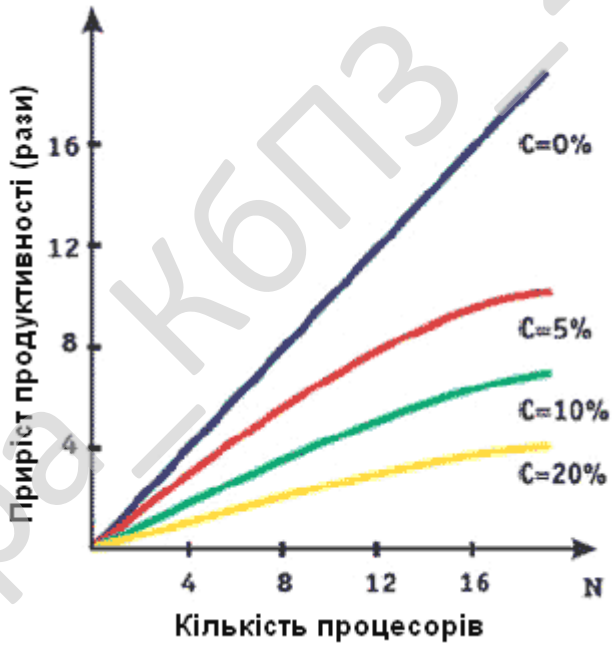

Рисунок 1.2 – Наслідки закону Амдала

Сьогодні тільки невелика частина програмного забезпечення може виконуватися на багатоядерних процесорах, що підтверджують результати тестів синтетичних і призначених для конкретних класів додатків. Реальний ріст продуктивності дають лише програми, оптимізовані під багатопоточність, такі як

Adobe Premiere Pro 1.5 і 3DMax. Тому дуже важлива розробка й впровадження драйверів пристроїв, що підтримують багатопоточність.

1.2 Область застосування

Активне впровадження багатоядерних систем передбачає істотну зміну стилю програмування: розроблювачі будуть змушені використовувати паралельні потоки, породження й обробку асинхронних подій і ін. Іншими словами, нова апаратна архітектура вимагає зміни програмної парадигми – переходу від послідовного стилю програмування до паралельного.

Певний вплив багатоядерні технології здатні зробити на системних розроблювачів і програмістів, що володіють великим досвідом в області однопроцесорних архитектур, але мають недостатньо спеціальних знань про конфігурації, що передбачають багатопроцесорну обробку.

Наявність декількох ядер може привести до збільшення складності розробки, пов'язаної з підключенням додаткових механізмів і внесенням змін у вихідний код, розроблений для одноядерних архітектур. Так, для взаємодії між собою додатки, виконувані в різних ядрах, можуть зажадати використання ефективних механізмів міжзадачної взаємодії (Interprocess Communication, IPC), інфраструктуру даних у спільно використовуваній пам'яті, а також базисні елементи синхронізації для захисту поділюваних ресурсів. Перенесення коду також являє собою проблему.

Тому дуже важливо створювати докладну документацію, емулятори та віртуальні стенди для ілюстрації роботи багатоядерних процесорів, щоб студенти та програміти могли вивчати принципи роботи та методи програмування багатоядерних платформ. Це завдання і було поставлене та вирішене у даній бакалаврській роботі.

Для того щоб продемонструвати, як саме масштабується продуктивність багатоядерного процесора залежно від оптимізації програмного коду до багатоядерної архітектури, розглянемо наступний приклад.

Є багатоядерний процесор з кількістю ядер рівною n. Припустимо, що на цьому процесорі виконується програма, що включає N інструкцій програмного коду, причому S інструкцій цього коду може виконуватися тільки послідовно один за одним, а P (рівне N – S) інструкцій є програмно незалежними одна від одної й можуть виконуватися одночасно на всіх ядрах процесора. Позначимо через s (рівне S/N) – частину інструкцій, виконуваних послідовно, а через p (рівне 1 – s) – частину інструкцій, виконуваних паралельно.

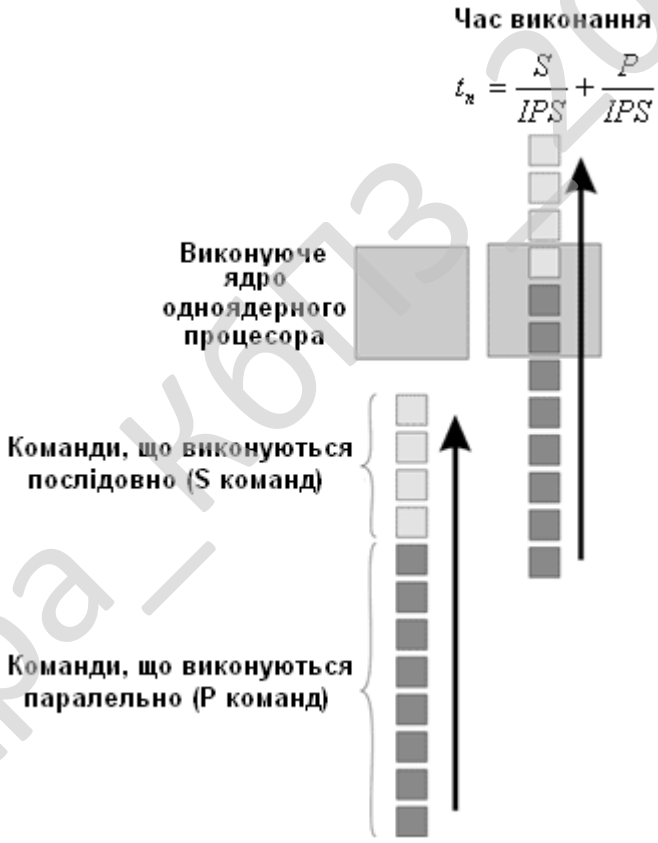

Рисунок 1.3 – Виконання програмного коду на одноядерному процесорі

У випадку застосування одноядерного процесора (рисунок 1.3) час, затрачуваний на виконання всього програмного коду, складе:

$$t_1 = N / IPS \qquad (1.2)$$

У випадку використання n-ядерного процесора (рисунок 1.4) час,

затрачений на виконання всього програмного коду, виявиться меншим за рахунок паралельного виконання P команд на n ядрах процесора й складе:

$$t_n = \frac{S}{IPS} + \frac{P}{n \cdot IPS} = t_1 \cdot \left(1 - p \cdot \left(1 - \frac{1}{n}\right)\right) \quad (1.3)$$

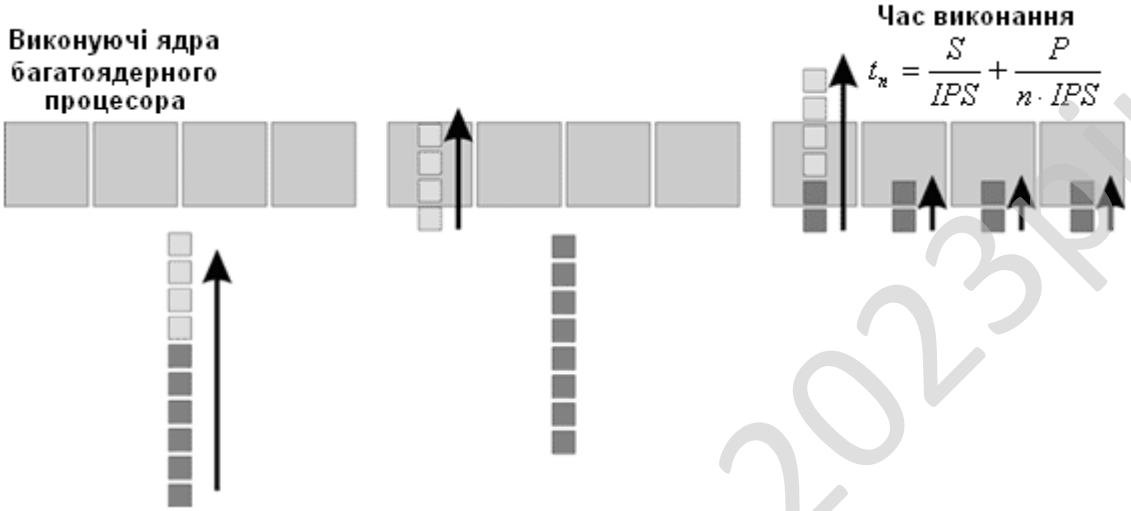

Рисунок 1.4 – Виконання програмного коду на багатоядерному процесорі

Оскільки приростом продуктивності в цьому випадку можна вважати скорочення часу виконання програми при використанні багатоядерного процесора в порівнянні з часом виконання тієї ж програми при застосуванні одноядерного процесора, тобто t_1/t_n, одержимо, що приріст продуктивності складе:

$$\frac{t_1}{t_n} = 1 - p \cdot \left(1 - \frac{1}{n}\right) \quad (1.4)$$

Багатоядерні процесори дозволяють відчути істотний приріст продуктивності при одночасній роботі декількох додатків. В ідеальному випадку кожний додаток може виконуватися на окремому ядрі процесора незалежно від інших додатків. Наприклад, можна паралельно конвертувати відеофайли, грати в ігри й т.п.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи візуалізації роботи багатоядерних платформ, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Операційна система, обрана для багатоядерної архітектури, може істотно вплинути на трудомісткість рішення проблем, пов'язаних із багатоядерними технологіями, однак багато чого буде залежати від того, чи підтримує ОС різні режими багатопроцесорної обробки, передбачені в багатоядерному мікропроцесорі. Дані режими характеризуються трьома основними реалізаціями:

– **Асиметрична багатопроцесорна обробка** (Asymmetric multiprocessing, AMP) – окрема ОС або індивідуальний образ однієї ОС працює на кожному ядрі процесора;

– **Симетрична багатопроцесорна обробка** (Symmetric multiprocessing, SMP) – єдиний образ ОС одночасно управляє всіма ядрами процесора, і додатки можуть використовувати будь-яке ядро;

– **Виняткова багатопроцесорна обробка** (Bound Multiprocessing, BMP) – єдиний образ ОС одночасно управляє всіма ядрами процесора, але кожний додаток закріплений за конкретним ядром.

Асиметрична багатопроцесорна обробка

Ассиметрична багатопроцесорність передбачає режим виконання, подібний використовуваному в традиційних однопроцесорних системах, що добре знає й розуміє більшість розроблювачів. Отже, цей режим пропонує відносно простий шлях перенесення існуючого коду. Він також реалізує прямий механізм для відстеження того, як використовуються ядра центрального процесора. Нарешті, у більшості випадків, режим дозволяє розроблювачам застосовувати стандартні інструменти й методи налагодження.

Асиметрична багатопроцесорна обробка може бути або однорідною – на кожному ядрі виконується операційна система одного типу й версії, або різнорідною – ядра працюють під керуванням різних ОС або різних версій однієї ОС. В однорідному середовищі виконання розроблювачі можуть використовувати кілька ядер шляхом вибору ОС, що підтримує розподілену програмну модель, наприклад QNX Neutrino. Установлена на цільову систему, дана модель дозволяє виконуваним в одному ядрі додаткам прозоро взаємодіяти з іншими додатками й системними службами (наприклад, із драйверами пристроїв і наборами протоколів) інших ядер, однак без значного завантаження центрального процесора, властивого традиційним формам міжзадачної взаємодії.

Різнорідне середовище виконання пред'являє трохи інші вимоги. У цьому випадку розроблювач повинен або реалізувати власну схему взаємодії, або вибрати дві ОС, які мають загальну інфраструктуру (імовірно, засновану на протоколі IP) для межпроцесорного взаємодії. Для того щоб уникнути конфліктів ресурсів, такі ОС також повинні передбачати стандартизовані механізми для доступу до поділюваних апаратних компонентів.

Приведемо приклад використання різнорідного середовища: одне ядро обробляє вхідний потік інформаційного обміну від апаратного інтерфейсу, а інше – відповідає за обробку вихідного трафіка. У силу того що трафік представляється двома незалежними потоками, двом ядрам немає необхідності взаємодіяти й обмінюватися даними між собою. У результаті операційна система не повинна забезпечувати механізм міжзадачної взаємодії між ядрами. Незважаючи на це, вона повинна гарантувати продуктивність у реальному часі, необхідну для того, щоб упоратися з потоками трафіка.

Існує й інший приклад використання однорідного середовища, у якому два ядра працюють із розподіленою панеллю керування, при цьому кожне ядро звертається до різних частин інформаційної панелі. Для правильного керування інформаційною панеллю додатки, виконувані на декількох ядрах, повинні функціонувати в погодженому режимі. Для реалізації цієї погодженості ОС

повинна забезпечувати надійну підтримку механізмів міжзадачної взаємодії, таких як інфраструктура спільної пам'яті для інформації таблиці маршрутизації.

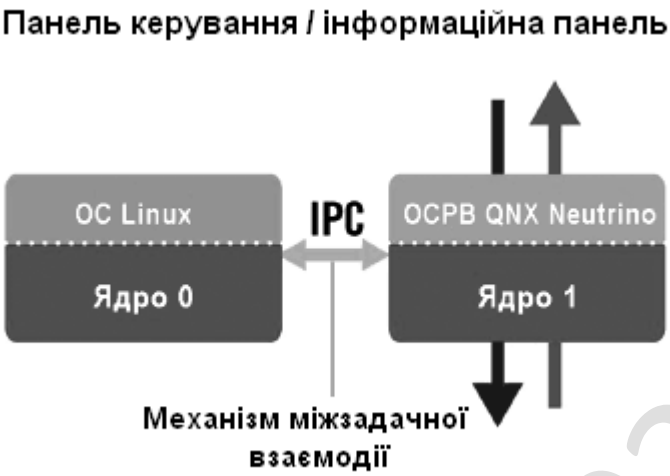

Рисунок 2.1 – Використання різнорідної асиметричної багатопроцесорної обробки (AMP) для режимів роботи панелі керування й інформаційної панелі

У наведеному прикладі використання різнорідного середовища виконання (рисунок 2.1) одне ядро реалізує панель керування, у той час як інше обробляє весь трафік, який надходить із інформаційної панелі, що означає забезпечення продуктивності в реальному часі. У цьому випадку обидві операційні системи, що виконуються на двох ядрах, повинні підтримувати послідовний механізм міжзадачної взаємодії, наприклад властивістю «прозорості», якою володіє протокол міжзадачної взаємодії (Transparent Inter-Process Communication Protocol). Протокол дозволяє ядрам обмінюватися даними шляхом можливого використання поділюваних структур і є стандартом при забезпеченні взаємодії між різнорідними операційними системами в мережному середовищі. У порівнянні з TCP/IP даний протокол має більш високу швидкість і ефективність, виключає втрату повідомлень і підтримує ще цілий ряд можливостей, що застосовуються у мережних пристроях наступного покоління.

Фактично у всіх випадках підтримка операційною системою основних і зручних для використання протоколів обміну даними може поліпшити міжядерну взаємодію. Зокрема, побудована з урахуванням парадигми розподіленого

програмування ОС може використовувати переваги паралелізму, що забезпечується декількома ядрами. Найбільш типовими ОС, що підтримують AMP, є системи жорсткого реального часу, такі як VxWorks, OSE і QNX, однак у даний момент і окремі реалізації Linux, адаптовані під реальний час, мають дану функціональність.

За допомогою асиметричної багатопроцесорності розроблювач додатків має можливість визначити, яким чином спільні апаратні ресурси, використовувані додатками, будуть розподілятися між ядрами. Як правило, це відбувається статично в процесі завантаження й включає розподіл фізичної пам'яті, використання периферійних пристроїв, а також керування перериваннями. Якщо система може розподіляти ресурси динамічно, то це спричинить складну координацію між ядрами.

Специфіка асиметричної багатопроцесорної обробки така, що процеси кожної ОС будуть завжди виконуватися на своєму ядрі, навіть якщо інші ядра перебувають у стані очікування. У результаті одне ядро може використовуватися недостатньо або зі значним перевищенням рівня робочої потужності. Для того щоб вирішити проблему таких станів, система може дозволити додаткам динамічно переміщатися від одного ядра до іншого. Такий шлях, однак, може викликати введення складних контрольних точок для перевірки інформації про стан або можливі переривання обслуговування в тих випадках, коли виконання додатка зупиняється на одному ядрі й відновляється на іншому. Також перенесення стає досить складним, а іноді й взагалі неможливим у випадках, коли на ядрах виконуються різні ОС.

Симетрична багатопроцесорна обробка

Розподіл ресурсів у багатоядерній архітектурі може бути важким, особливо коли програмні компоненти не мають інформації про те, яким чином інші компоненти використовують ці ресурси. Симетрична багатопроцесорність пропонує виконання тільки однієї ОС у всіх ядрах системи. Тому що ОС має можливість впливати на роботу всіх системних елементів у будь-який момент

часу, вона може розподіляти ресурси між декількома ядрами з мінімальним втручанням розроблювача додатків або зовсім без його участі. Більш того, ОС може підтримувати вбудовані стандартизовані операції, які дозволяють декільком додаткам безпечно й просто розділяти ресурси. Симетрична багатопроцесорна обробка застосовується звичайно в настільній і серверній операційній системах.

Запускаючи тільки одну ОС, можна динамічно розподіляти ресурси між спеціальними додатками, але не між ядрами центрального процесора, забезпечуючи тим самим більш ефективне використання доступного устаткування. Також це дозволяє засобам системного трасування збирати статистичну інформацію про виконання й взаємодію додатків на рівні багатоядерної системи в цілому, сприяючи правильному уявленню розроблювачів про оптимізацію й налагодження додатків. Наприклад, системний профайлер комплекту розроблювача QNX Momentics може відслідковувати переходи потоків керування від одного ядра до іншому, а також використання базових функцій ОС, подій планування, обміну повідомленнями між додатками й інші події за допомогою часових оцінок з високою дозволяючою здатністю. Синхронізація додатків також стає значно більш простою, тому що розроблювачі можуть використовувати базові функції ОС замість складних механізмів міжзадачної взаємодії.

Належним чином реалізована ОС із підтримкою симетричної багатопроцесорності забезпечує дані переваги, не змушуючи розроблювача використовувати спеціалізовані програмні інтерфейси додатків або мови програмування. Уже давно розроблювачі використовували програмний інтерфейс додатка pthreads зі стандарту POSIX у системах із симетричною багатопроцесорністю високого класу. Стандарт POSIX дозволяє розроблювачам писати код, що може бути використаний як в однопроцесорних, так і багатоядерних системах. У дійсності, деякі ОС (такі, як Linux, Solaris, QNX) дозволяють застосовувати той самий вихідний код при виконанні на кожному з типів процесорів.

Добре спроектовані ОС із симетричною багатопроцесорністю, такі як Windows, Solaris, QNX Neutrino, дозволяють потокам керування в межах додатка виконуватися одночасно на будь-якому ядрі. Такий паралелізм у будь-який момент часу надає додаткам повну обчислювальну потужність процесора. Якщо ОС дає відповідні можливості по пріоритетному перериванню обслуговування й призначенню пріоритетів потокам керування, то це може допомогти розроблювачеві додатків забезпечити виділення процесорних циклів тим додаткам, яким це найбільш необхідно.

У сценарії панелі керування, представленому на рисунку 2.2, симетрична багатопроцесорність дозволяє всім потокам керування, що належать різним процесам, виконуватися на будь-якому ядрі. Наприклад, процес інтерфейсу командного рядка (Command-Line Interface, CLI) може виконуватися в той час, як додаток трасування здійснює інтенсивні обчислення.

Рисунок 2.2 Використання симетричної багатопроцесорної обробки в панелі керування

Виняткова багатопроцесорна обробка

Виняткова багатопроцесорність – новий режим, запропонований компанією QNX Software Systems, що зберігає переваги прозорого адміністратора ресурсів симетричної багатопроцесорної обробки й дає розроблювачам

можливість прив'язувати програмні задачі до конкретних ядер. Як і у випадку симетричної багатопроцесорності, одна копія ОС зберігає повне представлення про всі системні ресурси, дозволяючи динамічно розподіляти й розділяти їх між додатками. Однак під час ініціалізації додатків обумовлена системним розроблювачем установка може змусити всі потоки керування додатка виконуватися тільки на заданому ядрі.

У порівнянні з повноцінним «плаваючим» режимом симетричної багатопроцесорності, такий підхід забезпечує кілька переваг. Він усуває переповнення кеш-пам'яті, що може погіршити продуктивність систем із симетричною багатопроцесорністю, дозволяючи додаткам, що розділяють той самий набір даних, виконуватися винятково на одному ядрі. Також він пропонує більш швидке в порівнянні із симетричною багатопроцесорністю налагодження додатків, оскільки всі виконувані в рамках одного додатка потоки керування запускаються на одному ядрі. Нарешті, він допомагає успадкованим додаткам, розробленим для однопроцесорних систем, коректно виконуватися, дозволяючи запускати їх на одному ядрі.

При використанні виняткової багатопроцесорності додаток, закріплений за одним ядром, не може задіяти інші ядра, навіть якщо вони простоюють. Розроблювач ОС може запропонувати інструменти для аналізу ступеня використання ресурсів (включаючи завантаження процесора) для кожного з додатків і рекомендації із кращого способу розподілу додатків по ядрах з метою досягнення максимальної продуктивності. Якщо ОС також дозволяє динамічно змінювати призначене процесорне ядро за допомогою додаткового мікропроцесора, то користувач одержує можливість динамічно перемикатися з одного ядра на інше без додаткового введення контрольних точок і зупинки/перезапуску додатка.

У прикладі, показаному на рисунку 2.3, система з винятковою багатопроцесорністю працює в напівдуплексному режимі. Приймаючий процес із декількома потоками керування працює на ядрі 0, а передаючий процес, що також

володіє декількома потоками, виконується на ядрі 1. Даний підхід рятує розроблювачів від трудомісткої задачі по збору інформації окремо по кожному ядру і її об'єднання для наступного аналізу.

Рисунок 2.3 – Використання виняткової багатопроцесорності в напівдуплексному режимі

У прикладі, показаному на рисунку 2.4, додатки панелі керування (інтерфейс командного рядка; експлуатація, адміністрування й технічне обслуговування; керування інформаційною панеллю) виконуються на ядрі 0, у той час як вхідні й вихідні додатки інформаційної панелі працюють спільно на ядрі 1. Розроблювачі можуть реалізувати механізм міжзадачної взаємодії (IPC) для цього сценарію, використовуючи локальні механізми ОС або синхронізовані захищені структури поділюваної пам'яті.

Рисунок 2.4 – Використання виняткової багатопроцесорної обробки (BMP) для режимів роботи панелі керування й інформаційної панелі

Займаючи проміжне місце між асиметричною й симетричною багатопроцесорністю, виняткова багатопроцесорна обробка пропонує ефективну стратегію переносу користувачам, що планують перейти на повну симетричну багатопроцесорність, і яких, однак, турбує працездатність існуючого програмного коду в діючій паралельній моделі виконання. Користувачі можуть переносити наявний код у багатоядерний процесор і з самого початку об'єднувати його для одного ядра, щоб забезпечити правильне функціонування.

Зв'язуючи додатки (і, можливо, окремі потоки) з конкретними ядрами, розроблювачі також можуть виявити потенційні проблеми паралелізму на рівні додатків і потоків. Рішення даних проблем дозволить додаткам виконуватися повністю одночасно, максимізуючи тим самим виграш у продуктивності, який надається багатоядерними процесорами.

Чи повинен розроблювач вибрати асиметричну, симетричну або виняткову багатопроцесорність?

Асиметрична багатопроцесорність добре працює з існуючими додатками, але має обмежену масштабованість за межами двох ядер. Симетрична багатопроцесорність пропонує прозоре керування ресурсами, але може не працювати із програмним забезпеченням, спроектованим для однопроцесорних систем. Виняткова багатопроцесорність пропонує ті ж переваги, що й симетрична, однак дозволяє додаткам, розрахованим на один процесор, працювати коректно, значно спрощуючи перенос існуючого програмного забезпечення. Гнучкість у виборі кожної із цих моделей дозволяє розроблювачам досягти оптимального балансу між продуктивністю, масштабованістю й простотою переносу.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

– Велика кількість поліпшень STL від Dinkumware.

– Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.

– Поліпшена підтримка Cmake.

– Велика кількість виправлень для підвищення стабільності і якості.

– Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.

– Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи візуалізації роботи багатоядерних платформ.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Опис будови багатоядерних процесорів

Розглянемо будову багатоядерних процесорів на прикладі Athlon 64 X2 та Intel Core 2 Duo. Ці дві найбільші фірми по виробництву процесорів представляють два різних підходи до будови багатоядерних платформ. Структурні схеми їх двоядерних процесорів показані на рисунку 3.1.

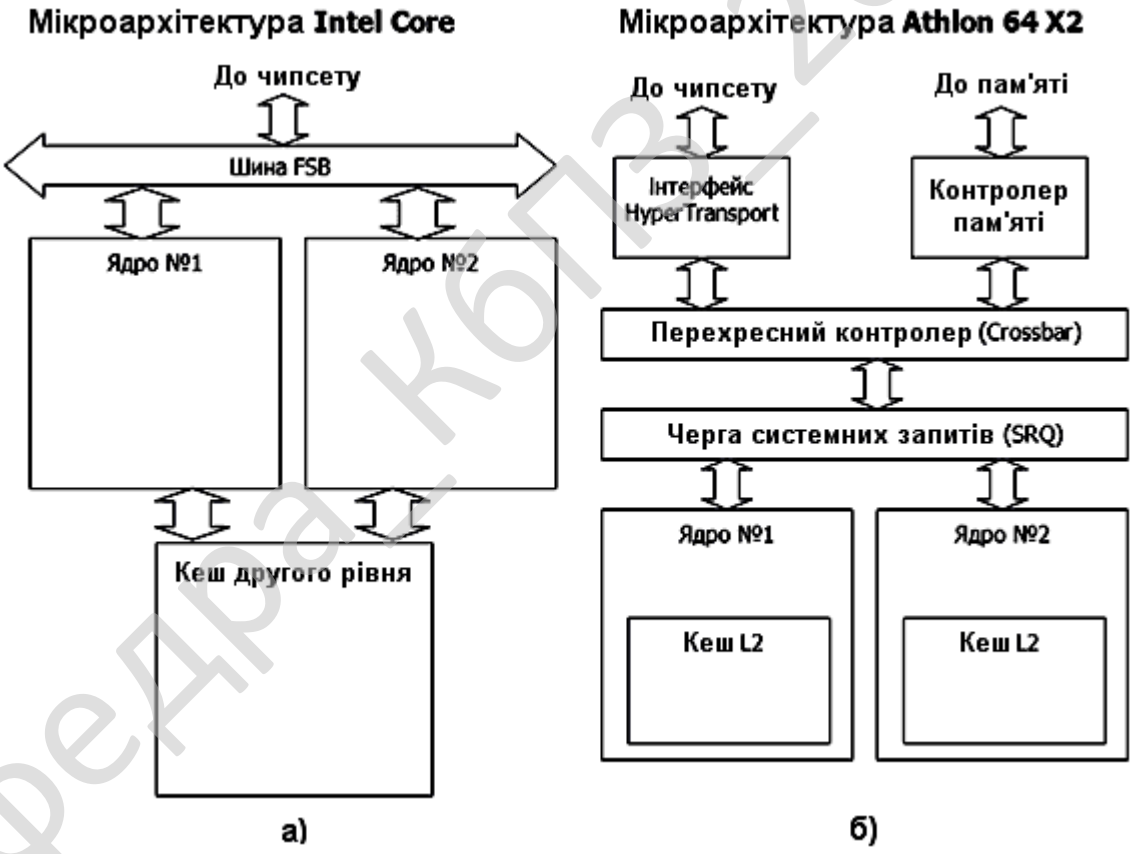

Рисунок 3.1 – Структурна схема двоядерного процесора Intel Core 2 Duo (а) та Athlon 64 X2 (б)

Як видно з рисунку процесори фірми AMD використовують індивідуальну кеш-пам'ять, а процесори фірми Intel – поділюваний кеш.

При створенні багатоядерних процесорів для настільних ПК мікропроцесорний гігант Intel пішов по шляху “найменшого опору”, продовживши традиції створення звичних для себе SMP-систем із спільною шиною. Виглядає подібна SMP-система надзвичайно просто: один чипсет, до якого підключається вся оперативна пам'ять, і одна процесорна шина, до якої підключені всі процесори. Ніякої загальної схемотехники в цих ядер немає.

Pentium являються найпростішими двоядерними процесорами. Централізований підхід Intel, по-перше, відрізняється відносною простотою, а по-друге, зручний тим, що в ньому кожний компонент комп'ютера виходить вузькоспеціалізованим, і піддається модернізації незалежно від інших компонентів. Можна, наприклад, використовувати з тим самим процесором зовсім різні типи оперативної пам'яті.

Архітектура AMD не просто відрізняється від Intel, вона концептуально інша, оскільки в ній немає якогось виділеного центра. Кожний із процесорів архітектури AMD64 є незалежною й “самодостатньою” одиницею. Із чисто технічної сторони AMD попросту інтегрувала практично всю функціональність північного мосту в центральний процесор.

На відміну від Intel в AMD використовується не SMP, а NUMA-система. Невелике технологічне відхилення приводить до зовсім іншої архітектури комп'ютера.

NUMA-системи (Non-Uniform Memory Access systems) – системи з "неоднорідним" доступом до пам'яті. Час доступу до пам'яті визначається її положенням відносно центрального процесора. Одна частина пам'яті "швидша", інша – "повільніша", у системі при цьому утворяться своєрідні "острівці" зі своєю, швидкою "локальною" оперативною пам'яттю, з'єднані відносно повільними лініями зв'язку. Звертання до "своєї" пам'яті відбуваються швидко, до "чужої" – повільніше, причому чим "далі" чужа пам'ять розташована, тим повільніше

виходить доступ до неї. Створювати NUMA-системи куди простіше, ніж SMP, а от програми писати складніше – без врахування неоднорідності пам'яті ефективну програму для NUMA уже не написати. Свою реалізацію NUMA-системи AMD називає SUMA.

Основа SUMA – шина HyperTransport. У кожний процесор інтегрується контролер "локальної" оперативної пам'яті (власне, у порівнянні з одноядерними процесорами AMD64 контролер пам'яті майже не змінився). Звертання до пам'яті "чужих" процесорів відбуваються по шині HyperTransport, причому робиться ця "переадресація" запитів абсолютно прозоро для обчислювального ядра процесора – її здійснює вбудований в Northbridge комутатор (CrossBar), що працює на повній частоті процесора. Цей же самий CrossBar забезпечує "автоматичну" маршрутизацію проходячих через процесор повідомлень від периферійних пристроїв і інших процесорів, включаючи обслуговування "чужих" запитів до оперативної пам'яті HT спеціально оптимізувалася для подібного режиму роботи. Шина дозволяє одночасно передавати дані на швидкості в “обидві сторони” до 8Гбайт/с сумарно. Модель пам'яті виходить неоднорідною (NUMA), але розходження у швидкості "своїх" і "чужих" ділянок оперативної пам'яті виходять відносно невеликими.

Шина HyperTransport – це двонаправлена послідовно/паралельна комп'ютерна шина, з високою пропускною здатністю й малими затримками. Історично HyperTransport розроблявся AMD як процесорна шина нового покоління спеціально для чипів з інтегрованим контролером пам'яті (архітектура AMD64). У багатопроцесорних системах на основі AMD Opteron підсистема пам'яті "розділена" між всіма процесорами – у кожного є своя локальна пам'ять, підключена через інтегрований контролер, і кожний може звертатися до пам'яті будь-якого іншого процесора. Локальна пам'ять "швидка", а пам'ять сусіда – "повільна", причому чим "далі" розташований сусід, тим повільніше пам'ять. Повільність є наслідком того, що для звертання до сусіда потрібно проробити цілий ряд операцій – переслати по міжпроцесорній шині запит, дочекатися його

виконання контролером пам'яті адресата, повернути дані по шині назад. Очевидно, що чим швидше при цьому шина, тим більш "однорідна" пам'ять.

Виходячи із цих міркувань і проектувалася нова шина, покликана забезпечити пропускну здатність не меншу, ніж в оперативної пам'яті, і мінімальні затримки на передачу даних і повідомлень. Вийшло дійсно непогано, що навіть дало AMD привід назвати свою архітектуру не NUMA, а SUMA – Slightly Uniform Memory Architecture, тобто "майже однорідна" архітектура пам'яті.

Але розроблювачі не стали робити типову швидку вузькоспеціалізовану процесорну (системну) шину. Навпроти, відповідно до віянь часу вони спорудили дуже швидку послідовну шину даних і передбачили можливості для її "переносу" у більш прості й повільні варіанти зі зменшеними частотами й розрядністю.

Тому HT (мінімальна ширина якої – 2 біти) з повним правом може називатися послідовною шиною – будь-які дані, передані по ній, упаковуються в пакети стандартного виду. Правда, вимоги швидкості наклали на протокол передачі даних сильні обмеження – настільки витончену "layered architecture", як в Intel, ми тут не побачимо, та й вплив фізичної реалізації лінків HT на загальну архітектуру шини дуже помітний.

HyperTransport підтримує автоматичне визначення ширини шини, від 2-х бітних ліній до 32-х бітних ліній. Повнорозмірна, повношвидкісна, 32-х бітна шина у двоспрямованому режимі здатна забезпечити пропускну здатність до 20800 Мб/с (2*(32/8)*2600), будучи, таким чином, найшвидшою шиною серед собі подібних.

“Базова” тактова частота шини HT – 200 МГц (тобто частота передачі даних – 400 МГц). Це мінімально можлива частота функціонування шини.

Шина HyperTransport підтримує технології енергозбереження, а саме ACPI. Це значить, що при зміні стану процесора (C-state) на енергозберігаюче, змінюється також і стан пристроїв (D-state). Наприклад, при відключенні процесора жорсткий диск також відключаються.

Відмінність HT від інших послідовних пакетних протоколів у тім, що при передачі одного пакета, можливо перервати передачу, і передати другий – більш терміновий. Така екзотична на перший погляд схема потрібна для забезпечення екстремально низьких затримок передачі повідомлень: при досить широкій шині типова затримка передачі повідомлення в HT становить 1-2 такти.

Чипсет сильно спрощується: усе, що від нього потрібно – це просто забезпечувати "мости" (тунелі) між HT і іншими типами шин. Ну й, можливо, заодно забезпечувати якусь кількість інтегрованих контролерів. Тут усе сумісно з усім: будь-які сучасні чипсети для мікроархітектури AMD64, теоретично, повинні працювати й з будь-якими процесорами AMD. І, зокрема, всі двоядерні процесори AMD повинні працювати з усіма раніше випущеними чипсетами для процесорів архітектури K8.

Інтеграція Northbridge у процесор і SUMA-архітектура не просто забезпечує “більш швидкий контролер оперативної пам'яті”, – вона заодно дозволяє дуже ефективно вирішувати й ряд властивих багатопроцесорним системам проблем. Наприклад, SUMA вирішує “проблему спільної пам'яті”.

Правда, латентність (час відгуку) оперативної пам'яті по мірках процесора залишається як і раніше величезним, але й із цією напастю навчилися ефективно боротися, використовуючи кеш-пам'ять значних розмірів і механізми апаратної й програмної предвибірки з пам'яті. Однак варто поставити в систему не одне, а два, чотири, а то й вісім процесорних ядер, як проблема "повільної пам'яті" вспливає з колишньою силою – особливо в архітектурі SMP (процесорів Intel) з контролером пам'яті в чипсепті.

В ідеалі будь-яка багатопроцессорная система повинна бути добре збалансована – занадто швидка пам'ять обходиться занадто дорого; занадто повільна – зводить ефект від установки декількох процесорів до мінімуму.

Програмне забезпечення багатоядерних систем

При переході з одноядерних процесорів на багатоядерні доводиться брати до уваги проблему послідовного виконання. Що це означає відносно до

багатоядерної системи? У ній виконання вважається послідовним, якщо в якийсь момент одне або більше ядер не можуть виконувати код одночасно з іншими ядрами. Така ситуація може виникнути по різним причинам: блокування при доступі до ресурсів, необхідність синхронізації процесів або потоків на ядрах, підтримка когерентності кеш-пам'яті, нерівномірність завантаження.

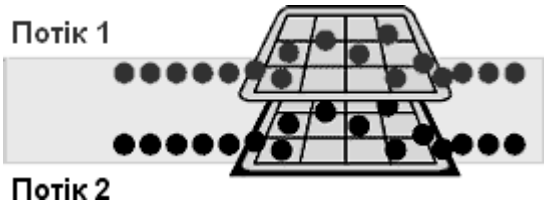

Рисунок 3.2 – Обробка потоків у двоядерному процесорі

Блокування виникають через неможливість (наприклад, у момент "збору сміття") одночасного доступу додатків на різних ядрах до жорсткого диску, до пристроїв введення/виведення або даним додатків. Дуже часто паралельні процеси, що виконуються на різних ядрах (рисунок 3.2), потрібно синхронізувати в певні моменти. Наприклад, додаток на одному з ядер повинен використовувати проміжні дані, які одержує додаток (потік, процес) на іншому ядрі. Перший додаток не може продовжити роботу до одержання цих даних, тобто перебуває в стані очікування. У такій ситуації неминучі накладні витрати на синхронізацію додатків (процесів, потоків), виконуваних на різних ядрах. У свою чергу, це обумовлює зниження ефективності паралельної роботи, що знаходить висвітлення в мережному законі Амдала. Виникає необхідність у підтримці когерентності (погодженості) кеш-пам'яті для всіх ядер при використанні поділюваної пам'яті.

Інструментальні засоби багатоядерних систем

Компілятори. Щоб одержати максимальну вигоду від використання багатоядерної архітектури потрібна підтримка на рівні компілятора. Так, в 2005 році Intel випустила версію 9.0 компілятора мов C++ і Фортран для платформ Linux і Windows. Цей компілятор дозволяє ефективно використовувати

можливості технології Hyper-Threading і багатоядерних процесорів. Він підтримує можливість автопаралелізму, тобто автоматичного виявлення в додатках можливості створення безлічі паралельних потоків з підтримкою специфікації OpenMP 2.5. Завдяки підтримці стандарту OpenMP компілятор Microsoft Visual C++ 2005 забезпечує паралельну багатопоточну обробку. Для цього потрібно або вказати параметр компілятора "/openmp", або встановити в конфігурації прапор "OpenMP Support". З листопада 2005 року компілятор gcc для мов C, C++ та Фортран 95 підтримує OpenMP за допомогою опції "-fopenmp". Варто згадати й набір компіляторів EKOPath компанії PathScale, призначених для 64-розрядних систем на базі Linux (AMD64 і EM64T).

Програмні відлагоджувачі. Програмісти добре знають, як важко налагоджувати багатопоточні додатки. При аварійному завершенні програми найчастіше потрібно проаналізувати стек викликів функцій у всіх потоках, але звичайний відлагоджувач показує тільки стек потоку, на якому відбулося аварійне завершення програми. Наприклад, стандартні засоби gdb погано пристосовані для налагодження багатопоточних додатків, тому пропонуються спеціальні версії цього відлагоджувача для конкретних операційних систем, у їхні ядра включаються додаткові можливості налагодження багатопоточних додатків. Одна з таких реалізацій -відлагоджувач компанії Etnus TotalView, призначений для платформ Linux, Unix і LynxOS. Він підтримує багатопоточність, MPI, OpenMP, мови програмування C/C++ і Фортран, а також змішані коди з використанням різних мов програмування. Корисним засобом оптимізації й налагодження паралельних програм є пакет Intel Threading Tools. Він забезпечує діагностику помилок і аналіз продуктивності багатопоточних додатків, що використовують моделі потоків Win32 і OpenMP. Відлагоджувач дозволяє виявляти взаємні блокування (deadlock) і перегони (race condition) між потоками, локалізувати проблеми на рівні вихідного коду, аналізувати ефективність способів підвищення продуктивності OpenMP-програм.

Апаратні відлагоджувачі. Для роботи з віртуальними машинами апаратний відлагоджувач повинен підтримувати ряд спеціальних функцій (зокрема, визначати, до якої віртуальної машини ставляться ті або інші процеси й потоки). Їх забезпечує, наприклад, TRACE32 компанії Lauterbach. Завдяки повній підтримці вбудованих апаратних блоків керування пам'яттю можна одночасно налагоджувати процеси на декількох віртуальних машинах і навіть два варіанти одного процесу на різних віртуальних машинах. Зокрема, Lauterbach оголосила про випуск програмного інструментарію інтегрованої підтримки ядра (kernel awareness) для операційної системи LynxOS-178. Щоб одержати доступ до всіх функцій TRACE32, не потрібно змінювати прикладні програми або ядро (застосовувати латки, перехоплювачі, інструментальні доповнення та ін.). Налагоджує саме той додаток, що буде діяти в кінцевому продукті, що дуже важливо для його сертифікації. Серед інших апаратних відлагоджувачів, що підтримують роботу із багатоядерними конфігураціями, назвемо Green Hills Probe і SuperTrace компанії Green Hills, WindPower ICE компанії Wind River, RealView ICE від ARM.

Стандарти багатоядерних систем

При розробці паралельних програм використовуються спеціалізовані бібліотеки й системи паралельного програмування PVM, LAM, CHMP та ін. Три основних підходи до реалізації цих систем розрізняються методами взаємодії паралельних завдань. Перший підхід базується на концепції обміну повідомленнями, другий – на використанні поділюваної пам'яті, третій опирається на стандарт POSIX і поєднує ці два підходи.

Найбільш відомим представником першої групи є специфікація MPI (Message Passing Interface) для мов С й Фортран, перший варіант якої з'явився в 1994 році. MPI забезпечує приблизно 200 функцій, охоплює безліч компіляторів і операційних систем. Серед найпоширеніших її реалізацій бібліотека MPICH. Крім того, пропонуються кілька комерційних реалізацій MPI, наприклад MPI/Pro компанії Verari Systems Software. MPI/Pro оптимізує час роботи паралельних

додатків і підтримує їхню масштабованість за рахунок балансування параметрів продуктивності й використання ресурсів. Verari пропонує версії MPI/Pro для різних операційних систем, у тому числі Windows, Linux, Mac OS X, LynxOS, і таких комунікаційних середовищ, як Gigabit Ethernet, Myrinet і InfiniBand.

До другої групи належить специфікація OpenMP (Open specifications for Multi-Processing). Її перша версія, що була випущена в 1997 році, призначалася для мови Фортран. До появи OpenMP причетні компанії IBM, Intel, Sun Microsystems і Hewlett-Packard. В 1998 році були створені варіанти OpenMP для мов C/C++, а останньої є версія 2.5. Підтримка специфікації OpenMP забезпечена у всіх компіляторах Intel починаючи із шостої версії, в Microsoft C/C++ починаючи з Visual Studio 2005, а також в GCC.

OpenMP – це набір спеціальних директив компіляторові (pragma), бібліотечних функцій і змінних середовищ. Найбільш оригінальні директиви компіляторові, які використовуються для позначення областей у коді й можуть виконуватися паралельно. Компілятор, що підтримує OpenMP, перетворить вихідний код і вставляє відповідні виклики функцій для паралельного виконання цих областей коду.

У третю групу входить специфікація POSIX (Portable Operating System interface for unIX).Основна специфікація розроблена як IEEE 1003.1 і схвалена як міжнародний стандарт ISO/IEC 9945-1:1990. З погляду організації паралельних обчислень найбільший інтерес представляють три частини стандарту 1003.1a (OS Definition), 1003.1b (Realtime Extensions) і 1003.1c (Threads). У рамках POSIX можна реалізувати паралельні обчислення на основі обміну повідомленнями (аналогічно MPI) або поділюваної пам'яті (як в OpenMP). Природно, в POSIX припустима й будь-яка комбінація цих методів. Найбільшою мірою стандарту POSIX відповідають (і відповідним чином сертифіковані) операційні системи реального часу LynxOS і Integrity.

Підтримка на рівні ОС

Багатоядерні процесори вимагають від операційних систем підтримки різних архитектур багатопроцесорної обробки. Компанія QNX Software Systems оголосила про випуск комплекту розроблювача QNX Momentics Multi-Core Edition. Цей набір інструментів призначений для створення програмного забезпечення і його міграції на багатоядерні апаратні рішення нового покоління, у тому числі процесори BCM12xx і BCM14xx компанії Broadcom, процесор MPC8641D компанії Freescale і багатоядерні процесори Intel. Будуть підтримуватися кілька моделей багатопроцесорності для багатоядерних архитектур: асиметрична AMP (забезпечення повного керування й відмовостійкості); симетрична SMP (максимальні паралелізм і масштабованість); "виняткова" BMP (підтримка міграції коду й зниження складності розробки).

Підтримку багатоядерних систем на базі процесорів AMD64, Sun UltraSPARC T1 і Intel забезпечує ОС Solaris 10. Наприклад, вбудована система віртуалізації й захисту інформації Solaris Containers дозволяє системному адміністраторові організовувати в рамках єдиної операційної системи кілька віртуальних системних розділів "зон". Кожній зоні можна призначити свій контейнер – набір локалізованих системних ресурсів. Контейнери можуть бути основою для керування ресурсами на рівні ядер. Реалізовані в Solaris 10 функції так званого "прогнозованого самовідновлення" (Predictive Self-Healing) забезпечують автоматичне визначення збоїв у роботі ядер і їхній перехід у пасивний режим без впливу на роботу інших ядер процесора. Підтримка багатоядерних систем реалізована в деяких дистрибутивах ОС Linux, наприклад Red Hat Enterprise Linux 4.

Багаторівнева віртуалізація

Поява багатоядерних процесорів дасть потужний додатковий поштовх масовому впровадженню технологій віртуалізації. Назвемо деякі з відомих підходів.

ARINC-653 (Avionics Application Software Standard Interface). Стандартний інтерфейс, розроблений компанією ARINC в 1997 році, вводить концепцію ізольованих розділів на основі універсального програмного інтерфейсу APEX (Application/Executive) між операційною системою й прикладним програмним забезпеченням. Вимоги інтерфейсу визначені так, щоб дозволити додаткам контролювати диспетчеризацію, зв'язок і стан внутрішніх оброблюваних елементів.

В 2003 році прийнята нова редакція ARINC-653, у якій введена концепція ізольованих віртуальних машин (розділів), рисунок 3.2. Її особливістю є жорстке й заздалегідь визначене квантування часу між віртуальними машинами, а метою – забезпечення гарантій того, що не виникне загальна відмова система. Стандарт ARINC-653 реалізований для операційних систем реального часу LynxOS-178, VxWorks, Integrity, CsLeos і ін.

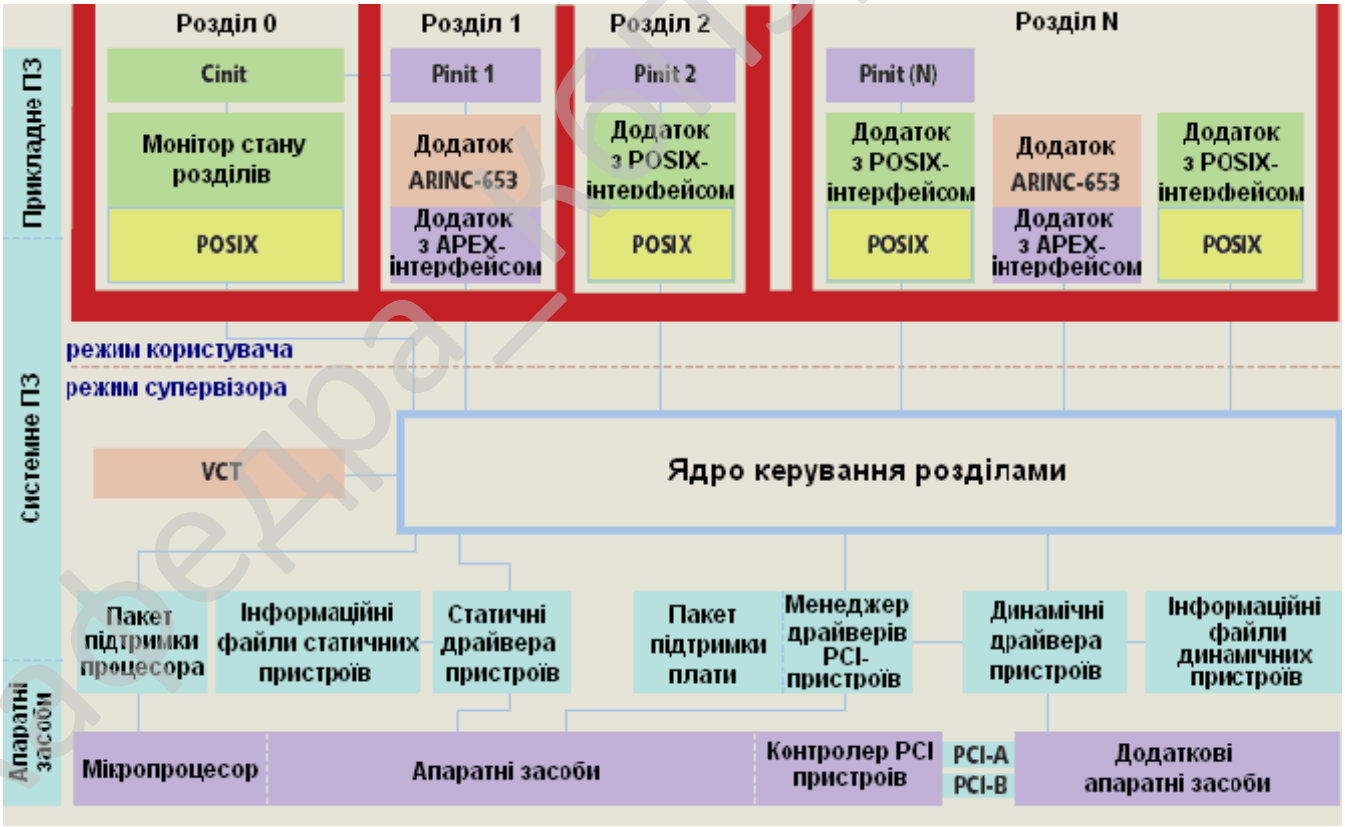

Рисунок 3.3 – Реалізація ARINC-653 в LynxOS-178

User-Mode Linux. UML ОС Linux у користувальницькому режимі -самий універсальний емулятор, що дозволяє створювати віртуальне устаткування, якого може й не бути на фізичному комп'ютері. Це досить зручно для тестування конфігурацій апаратного забезпечення. UML складається з набору заплат до ядра Linux, які дозволяють запускати інші операційні системи в консольних вікнах, і кожний користувач може незалежно завантажувати скільки завгодно операційних і віконних систем, аж до X11. User-Mode Linux допускається застосовувати для пристроїв з архітектурою IA-32 і PowerPC G5.

Програмні середовища віртуальних машин. Найбільш популярними з них є Microsoft Virtual PC і група програмних продуктів VMware. Система віртуальних машин дозволяє запускати на комп'ютері відразу кілька різних операційних систем і перемикатися з однієї на іншу без перезапуску комп'ютера. На комп'ютері, що працює під керуванням основної (базової) операційної системи, створюються один або кілька віртуальних комп'ютерів, на кожному з яких можна запустити "гостьову" ОС.

VMWare Workstation дозволяє запустити кілька екземплярів Windows, Linux і NetWare. Реалізовано повноцінну підтримку мережі, переносимість оточень і гнучкий підхід до роботи з оточенням. Проект Virtual PC споконвічно розробляла компанія Connectix, але на початку 2003 року його купила корпорація Microsoft.

Технологія віртуалізації Intel. VT, компонент багатоядерної технології підтримки віртуалізації на апаратному рівні, забезпечує підтримку віртуальних машин на рівні процесора за допомогою нового режиму VMX (Virtual Machine Extensions) і десяти команд vmptrld, vmptrst, vmclear, vmread, vmwrite, vmcall, vmlauch, vmresume, vmxoff і vmxon. При цьому підвищуються як надійність і продуктивність роботи додатків, так і рівень загальної безпеки.

Архітектура VT підтримує два класи ПЗ: монітор віртуальної машини VMM і "гостьове" програмне забезпечення. Використовуються два режими роботи root operation і non-root operation. Як правило, VMM працює в першому

режимі, а "гостьові" програми в другому. Підтримку технології віртуалізації Intel мають намір організувати такі виробники операційних систем, як RedHat, SuSe і MontaVista. Вона буде забезпечена й в інших програмних засобах віртуалізації, наприклад в VMware.

Паралельне програмування багатоядерних процесорів

Розробка алгоритмів та методів паралельних обчислень являє собою складну задачу (рисунок 3.4). Дії для визначення ефективних способів організації паралельних обчислень можуть полягати в наступному:

– виконати аналіз наявних обчислювальних схем і здійснити їхній поділ (декомпозицію) на частині (підзадачі), які можуть бути реалізовані в значній мірі незалежно одна від одної;

– виділити для сформованого набору підзадач інформаційні взаємодії, які повинні здійснюватися в ході рішення вихідного поставленого завдання;

– визначити необхідну (або доступну) для рішення завдання обчислювальну систему й виконати розподіл набору підзадач між ядрами процесора.

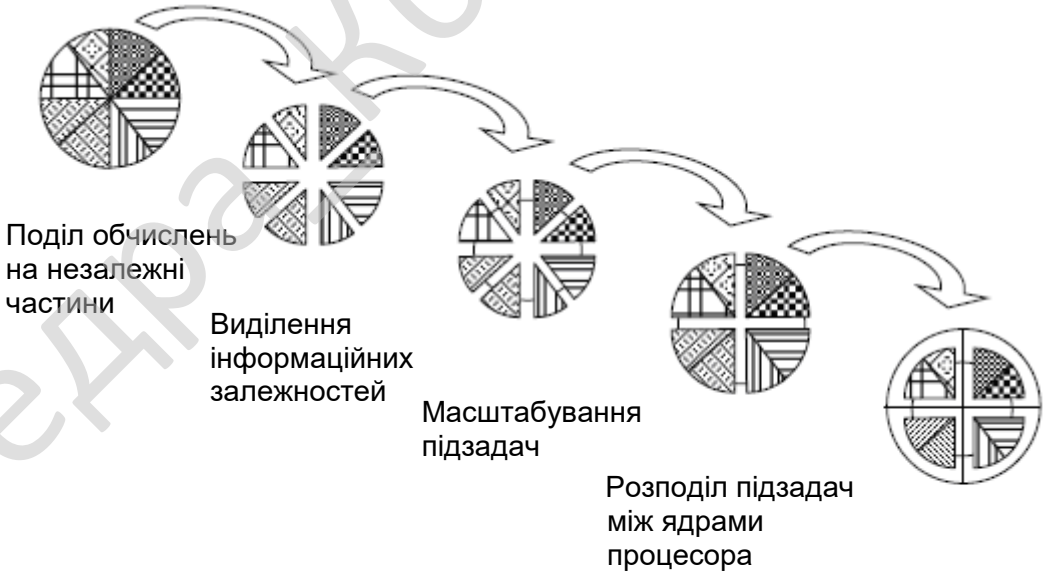

Рисунок 3.4 – Схема розробки паралельних алгоритмів для багатопроцесорних платформ

Обсяг обчислень для кожного ядра процесора повинен бути приблизно однаковий – це дозволить забезпечити рівномірне обчислювальне завантаження (балансування) ядер. Крім того, також зрозуміло, що розподіл підзадач між ядрами повинен бути виконаний таким чином, щоб кількість інформаційних зв'язків (комунікаційних взаємодій) між підзадачами була мінімальним.

Після виконання всіх перерахованих етапів проектування можна оцінити ефективність розроблювальних паралельних методів: для цього звичайно визначаються значення показників якості породжуваних паралельних обчислень (прискорення, ефективність, масштабованість). За результатами проведеного аналізу може виявитися необхідним повторення окремих (у граничному випадку всіх) етапів розробки – слід зазначити, що повернення до попередніх кроків розробки може відбуватися на будь-якій стадії проектування паралельних обчислювальних схем.

Тому часто виконуваною додатковою дією в наведеній вище схемі проектування є коректування складу сформованої множини задач після визначення наявної кількості процесорів – підзадачі можуть бути укрупнені (агреговані) при наявності малого числа ядер або, навпаки, деталізовані в протилежному випадку. У цілому, дані дії можуть бути визначені як *масштабування* розроблювального алгоритму й виділений як окремий етап проектування паралельних обчислень.

Щоб застосувати одержуваний в остаточному підсумку паралельний метод, необхідно виконати розробку програм для рішення сформованого набору підзадач і розмістити розроблені програми по ядрах процесора відповідно до обраної схеми розподілу підзадач. Для проведення обчислень програми запускаються на виконання (програми на стадії виконання звичайно називаються процесами), для реалізації інформаційних взаємодій програми повинні мати у своєму розпорядженні засоби обміну даними (канали передачі повідомлень).

Поділ обчислень на незалежні частини

Вибір способу поділу обчислень на незалежні частини ґрунтується на аналізі обчислювальної схеми рішення вихідного завдання. Вимоги, яким повинен задовольняти обраний підхід, звичайно складаються в забезпеченні рівного обсягу обчислень у виділених підзадачах і мінімуму інформаційних залежностей між цими підзадачами (за інших рівних умов потрібно віддавати перевагу рідким операціям передачі повідомлень великого розміру в порівнянні із частими пересиланнями даних невеликого обсягу). У загальному випадку, проведення аналізу й виділення задач являє собою досить складну проблему – ситуацію допомагає вирішити існування двох типів часто зустрічаємих обчислювальних схем (рисунок 3.5).

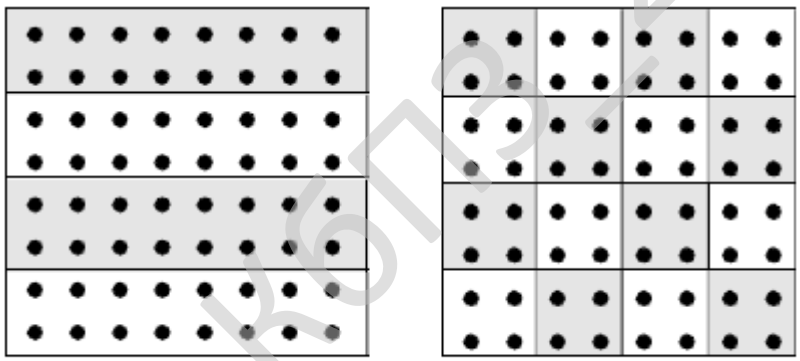

Рисунок 3.5 – Поділ даних матриці: а) стрічкова схема, б) блокова схема

Для великого класу задач обчислення зводяться до виконання однотипної обробки великого набору даних – до такого класу задач ставляться, наприклад, матричні обчислення, чисельні методи рішення рівнянь у частинних похідних та ін. У цьому випадку говорять, що існує паралелізм даних, і виділення підзадач зводиться до поділу наявних даних. Так, наприклад, для розглянутої навчальної задачі пошуку максимального значення при формуванні підзадач вихідна матриця може бути розділена на окремі рядки (або послідовні групи рядків) – так звана *стрічкова схема* поділу даних – або на прямокутні набори елементів – *блокова*

схема поділу даних. Для великої кількості задач поділ обчислень за даними приводить до породження одно-, дво– і тривимірних наборів під задач (рисунок 3.6), у яких інформаційні зв'язки існують тільки між найближчими сусідами (такі схеми звичайно йменуються сітками або ґратами).

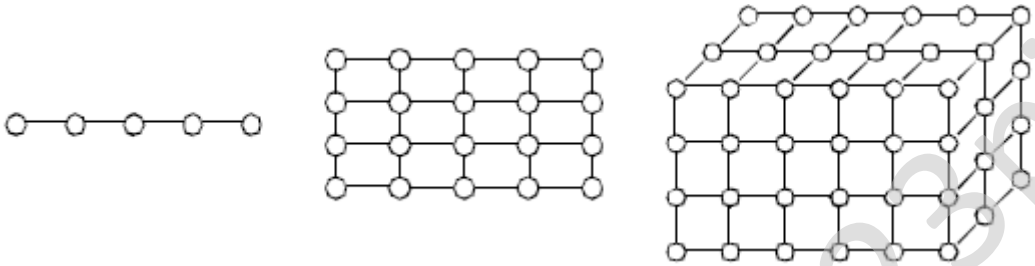

Рисунок 3.6 – Регулярні одно-, дво– і тривимірні структури базових підзадач після декомпозиції даних

Для іншої частини задач обчислення можуть складатися у виконанні різних операцій над тим самим набором даних – у цьому випадку говорять про існування *функціонального паралелізму* (як приклади можна привести задачі обробки послідовності запитів до інформаційних баз даних, обчислення з одночасним застосуванням різних алгоритмів розрахунку й т.п.). Дуже часто функціональна декомпозиція може бути використана для організації конвеєрної обробки даних (так, наприклад, при виконанні яких-небудь перетворень дані обчислення можуть бути зведені до функціональної послідовності введення, обробки й збереження даних).

Важливе питання при виділенні підзадач полягає у виборі потрібного *рівня декомпозиції* обчислень. Формування максимально можливої кількості підзадач забезпечує використання гранично досяжного рівня паралелізму розв'язуваної задачі, однак затрудняє аналіз паралельних обчислень. Застосування при декомпозиції обчислень тільки досить "великих" підзадач приводить до ясної схеми паралельних обчислень, однак може затрудняти ефективне використання досить великої кількості ядер процесора. Можливо розумне сполучення цих двох

підходів може складатися в застосуванні як конструктивні елементи декомпозиції тільки тих підзадач, для яких методи паралельних обчислень є відомими. Так, наприклад, при аналізі задачі матричного множення в якості підзадач можна використовувати методи скалярного добутку векторів або алгоритми матрично-векторного добутку. Подібний проміжний спосіб декомпозиції обчислень дозволить забезпечити й простоту подання обчислювальних схем, і ефективність паралельних розрахунків. Обрані підзадачі при такому підході будемо йменувати далі базовими, які можуть бути *елементарними* (неподільними), якщо не допускають подальшого поділу, або *складними* – у протилежному випадку.

Для розглянутої навчальної задачі достатній рівень декомпозиції може складатися, наприклад, у поділі матриці на множину окремих рядків і одержанні на цій основі набору підзадач пошуку максимальних значень в окремих рядках; породжувана при цьому структура інформаційних зв'язків відповідає лінійному графу (рисунок 3.7).

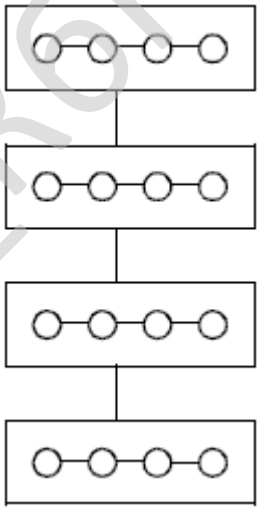

Рисунок 3.7 – Структура інформаційних зв'язків задачі після декомпозиції

Для оцінки коректності етапу поділу обчислень на незалежні частини можна скористатися наступним списком питань:

– Виконана декомпозиція не збільшує обсяг обчислень і необхідний

обсяг пам'яті?

– Чи можливе при обраному способі декомпозиції рівномірне завантаження всіх наявних ядер процесора?

– Чи досить виділених частин процесу обчислень для ефективного завантаження наявних процесорів (з урахуванням можливості збільшення їхньої кількості)?

Виділення інформаційних залежностей

При наявності обчислювальної схеми рішення задачі після виділення базових підзадач визначення інформаційних залежностей між ними звичайно не викликає великих труднощів. При цьому, однак, слід зазначити, що насправді етапи виділення підзадач і інформаційних залежностей досить складно піддаються поділу. Виділення підзадач повинне відбуватися з обліком виникаючих інформаційних зв'язків, після аналізу обсягу й частоти необхідних інформаційних обмінів між підзадачами може знадобитися повторення етапу поділу обчислень.

При проведенні аналізу інформаційних залежностей між підзадачами варто розрізняти:

– *структурні* й *довільні способи взаємодії* – для структурних способів організація взаємодій приводить до формування деяких стандартних схем комунікації (наприклад, у вигляді кільця, прямокутних ґрат і т.д.), для довільних структур взаємодії схема виконуваних операцій передач даних не носять характеру однорідності;

– *статичні* або *динамічні* схеми передачі даних – для статичних схем моменти й учасники інформаційної взаємодії фіксуються на етапах проектування й розробки паралельних програм, для динамічного варіанта взаємодії структура операції передачі даних визначається в ході виконуваних обчислень;

– *синхронні* й *асинхронні способи взаємодії* – для синхронних способів операції передачі даних виконуються тільки при готовності всіх учасників взаємодії й завершуються тільки після повного закінчення всіх комунікаційних

дій, при асинхронному виконанні операцій учасники взаємодії можуть не чекати повного завершення дій по передачі даних. Для представлених способів взаємодії досить складно виділити кращі форми організації передачі даних: синхронний варіант, як правило, більш простий для застосування, у той час як асинхронний спосіб часто дозволяє істотно знизити часові затримки, викликані операціями інформаційної взаємодії.

Для оцінки правильності етапу виділення інформаційних залежностей можна скористатися наступним списком питань:

– Чи відповідає обчислювальна складність підзадач інтенсивності їхніх інформаційних взаємодій?

– Чи є однаковою інтенсивність інформаційних взаємодій для різних підзадач?

– Чи є схема інформаційної взаємодії локальною?

– Чи не перешкоджає виявлена інформаційна залежність паралельному рішенню підзадач?

Масштабування набору підзадач

Масштабування розробленої обчислювальної схеми паралельних обчислень проводиться у випадку, якщо кількість наявних підзадач відрізняється від числа планованих до використання ядер процесора. Для скорочення кількості підзадач необхідно виконати укрупнення (агрегацію) обчислень. Застосовувані тут правила збігаються з рекомендаціями початкового етапу виділення підзадач: обумовлені підзадачі, як і раніше, повинні мати однакову обчислювальну складність, а обсяг та інтенсивність інформаційних взаємодій між підзадачами повинні залишатися на мінімально можливому рівні. Як результат, першими претендентами на об'єднання є підзадачі з високим ступенем інформаційної взаємозалежності.

При недостатній кількості наявних підзадач для завантаження всіх доступних до використання ядер процесора необхідно виконати деталізацію (декомпозицію) обчислень. Як правило, проведення подібної декомпозиції не

викликає яких-небудь труднощів, якщо для базових задач методи паралельних обчислень є відомими.

Виконання етапу масштабування обчислень повинне звестися, в остаточному підсумку, до розробки правил агрегації й декомпозиції підзадач, які повинні параметрично залежати від числа ядер, задіяних для обчислень.

Для розглянутої задачі пошуку максимального значення агрегація обчислень може складатися в об'єднанні окремих рядків у групи (стрічкова схема поділу матриці), при декомпозиції підзадач рядки вихідної матриці можуть розбиватися на кілька частин (блоків).

Контрольні питання для оцінки правильності етапу масштабування:

– Чи не погіршиться локальність обчислень після масштабування наявного набору підзадач?

– Чи мають підзадачі після масштабування однакову обчислювальну й комунікаційну складність?

– Чи відповідає кількість задач числу наявних ядер?

– Чи залежать параметрично правила масштабування від кількості ядер?

Розподіл підзадач між ядрами

Розподіл підзадач між ядрами є завершальним етапом розробки паралельного методу. Треба відзначити, що керування розподілом навантаження для ядер можливий тільки для обчислювальних систем з розподіленою пам'яттю, для систем із загальною пам'яттю розподіл навантаження звичайно виконується операційною системою автоматично. Крім того, даний етап розподілу підзадач між ядрами є надлишковим, якщо кількість підзадач збігається із числом наявних ядер, а топологія передачі даних між ними являє собою повний граф (тобто всі ядра зв'язані між собою прямими лініями зв'язку).

Основний показник успішності виконання даного етапу – *ефективність використання ядер*, обумовлена як відносна частка часу, протягом якого ядра використовувалися для обчислень, пов'язаних з рішенням вхідної задачі. Шляхи досягнення гарних результатів у цьому напрямку залишаються колишніми: як і

раніше, необхідно забезпечити рівномірний розподіл обчислювального навантаження між ядрами й мінімізувати кількість повідомлень, переданих між ними. Точно так само як і на попередніх етапах проектування, оптимальне рішення проблеми розподілу підзадач між ядрами ґрунтується на аналізі інформаційної зв'язності графа "підзадачі – повідомлення". Так, зокрема, підзадачі, що мають інформаційні взаємодії, доцільно розміщати на ядрах, між якими існують прямі лінії передачі даних.

Слід зазначити, що вимога мінімізації інформаційних обмінів між ядрами може суперечити умові рівномірного завантаження. Ми можемо розмістити всі підзадачі на одному ядрі й повністю усунути між'ядерну передачу повідомлень, однак зрозуміло, що завантаження більшості ядер у цьому випадку буде мінімальною.

Рішення питань балансування обчислювального навантаження значно ускладнюється, якщо схема обчислень може змінюватися в ході рішення задачі. Причиною цього можуть бути, наприклад, неоднорідні сітки при рішенні рівнянь у частинних похідних, розрідженість матриць і т.п. Крім того, використовувані на етапах проектування оцінки обчислювальної складності рішення підзадач можуть мати наближений характер, і, нарешті, кількість підзадач може змінюватися в ході обчислень. У таких ситуаціях може знадобитися перерозподіл базових підзадач між ядрами вже безпосередньо в ході виконання паралельної програми. Дані питання є одними з найбільш складних в області паралельних обчислень.

Як приклад дамо коротку характеристику широко використовуваного способу динамічного керування розподілом обчислювального навантаження, звичайно називаємого *схемою "менеджер – виконавець"*. При використанні даного підходу передбачається, що підзадачі можуть виникати й завершуватися в ході обчислень, при цьому інформаційні взаємодії між підзадачами або повністю відсутні, або мінімальні. Відповідно до розглянутої схеми для керування розподілом навантаження в системі виділяється окреме ядро-менеджер, якому доступна інформація про всі наявні підзадачі. Інші ядра системи є виконавцями,

які для одержання обчислювального навантаження звертаються до ядра-менеджера. Породжувані в ході обчислень нові підзадачі передаються назад ядру-менеджерові й можуть бути отримані для рішення при наступних обходах ядер-виконавців. Завершення обчислень відбувається в момент, коли ядра-виконавці завершили рішення всіх переданих їм підзадач, а ядро-менеджер не має якихось обчислювальних робіт для виконання.

Контрольні питання для перевірки етапу розподілу підзадач:

– Чи не приводить розподіл декількох задач на одне ядро до росту додаткових обчислювальних витрат?

– Чи існує необхідність динамічного балансування обчислень?

– Чи не є ядро-менеджер "вузьким" місцем при використанні схеми "менеджер-виконавець"?

Принципи розпаралелювання

Матриці й матричні операції широко використовуються при математичному моделюванні найрізноманітніших процесів, явищ і систем. Матричні обчислення становлять основу багатьох наукових та інженерних розрахунків – серед областей додатків можуть бути зазначені обчислювальна математика, фізика, економіка й ін.

З урахуванням значимості ефективного виконання матричних розрахунків багато стандартних бібліотек програм містять процедури для різних матричних операцій. Обсяг програмного забезпечення для обробки матриць постійно збільшується – розробляються нові ощадливі структури зберігання для матриць спеціального типу (трикутних, стрічкових, розріджених і т.п.), створюються різні високоефективні машинно-машинно-залежні реалізації алгоритмів, проводяться теоретичні дослідження для пошуку більш швидких методів матричних обчислень.

Для багатьох методів матричних обчислень характерним є повторення тих самих обчислювальних дій для різних елементів матриць. Дана властивість свідчить про наявність паралелізму даних при виконанні матричних розрахунків,

і, як результат, розпаралелювання матричних операцій зводиться в більшості випадків до поділу оброблюваних матриць між ядрамивикористовуваної обчислювальної системи. Вибір способу поділу матриць приводить до визначення конкретного методу паралельних обчислень; існування різних схем розподілу даних породжує цілий ряд *паралельних алгоритмів матричних обчислень*.

Найбільш загальні й широко використовувані способи поділу матриць складаються в розбивці даних на *смуги* (по вертикалі або горизонталі) або на прямокутні фрагменти – *блоки* (рисунок 3.8).

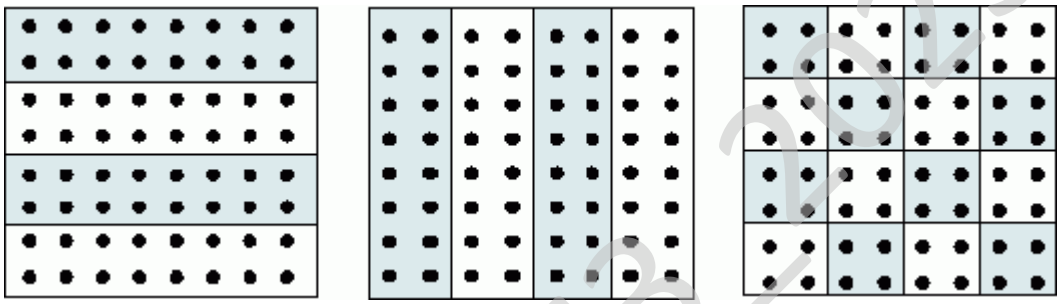

Рисунок 3.8 – Способи розподілу елементів матриці між ядрами обчислювальної системи

Стрічкова розбивка матриці. При стрічковій розбивці кожному ядру виділяється та або інша підмножина рядків (горизонтальна розбивка) або стовпців (вертикальна розбивка) матриці. Поділ рядків і стовпців на смуги в більшості випадків відбувається на безперервній (послідовній) основі. При такому підході для горизонтальної розбивки по рядках, наприклад, матриця A представляється у вигляді (3.1).

$$A = (A_0, A_{1,\ldots,} A_{p-1})^T, A_i = (a_{i0,} a_{i1,} a_{ik-1}), i_j = ik + j, 0 \le j < k, k = m/p, \quad (3.1)$$

де $a_i=(a_{i1},a_{i2},\ldots,a_{in})$, $0 \le i<m$, є i-тий рядок матриці A (передбачається, що кількість рядків m кратне числу ядер процесора p, тобто $m = k \cdot p$).

Інший можливий підхід до формування смуг складається в застосуванні тої або іншої схеми *чергування* (*циклічності*) рядків або стовпців. Як правило, для чергування використовується число процесорів p – у цьому випадку при

горизонтальній розбивці матриця A приймає вид (3.2)

$$A = (A_0, A_{1,\ldots,} A_{p-1})^T,\ A_i = (a_{i0,}\ a_{i1,}\ a_{iк-1}),\ i_j = i + jp,\ 0 \le j < k,\ k = m/p, \quad (3.2)$$

Циклічна схема формування смуг може виявитися корисною для кращого балансування обчислювального навантаження ядер (наприклад, при рішенні системи лінійних рівнянь із використанням методу Гауса).

Блокова розбивка матриці. При *блоковому* поділі матриця ділиться на прямокутні набори елементів – при цьому, як правило, використовується поділ на безперервній основі. Нехай кількість ядер становить $p = s \cdot q$, кількість рядків матриці є кратною s, а кількість стовпців – кратною q, тобто $m = k \cdot s$ и $n = l \cdot q$. Представимо вхідну матрицю A у вигляді набора прямокутних блоків наступним чином:

$$A = \begin{pmatrix} A_{00} & A_{02} & A_{0q-1} \\ A_{s-11} & A_{s-12} & A_{s-1q-1} \end{pmatrix}, \quad (3.3)$$

де A_{ij} – блок матриці, що складається з елементів:

$$A = \begin{pmatrix} a_{i0\,j0} & a_{i0\,j1} & a_{i0\,jl-1} \\ a_{ik-1\,j0} & a_{ik-1\,j1} & a_{ik-1\,jl-1} \end{pmatrix}, \quad \begin{matrix} i_v = ik + v,\ 0 \le v < k, \\ i_u = il + u,\ 0 \le v < l, \end{matrix} \quad (3.4)$$

При такому підході доцільно, щоб обчислювальна система мала фізичну або, принаймні, логічну топологію ґрат з s рядків і q стовпців. У цьому випадку при поділі даних на безперервній основі ядра, сусідні в структурі ґрат, обробляють суміжні блоки вихідної матриці. Слід зазначити, однак, що й для блокової схеми може бути застосоване циклічне чергування рядків і стовпців.

3.2 Розробка структурної схеми

Структурна схема зображена на рисунку 3.9. З неї видно, що в даній бакалаврській роботі візуалізується робота багатоядерного процесору.

Зі схеми бачимо, зо робота з багатоядерним процесором складається з роботи з наступними структурними блоками:

– N ядер, які включають у себе обробку даних у кеш-пам’яті першого

рівня;

– рівень абстракції ресурсів, через який кеш-пам'ять першого рівня взаємодіє з кеш-пам’яттю другого рівня;

– кеш-пам'ять другого рівня.

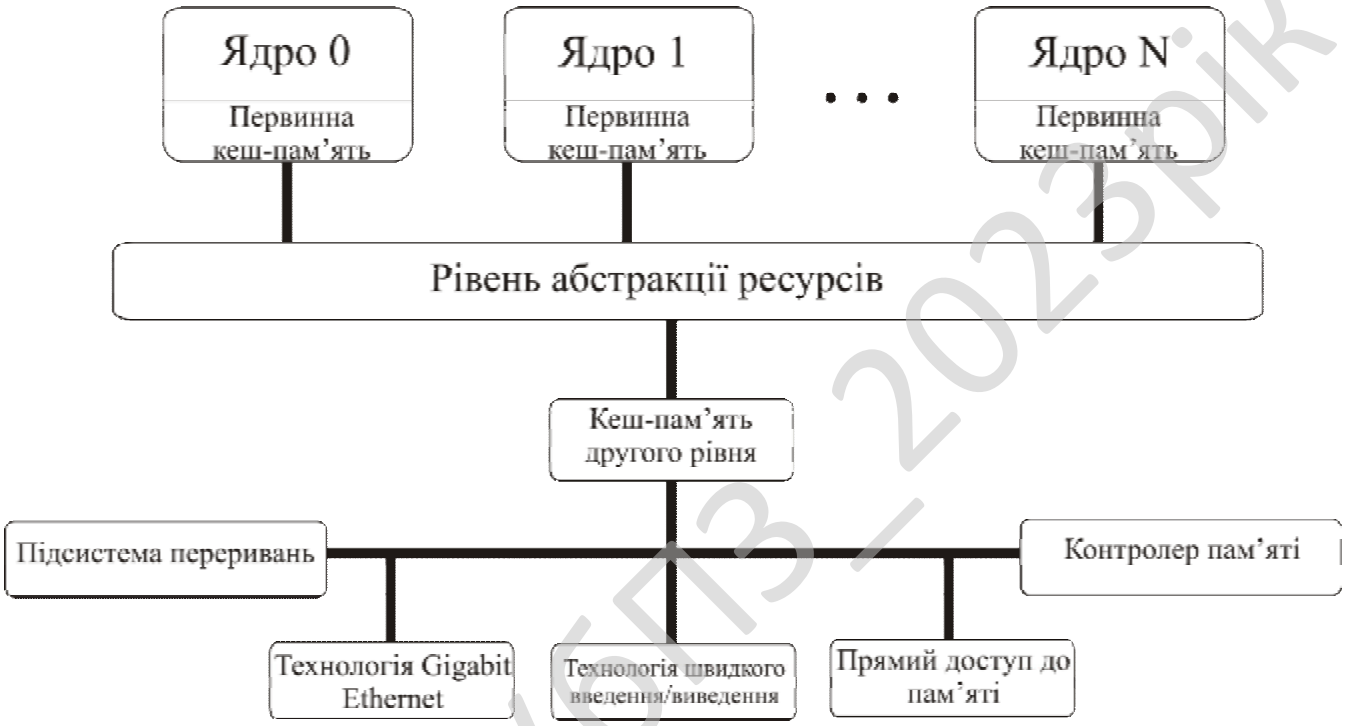

Рисунок 3.9 – Структура типового багатоядерного процесору

Кеш-пам'ять другого рівня структурно взаємодіє з одного боку з кеш-памяттю першого рівня, а з іншого боку з наступними структурними блоками системи:

– підсистема переивань;

– контролером пам’яті;

– блоком прямого доступу до пам’яті (DMA);

– технологією швидкого введення/виведення;

– технологією Gigabit Ethernet.

Розглянемо ці структурні блоки більш детальніше.

Технологія прискорення введення-виводу

Технологія прискорення введення-виводу I/OAT (Intel I/O Acceleration Technology) з'явилася в той час, коли потреби додатків (таких, як електронна комерція, обмін повідомленнями, додатка для кластерів пристроїв зберігання даних і серверів) почали обганяти здатність серверів до реагування і їхньої можливості в частині швидкого й надійного обміну мережними даними з додатками. У той час як продуктивність серверних процесорів і пропускна здатність мереж за останні роки значно виросли, основний метод обміну даними залишився колишнім. Сьогодні вся тяжкість обробки даних, доступу до пам'яті й реалізації протоколів обміну для кожного пакета даних лежить на серверному процесорі. У результаті робота серверних додатків уповільнюється, а час відгуку, надійність і зручність роботи перестають відповідати потребам користувачів.

Для рішення цієї проблеми в технології прискорення введення-виводу Intel застосовується загальплатформний підхід. Завдання керування даними розподіляється між всіма компонентами платформи – процесором, набором мікросхем, мережним контролером і ПЗ. Загальплатформний підхід дозволяє знизити навантаження на процесор і прискорити обмін даними. Завантаженість процесора знижується завдяки тому, що набір мікросхем і мережний контролер одержують можливість зчитувати дані з пам'яті й записувати їх на згадку.

Intel також оптимізувала протокол TCP/IP – відкритий "звід правил", що дозволяє комп'ютерам всіх типів обмінюватися даними, спілкуючись на одній мові. У результаті завантаженість процесорів у серверах архітектури Intel знизилася наполовину, а обчислювальні ресурси вивільнилися для рішення інших завдань. У середньому такий підхід дозволяє прискорити обмін даними між платформою й додатками на 30% і звільняє процесор для виконання іншої обчислювальної роботи.

Крім того, підхід, застосовуваний у технології I/OAT, дозволяє відмовитися від штучних надбудов, застосовуваних в існуючих технологіях, – таких, як механізми розвантаження TCP (TCP offload engine, TOE). Як відомо,

TOE – це спеціалізовані дорогі мікросхеми, призначені для розвантаження процесора при обробці протоколу TCP/IP, але вони не вирішують двох основних проблем, пов'язаних із процесором: зниження системних витрат і організації доступу до пам'яті. У результаті TOE ефективні тільки для таких додатків, де велика інформаційна складова пакетів даних – наприклад, для високопродуктивних систем керування базами даних або для сховищ даних.

Корпорація Microsoft обіцяє забезпечити убудовану підтримку технології I/OAT у майбутніх версіях ОС Windows Server. У цих версіях також буде використовуватися технологія, що дозволяє збалансувати трафік TCP/IP при використанні багатоядерних процесорів.

Платформа під кодовою назвою Richford буде містити два процесори Intel Itanium під кодовою назвою Tukwila, а за ними підуть процесори Intel Itanium наступного покоління під кодовою назвою Poulson.

Перші двухядерні процесори Intel Xeon MP (кодова назва Paxville) були анонсовані в I кварталі 2006 р., а масштабні програми надання зразків цих процесорів підприємствам і розроблювачам ПЗ стартували уже наприкінці 2005р. Платформа під кодовою назвою Reidland базується на процесорах Intel Xeon MP, що містять більше двох ядер, відомих під кодовою назвою Whitefield. Ці багатоядерні процесори були випущені в 2007 р. Процесори Intel Xeon MP призначені для серверів із чотирма й більше процесорами.

Платформа під кодовою назвою Bensley, орієнтована на масові двухпроцесорні сервери, з'явилася в I кварталі 2006 р. і була заснована на двухядерному процесорі Intel Xeon під кодовою назвою Dempsey. Процесори Dempsey також використовуються в продуктивних платформах для робочих станцій (кодова назва Glidewell).

Платформа для цифрового офісу під кодовою назвою Lyndon побудована на наборах мікросхем серії Intel 945/955 і процесорах Pentium 4 5xx/6xx, а також нових двухядерних процесорах Pentium D (кодова назва Smithfield). Платформа

Lyndon підтримує технології Intel Active Management Technology і Virtualization Technology.

Доля системної шини

Щоб повною мірою реалізувати потенціал росту продуктивності, забезпечуваний декількома ядрами, необхідний спосіб надати процесору достатню кількість даних. На думку експертів, існуюча архітектура системної шини Intel здатна задовольнити вимоги максимум чотирьох ядер, залежно від їхньої частоти. У цій архітектурі системна шина зв'язує центральний процесор з основною пам'яттю. Контролер пам'яті, що входить до складу відповідних наборів мікросхем, відповідає за координацію трафіка даних при його передачі з пам'яті в центральний процесор. Деякі компанії, зокрема, AMD, Sun Microsystems і IBM, уже інтегрували контролери пам'яті в кристали центральних процесорів. Нагадаємо, що інтегрований контролер пам'яті скорочує рівень затримки (час, необхідний для передачі порції даних від одного компонента системи іншому).

За прогнозами аналітиків, Intel в остаточному підсумку повинна відмовитися від архітектури системної шини просто для того, щоб підтримувати продуктивність процесорів на необхідному рівні. Рішення корпорації інтенсифікувати розробку своїх багатоядерних архитектур дозволить рано або пізно реалізувати цю концепцію. Так, Intel випустила двухядерні процесори в 2005р., що дозволило зберегти архітектуру системної шини. Але чотырьохядерні процесори, стали першими, у яких був використаний єдиний системний інтерфейс між декількома такими процесорами.

Як думають, інтегрований контролер міг би дозволити Intel збільшити продуктивність серверних додатків, що інтенсивно використовують пам'ять, і забезпечити доставку даних до декількох ядер, запланованим для процесорів Xeon і Itanium цього покоління. Фактично процесор Itanium зразка 2007 р. може мати до восьми ядер.

Втім, експерти не виключають і інший сценарій. Intel може зберегти свою системну шину за рахунок додавання до процесорів так званих модулів арбітражу

шини, які забезпечать спільне використання процесорами ресурсів, у тому числі засобів доступу до пам'яті або систем вводу-виводу. Ряд виробників уже встановлює подібні модулі у свої багатопроцесорні сервери. У багатьох серверах з вісьма або більше процесорами вони фактично згруповані по чотирьох. Для Intel не складе великої роботи організувати той же тип архітектури в процесорі із чотирма й більше ядрами.

3.3 Розробка функціональної схеми

На рисунку 3.10 наведена функціональна схема розробленого, у результаті проведених у бакалаврській роботи досліджень, програмного забезпечення.

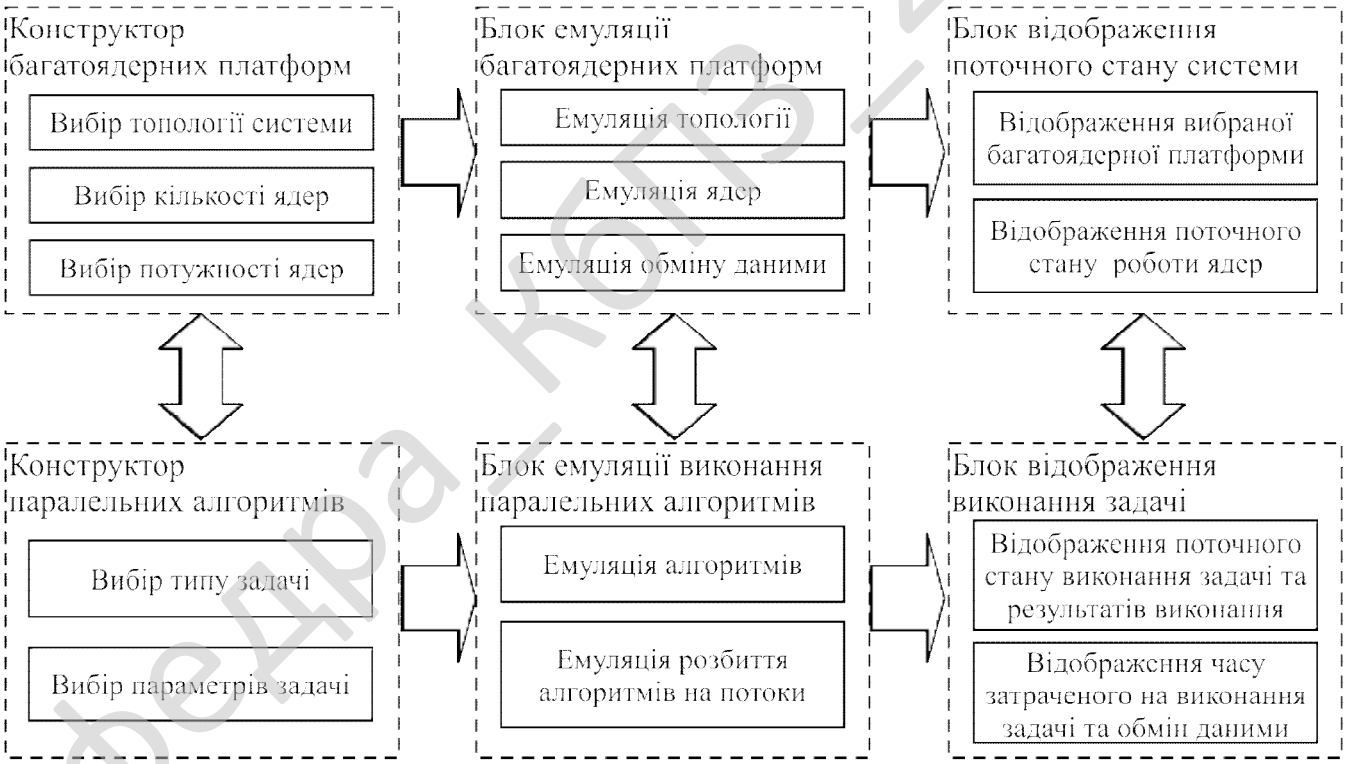

Рисунок 3.10 – Функціональна схема системи

З цієї схеми бачимо, що розроблене програмне забезпечення функціонально складається з наступних основних блоків:

– блок, який конструює багатоядерні платформи;

– блок, який конструює паралельні алгоритми;

– блок емуляції багатоядерних платформ;

– блок емуляції виконання паралельних алгоритмів;

– блок відображення поточного стану системи;

– блок відображення стану виконання задачі.

Усі ці функціональні блоки, які взаємодіють між собою, згідно зв'язків наведених у схемі, складаються, у свою чергу, з функціональних під блоків. Розглянемо це більш детальніше.

Почнемо розгляд з конструктора багатоядерних платформ. Він має у собі наступні функціональні підблоки:

– функціональний підблок вибору топології системи;

– функціональний підблок вибору кількості ядер;

– функціональний підблок вибору потужності ядер.

Розглянемо блок конструктора паралельних алгоритмів. Він має у собі наступні функціональні підблоки:

– функціональний підблок вибору типу задачі;

– функціональний підблок вибору парметрів задачі.

Розглянемо блок емуляції багатоядерних платформ. Він має у собі наступні функціональні підблоки:

– функціональний підблок емуляції топології;

– функціональний підблок емуляції ядер;

– функціональний підблок емуляції обміну ядрами.

Розглянемо блок емуляції виконання паралельних алгоритмів. Він має у собі наступні функціональні підблоки:

– функціональний підблок емуляції алгоритмів;

– функціональний підблок емуляції розбиття алгоритмів на потоки.

Розглянемо блок відображення поточного стану ситеми. Він має у собі наступні функціональні підблоки:

– функціональний підблок відображення вибраної багатоядерної платформи;

– функціональний підблок відображення поточного стану роботи ядер.

Розглянемо блок вдображення виконання задачі. Він має у собі наступні функціональні підблоки:

– функціональний підблок відображення поточного стану виконання задачі та результатів виконання;

– функціональний підблок відображення часу затраченого на виконання задачі та обмін даними.

Таким чином розглянувши склад функціональної схеми розробленої системи перейдемо до розгляду процесів, які взаємодіють у системі.

3.4 Розробка діаграми процесів

На рисунку 3.11 зображена діаграма взаємодії процесів, які відбуваються у розробленій системі.

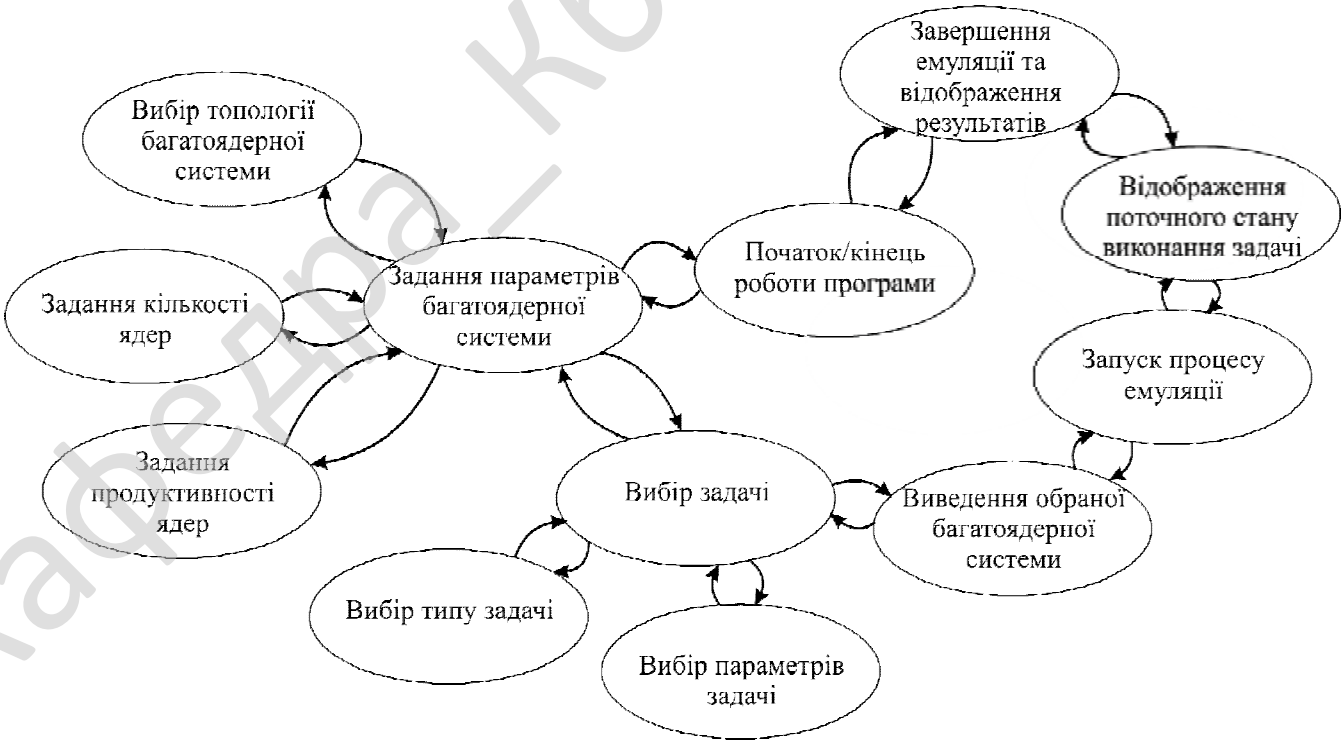

Рисунок 3.11 – Діаграма процесів системи

З цієї діаграми бачимо, що програма починає роботу з запуску процесу початку/кінця програми.

Цей процес взаємодіє з двома основними процесами:

– процес завдання параметрів багатоядерної системи;

– процес роботи емуляції.

Розглянемо ці процеси більш детальніше.

Процес завдання параметрів багатоядерної системи взаємодіє з наступними процесами:

– процес вибору топології багатоядерної системи;

– процес завдання кількості ядер;

– процес завдання продуктивності ядер;

– процес вибору задачі.

Процес вибору задачі у свою чергу взаємодіє з наступними процесами:

– процес вибору типу задачі;

– процес вибору параметрів;

– процес виведення обраної багатоядерної системи.

Останній процес взаємодіє з процесом запуску процесу емуляції. Після запуску процесу емуляції запускається процес відображення на моніторі для користувача поточного стану виконання задачі.

Закінчується робота програми процесом завершення емуляції та відображення результатів дослідження.

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 зображена блок-схема алгоритму виконання основної програми.

З неї бачимо, що робота програми починається з виведення на екран основного вікна програми.

Після цього виконується вибір будувати, або ні багатоядерну систем.

Якщо її потрібно будувати, то відбувається вибір топології багатоядерної платформи.

За цим користувачем вводиться кількість ядер, з яких складається багатоядерна система.

Після чого користувачем вводиться потужність кожного ядра.

Якщо її не потрібно будувати, то преходимо до вибору задачі.

При вибору задачі відбуваються наступні дії:

– вибирається тип задачі;

– відбувається введення користувачем параметрів задачі.

За цим запускається процес емуляції.

Він складається з наступних кроків:

– запуск процесу емуляції;

– виведення поточного стану системи.

Після виконання усіх вищеперерахованих дій програма закінчує свою роботу.

На рисунку 4.2 зображена блок-схема роботи підпрограми емуляції виконання паралельних підзадач.

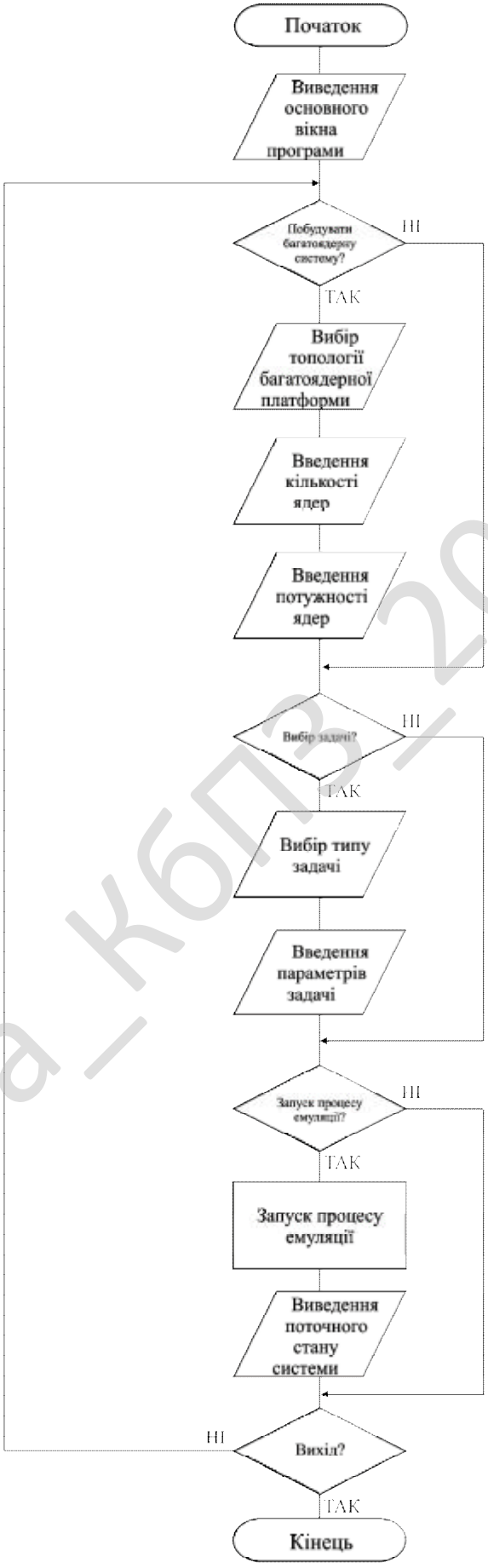

Рисунок 4.1 – Блок-схема роботи основної програми

З блок-схеми бачимо, що емуляції виконання паралельних підзадач виконується наступним чином.

Спершу відбувається ініціалізація змінних.

Після чого, одному з ядра призначується ранг 0, що дає йому право бути «адміністратором» процесу.

Після призначення відбувається підготовка даних.

Задача поділяється на незалежні під задачі, які розсилаються на всі ядра.

За цим починається виконання задачі наступним чином:

– обчислюються під задачі на кожному з ядер;

– якщо потрібен обмін даними, то відбувається синхронізація та обмін даними;

– у іншому випадку перевіряється завершення обчислення задачі.

Якщо обчислення завершені то результати відправляються на ядро з рангом 0. На цьому ядрі відбувається складання результатів підзадач.

Після цього результати виводяться на екран користувачу, й підпрограма закінчує свою роботу.

На рисунку 4.3 зображена блок-схема роботи підпрограми визначення трудомісткості операцій передачі даних між ядрами.

Для поняття принципу визначення трудомісткості наведемо наступні теоретичні подання. При всій розмаїтості виконуваних операцій передачі даних при паралельних способах рішення складних завдань, певні процедури взаємодії багатоядерних процесорів можуть бути віднесені до числа основних комунікаційних дій, або найбільше широко розповсюджених у практиці паралельних обчислень, або тих, до яких можуть бути зведені багато інших процесів прийому-передачі повідомлень. Важливо відзначити також, що в рамках подібного базового набору для більшості операцій комунікації існують процедури, зворотні по дії вихідним операціям (так, наприклад, операції передачі даних від одного ядра всім наявним ядрам відповідає операція прийому в одному ядрі повідомлень від всіх інших ядер).

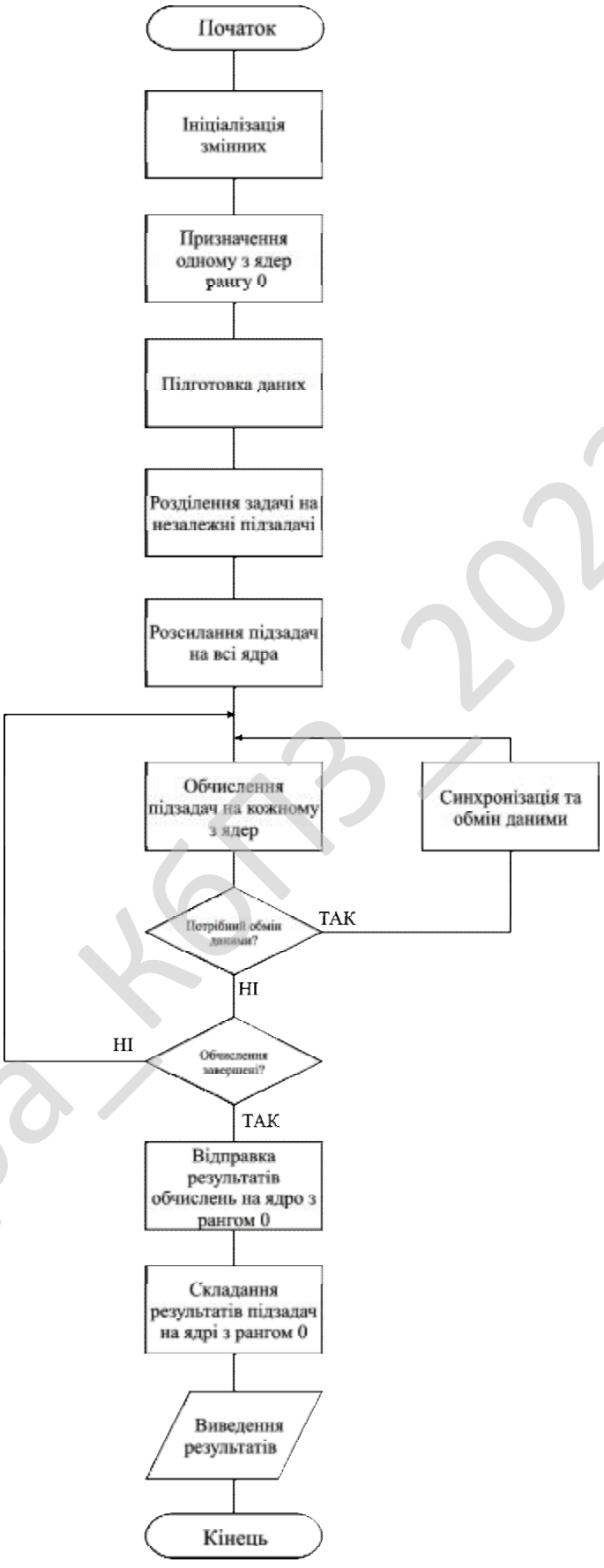

Рисунок 4.2 – Блок-схема роботи підпрограми емуляції виконання паралельних підзадач

Як результат, розгляд комунікаційних процедур доцільно виконувати попарно, оскільки в багатьох випадках алгоритми виконання прямої і зворотної операцій можуть бути отримані виходячи з однакових передумов.

Розгляд основних операцій передачі даних у цій лекції буде здійснюватися на прикладі таких топологій багатоядерних процесорів, як кільце, двовимірні ґрати й гіперкуб. Для двовимірних ґрат буде передбачатися також, що між граничними ядрами в рядках і стовпцях ґрати є канали передачі даних (тобто топологія багатоядерного процесора являє собою тор). Як і раніше, величина m буде означати розмір повідомлення в словах, значення p визначає кількість ядер у багатоядерного процесора, а змінна N задає розмірність топології гіперкуба.

Передача даних між двома ядрами

Трудомісткість даної комунікаційної операції може бути отримана шляхом підстановки довжини максимального шляху (діаметра міжядерних співвідношень) у вираження для часу передачі даних при різних методах комунікації (таблиця 4.1).

Таблиця 4.1 – Час передачі даних між двома ядрами

Топологія	Передача повідомлень	Передача пакетів
Кільце	$t_н + mt_к\lfloor p/2 \rfloor$	$t_н + mt_к + t_c\lfloor p/2 \rfloor$
Ґрати-Тор	$t_н + 2mt_к\lfloor \sqrt{p}/2 \rfloor$	$t_н + mt_к + 2t_c\lfloor \sqrt{p}/2 \rfloor$
Гіперкуб	$t_н + mt_к \log_2 p$	$t_н + mt_к + t_c \log_2 p$

Передача даних від одного ядра всім іншим ядрам багатоядерного процесора

Операція передачі даних (того самого повідомлення) від одного ядра всім іншим ядрам багатоядерного процесора (one-to-all broadcast або single-node broadcast) є одним з найбільше часто виконуваних комунікаційних дій. Двоїста їй операція – прийом на одному ядрі повідомлень від всіх інших ядер багатоядерного процесора (single-node accumulation). Подібні операції використовуються, зокрема, при реалізації матрично-векторного множення,

рішенні систем лінійних рівнянь методом Гаусса, рішенні завдання пошуку найкоротших шляхів і ін.

Найпростіший спосіб реалізації операції розсилання складається в її виконанні як послідовності попарних взаємодій ядер багатоядерного процесора. Однак при такому підході більша частина пересилань є надлишковою й можливо застосування більше ефективних алгоритмів комунікації. Виклад матеріалу буде проводитися спочатку для методу передачі повідомлень, потім – для пакетного способу передачі даних.

Передача повідомлень. Для кільцевої топології ядро – джерело розсилання може ініціювати передачу даних відразу двом своїм сусідам, які, у свою чергу, прийнявши повідомлення, організують пересилання далі по кільцю. Трудомісткість виконання операції розсилання в цьому випадку буде визначатися співвідношенням:

$$t_{нд} = (t_н + mt_к)\lceil p/2 \rceil. \quad (4.1)$$

Для топології типу ґрати-тор алгоритм розсилання можуть бути отримані зі способу передачі даних, застосованого для кільцевої структури багатоядерного процесора. Так, розсилання може бути виконана у вигляді двуетапної процедури. На першому етапі організується передача повідомлення всім ядрам багатоядерного процесора, що розташовується на тій же горизонталі ґрат, що і ядро – ініціатор передачі. На другому етапі ядра, що одержали копію даних на першому етапі, розсилають повідомлення по своїх відповідних вертикалях. Оцінка тривалості операції розсилання відповідно до описаного алгоритму визначається співвідношенням:

$$t_{нд} = 2(t_н + mt_к)\lceil \sqrt{p}/2 \rceil. \quad (4.2)$$

Для гіперкуба розсилання може бути виконана в ході N-етапної процедури передачі даних. На першому етапі ядро-джерело повідомлення передає дані одному зі своїх сусідів (наприклад, по першій розмірності) – у результаті після першого етапу є два ядра, що мають копію пересилаємих даних (даний результат можна інтерпретувати також як розбивка вихідного гіперкуба на два таких

однакових по розміру гіперкуба розмірності N-1, що кожний з них має копію вихідного повідомлення). На другому етапі два ядра, задіяні на першому етапі, пересилають повідомлення своїм сусідам по другій розмірності й т.д. У результаті такого розсилання час операції оцінюється за допомогою вираження:

$$t_{n\partial} = (t_u + mt_k)\log_2 p. \qquad (4.3)$$

Порівнюючи отримані вираження для тривалості виконання операції розсилання, можна відзначити, що найкращі показники має топологія типу гіперкуб; більше того, можна показати, що даний результат є найкращим для обраного способу комунікації за допомогою передачі повідомлень.

Передача пакетів. Для топології типу кільце алгоритм розсилання може бути отриманий шляхом логічного подання кільцевої структури багатоядерного процесора у вигляді гіперкуба. У результаті на етапі розсилання ядро – джерело повідомлення передає дані ядру, що перебуває на відстані p/2 від вихідного ядра. Далі, на другому етапі обоє ядра, що вже мають розсилаються дані після першого етапу, передають повідомлення ядрам, що перебувають на відстані p/4, і т.д. Трудомісткість виконання операції розсилання при такому методі передачі даних визначається співвідношенням:

$$t_{n\partial} = \sum_{i=1}^{\log_2 p} (t_u + mt_k + t_c p/2^i) = (t_u + mt_k)\log_2 p + t_c(p-1) \qquad (4.4)$$

(як і раніше, при досить більших повідомленнях часом передачі службових даних можна зневажити).

Для топології типу ґрати-тор алгоритм розсилання можуть бути отримані зі способу передачі даних, застосованого для кільцевої структури багатоядерного процесора, відповідно до того ж способу узагальнення, що й у випадку використання методу передачі повідомлень. Одержуваний у результаті такого узагальнення алгоритм розсилання характеризується наступним співвідношенням для оцінки часу виконання:

$$t_{n\partial} = (t_u + mt_k)\log_2 p + 2t_c(\sqrt{p} - 1). \qquad (4.5)$$

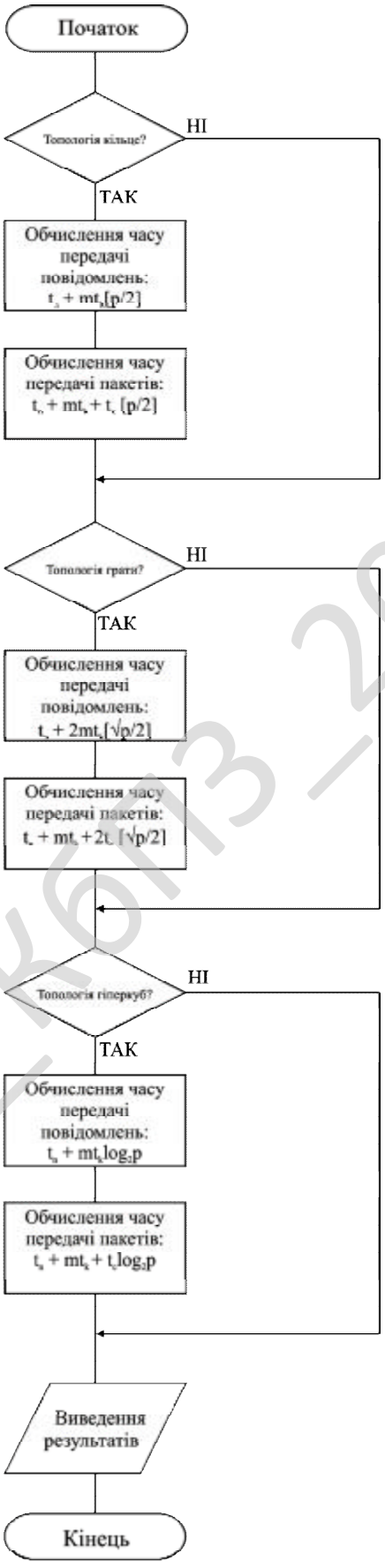

Рисунок 4.3 – Блок-схема роботи підпрограми визначення трудомісткості операцій передачі даних між ядрами

Для гіперкуба алгоритм розсилання (і, відповідно, тимчасові оцінки тривалості виконання) при передачі пакетів не відрізняється від варіанта для методу передачі повідомлень.

4.2 Захист розробленого програмного забезпечення

Дані які використовуються у даній роботі захищаються алгоритмом ДСТУ 7564:2014 («Купина»). В Україні на основі консервативного підходу із залученням відомих і добре досліджених конструкцій була розроблена ґеш-функція, що базується на новому блоковому шифрі „Калина” (ДСТУ 7624:2014).

Національний стандарт ДСТУ 7564:2014 визначає криптографічну функцію ґешування „Купина”, додатковий режим її застосування для формування коду автентифікації повідомлення (імітовставки), а також значення для перевірки реалізацій.

Для скорочення обсягу тексту національного стандарту була застосована математична нотація, що дозволяє отримати точний і компактний запис.

Водночас, такий підхід може ускладнювати сприйняття сутності алгоритму для фахівців, що не мають фундаментальної криптологічної освіти.

У роботі наводиться розгорнутий альтернативний опис функції ґешування „Купина” із позначеннями, традиційними для галузі комп’ютерних наук.

Термінологія та позначення

Вектор ініціалізації – бітова послідовність фіксованої довжини (512 або 1024 біта), що використовується як початкове значення при обчисленні ґеш-значення.

Внутрішній стан – бітова послідовність фіксованої довжини (512 або 1024 біта), що є проміжним значенням на кожній ітерації перетворення функції ґешування, а також вхідним та вихідним значеннями перетворень P і Q; для цих перетворень внутрішній стан подається як матриця розміром $8\times c$ байт.

Ґеш-значення (ґеш-вектор) – бітова послідовність фіксованої довжини ($n = 8 \cdot s$, $s \in \{1,2,...,64\}$), що є результатом роботи функції ґешування.

Доповнення – вставка додаткових біт у кінець повідомлення для отримання кратності довжини бітової послідовності довжині внутрішнього стану функції ґешування.

Повідомлення – бітова послідовність довжини від 0 біт (порожній рядок) до 2^{96}-1 біт.

Функція стиснення – ітеративне перетворення, що відображає l-бітний блок повідомлення та l-бітне значення, отримане функцією стиснення на попередньому кроці, у нове l-бітне значення.

Далі використовуються наступні позначення:

– $\oplus$ – додавання за модулем 2 (XOR);

– 0x – префікс числа, що записане у шістнадцятковій системі числення;

– $a \bmod b$ – ціле невід'ємне число, що дорівнює залишку від ділення цілого числа a на натуральне число b;

– B_i – i -й байт вхідної послідовності;

– C^i – константа перетворення XORRoundKey або Add64RoundKey для i -го циклу;

– c – кількість стовпців внутрішнього стану в матричному поданні;

– φ – функція стиснення;

– H – визначена у стандарті функція ґешування;

– $H(M)$ – результат обчислення функції ґешування для повідомлення M (ґешзначення);

– IV – вектор ініціалізації;

– l – розмір внутрішнього стану функції ґешування (у бітах), $l \in \{512,1024\}$;

– M – повідомлення;

– m_i – i -й блок повідомлення M;

– n – довжина обчисленого ґеш-значення;

– N – довжина повідомлення M без доповнення;

– P, Q – складові перетворення функції стиснення;

– P_{512} – перетворення P для 512-бітного внутрішнього стану;

– P_{1024} – перетворення P для 1024-бітного внутрішнього стану;

– Q_{512} – перетворення Q для 512-бітного внутрішнього стану;

– Q_{1024} – перетворення Q для 1024-бітного внутрішнього стану;

– r – кількість ітерацій у перетвореннях P і Q ($r = t$ в ДСТУ 7564:2014);

– S – внутрішній стан ґеш-функції;

– t – кількість блоків m, з яких складається повідомлення M, включаючи доповнення;

– v_i – i -й біт вхідної послідовності;

– Ω – завершальне перетворення;

– Купина-n – режим використання функції ґешування з усіченням обчисленого ґешзначення до розміру n біт.

Загальні положення

Під функцією ґешування H розуміється залежне від вектора ініціалізації IV відображення послідовності біт M у ґеш-значення $H(M)$ фіксованої довжини n.

ДСТУ 7564:2014 визначає функцію ґешування, яка виконує перетворення «Купина-256» або «Купина-512», що забезпечують обчислення ґеш-значення з довжинами 256 або 512 біт відповідно.

Ґеш-значення довжиною 256 бітів додатково може бути усічено до бітової послідовності довжиною від 8 до 248 біт з кроком у 8 біт, 512 бітів може бути усічене до бітової послідовності довжиною від 264 до 504 біт з кроком у 8 біт.

Режим роботи для формування ґеш-значення довжиною n біт позначається як «Купина-n».

Основними режимами роботи функції ґешування, що рекомендуються до застосування, є «Купина-256», «Купина-384» і «Купина-512».

Структура перетворення

Функція ґешування, визначена в ДСТУ 7564:2014, формує ґеш-значення для повідомлення, що складається з бітової послідовності довжини від 0 біт (порожній рядок) до 2^{96}-1 біт.

При формуванні ґеш-значення повідомлення доповнюється, далі поділяється на l-бітні блоки $m_0,\ldots, m_t$, після чого виконується обробка кожного блоку шляхом ітеративного виконання функції стиснення φ.

При цьому формуються значення $h_i = \varphi(h_{i-1}, m_i)$ де $i = 1,\ldots,t$, а початкове значення $h_0 = \text{IV}$.

Після обробки останнього блоку повідомлення результуюче ґеш-значення обчислюється як $H(M) = \Omega(h_t)$, де Ω – завершальне перетворення, що повертає n -бітне значення, кратне 8 ($0 < n \le l/2$).

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розроблене програмне забезпечення призначене для системи візуалізації роботи багатоядерних платформ. Программа являє собою інтегроване середовище візуалізації роботи багатоядерних платформ.

Розроблена система дозволяє імітувати паралельні обчислення на багатоядерному комп'ютері з візуалізацією процесу паралельного рішення обчислювальної задачі.

Можливості програми:

– визначення топології багатоядерної обчислювальної системи, завдання числа ядер у цій топології, встановлення продуктивності процесорів, вибір характеристики комунікаційного середовища й спосіб комунікації;

– здійснення постановки обчислювальної задачі, для якої в складі системи є реалізовані паралельні алгоритми рішення, виконання завдання параметрів задачі;

– вибір паралельного методу для рішення обраної задачі;

– встановлення параметрів візуалізації для вибору бажаного темпу демонстрації, способу відображення даних, ступеня, що пересилаються між ядрами, детальності візуалізації виконуваних паралельних обчислень;

– виконання емуляції паралельного рішення обраної задачі;

– накопичування й аналіз результатів виконаних емуляцій.

Програма має простий і інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1.

У лівій частині вікна та в меню користувача розміщені параметри, що може змінювати користувач та команди, які він може запускати.

У правій частині вікна зображується процес емуляції. У полі «Емуляція багатоядерної платформи» відображаються ядра процесора, з'єднані в обрану

топологію та стан виконання алгоритму.

Стрічковий індикатор відображає поточну стадію виконання алгоритму. У рядках «Загальні витрати часу» і «Витрати на передачу даних» представлені часові характеристики алгоритму.

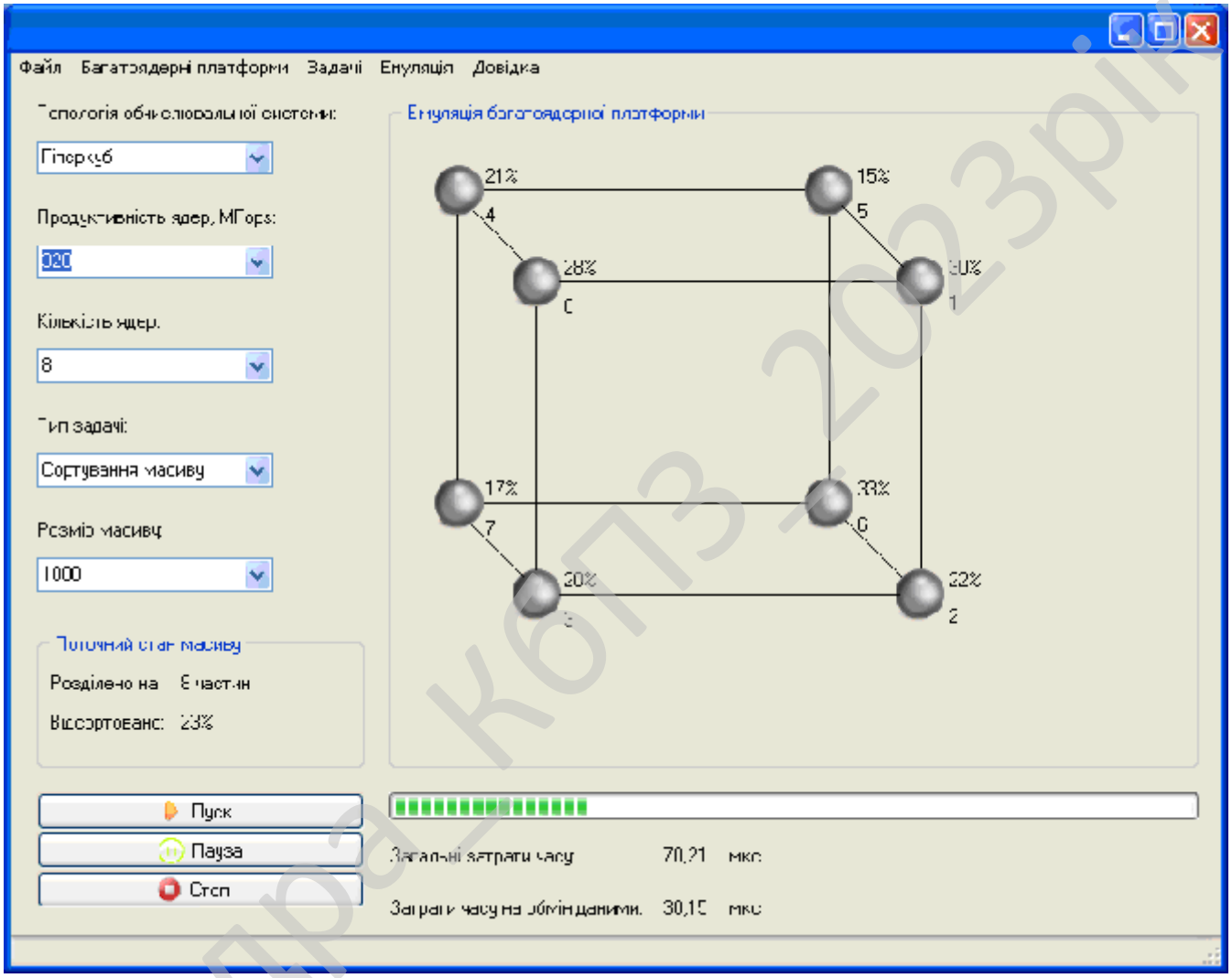

Рисунок 5.1 – Головне вікно програми

Формування моделі обчислювальної системи

Для формування моделі обчислювальної системи необхідно визначити її топологію, кількість ядер, продуктивність кожного ядра й характеристики комунікаційного середовища.

Вибір топології мережі

Топологія мережі передачі даних визначається структурою ліній комутації між ядрами обчислювальної системи. У розробленій системі забезпечується підтримка наступних типових топологій:

– **повний граф** – система, у якій між будь-якою парою ядер існує пряма лінія зв'язку;

– **лінійка** – система, у якій кожне ядро має лінії зв'язку тільки із двома сусідніми (з попереднім і наступним) ядрами;

– **кільце** – дана топологія виходить із лінійки ядер з'єднанням першого й останнього ядер лінійки;

– **ґрати** – система, у якій граф ліній зв'язку утворять прямокутну двовимірну сітку;

– **гіперкуб** – дана топологія представляє окремий випадок структури N-мірних ґрат, коли по кожній розмірності сітки є тільки два ядра (тобто гіперкуб містить 2^N процесорів при розмірності N);

Правила використання системи:

1. **Запуск системи**. Для запуску системи виділіть файл Program_Model_MultiCore.exe й виконаєте подвійне клацання лівою кнопкою миші (або натисніть клавішу Enter). Далі виберіть команду «Створити» (пункт меню «Файл»).

2. **Вибір топології обчислювальної системи**. Для вибору топології обчислювальної системи варто скористатися командою «Топологія» пункту меню «Багатоядерні платформи», або полем «Топологія обчислювальної системи».

Завдання кількості ядер

Для обраної топології система дозволяє встановити необхідну кількість ядер. Виконуваний при цьому вибір конфігурації системи здійснюється відповідно до типу використовуваної топології. Так, наприклад, число ядер у двовимірних ґратах повинне бути повним квадратом (розміри ґрат по горизонталі й вертикалі збігаються), а число ядер у гіперкубі – ступенем числа 2.

Визначення продуктивності ядер

Під продуктивністю ядер в розробленій системі розуміється кількість операцій із плаваючою точкою, що ядро може виконати за секунду (floating point operations per second – flops). Важливо відзначити, що при побудові оцінок часу виконання емуляції передбачається, що всі машинні команди є однаковими й відповідають одній і тій же операції із плаваючою точкою.

Для вибору числа ядер необхідно використати команду «Кількість ядер» пункту меню «Багатоядерні платформи», або скористатися полем з такою ж назвою на головному вікні програми.

Для завдання продуктивності ядер, що становлять багатоядерну обчислювальну систему, варто виконати команду «Продуктивність ядер» пункту меню «Багатоядерні платформи», або скористатися полем з такою ж назвою на головному вікні програми.

Вибір задачі та її параметрів

Вибір задачі та її параметрів здійснбється за допомогою пункту меню «Задачі», або за допомогою поля «Тип задачі» на головному вікні програми та додаткових полів, що з'являються після вибору типу задачі і стосуються її параметрів.

Доступні наступні задачі:

- сортування масиву;
- множення матриць.

Процес емуляції

Для управління процесом емуляції на головному вікні програми розміщені кнопки «Пуск», «Пауза» та «Стоп» також ці команди продубльовані у пункті меню «Емуляція».

Поточний стан процесу емуляції відображається у полі «Емуляція багатоядерної платформи». В залежності від того яка здача виконується відображаються наступні значення:

- «Поточний стан масиву» при виконанні алгоритму *сортування*;

– «Результат множення матриць» при виконанні матричного множення;

– «Результат обробки графа» при виконанні алгоритмів на графах.

Довідка про програму

Коротку доводку про розроблену програму та її автора можна переглянути вибравши підпункт «Про програму…» з меню «Довідка», після чого з'явиться вікно зображене на рисунку 5.2.

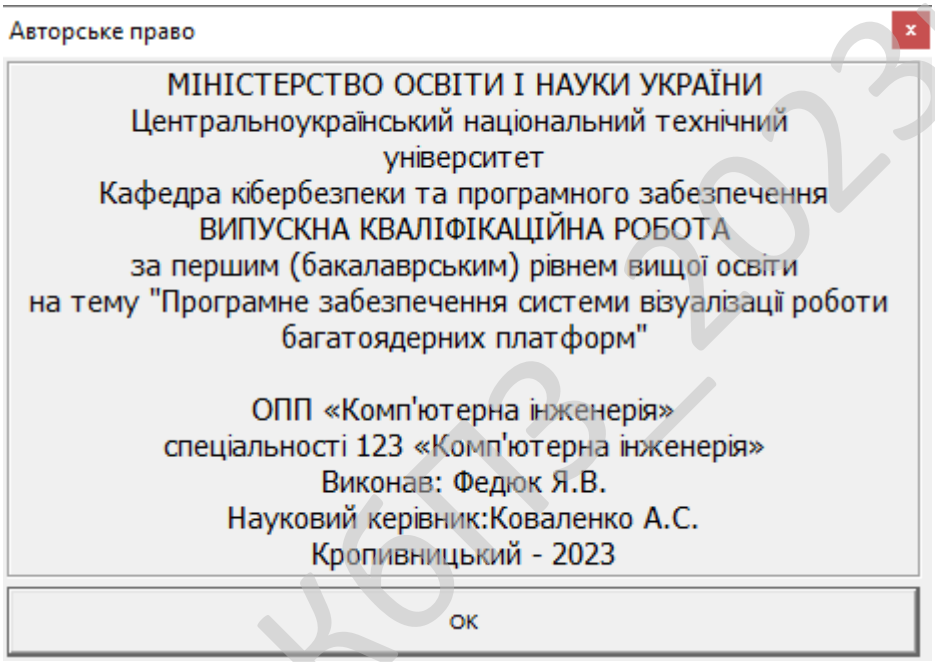

Рисунок 5.2 – Довідка про програму

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи візуалізації роботи багатоядерних платформ.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем візуалізації роботи багатоядерних платформ.

– Досліджена система візуалізації роботи багатоядерних платформ.

– На основі отриманих результатів досліджень створена програмна реалізація системи візуалізації роботи багатоядерних платформ.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання візуалізації роботи багатоядерних платформ.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Delphi 10. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи візуалізації роботи багатоядерних платформ. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне

програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ДСТУ 7564:2014.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kovalenko O., Popereshnyak S., Grinenko S., Grinenko O., Radivilova T. «Methods for Assessing the Maturity Levels of Software Ecosystems». *CEUR Workshop Proceedings* Volume 2654, 2019, Pages 251-261. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=5633fba897776a6e0f3d5633fbcd3fbe **(Scopus)**.

2. Kovalenko O., Drieieva H., Simakhin V., Bondar S., Drieiev O., Zhumadilova M. «Multifractal Properties of Traffic Generator Based on Markov Chains ». *CEUR Workshop Proceedings* Volume 2588, 2019, Pages 567-579. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=176e2cada8976a6e0f3d5633fbcd3fbe **(Scopus)**.

3. Kovalenko Oleksandr Qualitative risk analysis of software development / Oleksandr Kovalenko, Jamil Al-Azzeh, Oleksii Smirnov, Anna Kovalenko, Serhii Smirnov // Asian Journal of Information Technology. – Volume 17 Issue 3. – Medwell Journals. – 2018. – P. 218-230. ISSN: 1682-3915. URL: http://medwelljournals.com/abstract/?doi=ajit.2018.218.230 Doi: ajit.2018.218.230

4. Kovalenko Oleksandr, The mathematical model of the testing technology for DOM XSS vulnerabilities / O. Kovalenko, O. Smirnov, A.Kovalenko, S. Smirnov, V. Vialkova // Scientific & practical cyber security journal (SPCSJ) Volume 2 Issue 1, P. 22-28. Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2018 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/04-3-o.kovalenko-a.kovalenko-o.smirnov-s.smirnov-v.vialkova.pdf

5. Коваленко А.В. Технология тестирования DOM XSS уязвимости / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Scientific & practical cyber security journal (SPCSJ) Volume 1. Issue 1. P. 38-45 Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2017 ISSN: 2587-

4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/8-dom-xss-testing-technology-vulnerabilities.pdf

6. Коваленко О.В. Моделі та методи розроблення програмного забезпечення комп'ютерних систем для підвищення безпеки даних: монографія / О.В. Коваленко // К.: Вид. «КОД» – 2019. – 305 с.

7. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

8. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

9. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

10. Коваленко А.В. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник наукових праць "Системи обробки інформації". – Випуск 4(120). – Х.: ХУПС – 2014. – С. 161-164.

11. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 5(142). – Х.: ХУПС – 2016. – С. 153-157.

12. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць "Системи обробки інформації". –

Випуск 3(140). – Х.: ХУПС – 2016. – С. 40-42.

13. Коваленко А.В. Метод качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

14. Коваленко А.В. Метод количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

15. Коваленко А.В. Использование псевдобулевых методов бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

16. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 2 (38). – Полтава: ПолтНТУ. – 2016. – С. 93-100.

17. Коваленко А.В. Технология тестирования уязвимости к SQL инъекциям / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 5 (45). – Полтава: ПолтНТУ. – 2017. – С. 66-71.

18. Коваленко А.В. Масштабирование имитационной модели технологии тестирования безопасности / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (46). – Полтава: ПолтНТУ. – 2017. – С. 181-184.

19. Коваленко А.В. Имитационная модель технологии тестирования безопасности Web-приложений / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 1 (47). – Полтава: ПолтНТУ. – 2018. – С. 114-123.

20. Коваленко О.В. Методи якісного аналізу та кількісної оцінки ризиків розроблення програмного забезпечення/ О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 3 (49). – Полтава: ПолтНТУ. – 2018. – С. 116-125.

21. Коваленко О.В. Управління ризиками розроблення програмного

забезпечення за умови обмеженості коштів виділених на усунення помилок безпеки/ О.В. Коваленко // Техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Випуск 31. – Кропивницький: ЦНТУ. – 2018. – С. 128-140.

22. Коваленко О.В. GERT-мережевий синтез технології тестування на вразливість WEB-додатків/ О.В. Коваленко // Захист інформації. – Випуск 20(2) – К.: НАУ. – 2018. – С. 89-94.

23. Коваленко О.В. Імітаційна модель технології тестування безпеки на основі положень теорії масштабування / О.В. Коваленко // Безпека інформації. – Випуск 24 (2). – К.: НАУ. – 2018. – С. 110-117.

24. Коваленко О.В. Оцінка ефективності технології тестування безпеки / О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 2, 2018. – С. 137-141

25. Коваленко О.В. Методи та засоби управління безпекою додатків / О.В. Коваленко // Інформаційно-керуючі системи на залізничному транспорті. №4, 2018. – С. 41-44.

26. Коваленко О.В. Розробка інформаційної технології передтестової компіляції та розподілу доступу / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 4 (50). – Полтава: ПолтНТУ. – 2018. – С. 115-119.

27. Коваленко О.В. Аналіз та дослідження інформаційних технологій розробки програмного забезпечення/ О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 5, 2018. – С. 131-137.

28. Коваленко О.В. Удосконалений метод управління ризиками розроблення програмного забезпечення на основі напівмарковської моделі прийняття рішень/ О.В. Коваленко // Сучасні інформаційні системи. – Випуск 2(3). – Харків. – 2018. – С. 41-48.

29. Коваленко О.В. Математичні моделі технології тестування DOM XSS

вразливості та вразливості до SQL ін'єкцій / О.В. Коваленко // Вісник Черкаського державного технологічного університету. Серія : Технічні науки №4, 2018. – С. 29-36.

30. Коваленко О.В. Математична модель технології тестування вразливості до SQL ін’єкцій / О.В. Коваленко // Системи управління, навігації та зв’язку. – Випуск 6 (58). – Полтава: ПолтНТУ. – 2019. – С. 43-47.

31. Коваленко О.В. Математична модель технології тестування комплексу DOM XSS вразливостей для аналітичної оцінки часових витрат / О.В. Коваленко // Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 173-180, 2019.

32. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць II міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 24-27 лютого 2016 р. – Київ: Європейський університет. – 2016. – С. 138-139.

33. Коваленко А.В. Анализ и оценка рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез «Securitea informationala 2015-2016». Conferenta internationala (editia a XII-a). Chisinau. Moldova. 3 martie 2016. – Chisinau: ADSEM. – 2016. – P. 96-102.

34. Коваленко А.В. Исследование источников и причин риска разработки программного обеспечения, этапов и работ, при выполнении которых возникает риск / А.В. Коваленко, А.А. Смирнов // Збірник тез VII всеукраїнської науково-практичної конференції "Інформатика та системні науки (ІСН-2016)". м. Полтава. 10-12 березня 2016 р. – Полтава.: ПУЕТ – 2016. – С. 264-266.

35. Коваленко А.В. Оценка показателя чистой приведенной стоимости для количественной оценки рисков проекта разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних

систем”. м. Київ. 10-11 березня 2016 р. – Київ: КНУ ім. Тараса Шевченко – 2016. – С. 81-82.

36. Коваленко А.В. Методика структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 24-25 березня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 71-72.

37. Коваленко А.В. Методы качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2016). м. Харків. 30 березня – 1 квітня 2016 р. – Харків: НТУ «ХПІ». – 2016. – С. 6-7.

38. Коваленко А.В. Структурная идентификация рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 15-16 квітня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 175-182.

39. Коваленко А.В. Исследование разработанной методики структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез VIII міжнародної науково-практичної конференції “Проблеми і перспективи розвитку IT-індустрії”. м. Харків. 28-29 квітня 2016 р. – Харків: ХНЕУ. – 2016. – С. 49.

40. Коваленко А.В. Исследование дерева рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез III міжнародної науково-практичної конференції «Інформаційна та економічна безпека» (INFECO-2016)». м. Харків. 28-30 квітня 2016 р. – Харків: ХННІ ДВНЗ «УБС». – 2016. – С. 174-178.

41. Коваленко А.В. Методы качественного анализа и количественной

оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Сборник тезисов XII международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 30 мая – 02 июня 2016 г – Варна. ТУВ. – 2016. – С. 585-589.

42. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Матеріали Всеукраїнської науково-практичної конференції «Кібербезпека в Україні: правові та організаційні питання». м. Одеса, 21 жовтня 2016 р. – Одеса : ОДУВС, 2016. – С.146-148.

43. Коваленко А.В. Метод управления рисками разработки программного обеспечения с использованием псевдобулевых методов бивалентного программирования / А.В. Коваленко, А.А. Смирнов // Матеріали Всеукраїнської науково-практичної конференції «Актуальні задачі та досягнення у галузі кібербезпеки». м. Кропивницький, 23-25 листопада 2016 року – Кропивницький: ЦНТУ, 2016. – С. 162.

44. Коваленко А.В. Псевдобулевые методы бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко, С.А. Смирнов // Збірник наукових праць III міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 22-25 лютого 2017 р. – Київ: Європейський університет. – 2017. – С. 158-162.

45. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез II науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем”. м. Київ. 23-24 березня 2017 р. – Київ: КНУ ім. Тараса Шевченко – 2017. – С. 203-205.

46. Коваленко А.В. Алгоритмы анализа уязвимостей при управлении рисками разработки программного обеспечения / А.В. Коваленко,

А.А. Смирнов, А.С. Коваленко // Conferenta internationala (editia a XIII-a). «Securitea informationala 2017». Chisinau. Republic of Moldova. 4-5 aprilie 2017. – Chisinau: ADSEM. – 2017. – P. 19-22.

47. Коваленко А.В. Алгоритм анализа DOM XSS уязвимости при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез дев'ятнадцятого міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кропивницький 7-8 квітня 2017 р. – Кропивницький: ГЛА НАУ. – 2017. – С. 125-127.

48. Коваленко А.В. Алгоритм анализа уязвимости SQL Injection для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез другої міжнародної науково-технічної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2017). м. Харків.10-12 квітня 2017 р. – Харків: НТУ «ХПІ». – 2017. – С. 27.

49. Коваленко А.В. Метод управления рисками разработки программного обеспечения на основе алгоритмов анализа уязвимостей / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 20-22 квітня 2017 р. Кіровоград: КНТУ. – 2017. – С. 92.

50. Коваленко А.В. Алгоритмы анализа DOM XSS уязвимости и уязвимости SQL Injection при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез IX міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 20-21 квітня 2017 р. – Харків: ХНЕУ. – 2017. – С. 61.

51. Kovalenko O.V. Method of testing the dom xss vulnerability / Kovalenko Oleksandr, Kovalenko Anna, Smirnov Oleksii, Smirnov Serhii // International

Conference «information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. – 2017. – P. 7.

52. Коваленко О.В. Метод тестування DOM XSS уразливості / О.В. Коваленко, О.А. Смірнов, А.С. Коваленко, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп’ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті». м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

53. Коваленко О.В. GERT-модель технології тестування DOM XSS уразливості / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць IV міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 21-24 лютого 2018 р. – Київ: Європейський університет. – 2018. – С. 65-70.

54. Коваленко О.В. Технології тестування уразливостей Web-застосунків з використанням GERT-моделі / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної конференції "Комп'ютерні інтелектуальні системи та мережі (КІСМ-2018)". м. Кривий Ріг. 21-23 березня 2018 р. – Кривий Ріг.: ДВНЗ КНУ – 2018. – С. 227-230.

55. Коваленко А.В. Тестирование уязвимости Web-приложений к атаке вида межсайтовый скриптинг / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез «Securitea informationala 2018». Conferenta internationala (editia a XIV-a). Chisinau. Moldova. 20-21 martie 2018. – Chisinau: ADSEM. – 2018. – P. 54-56.

56. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез X міжнародної науково-практичної

конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 19-20 квітня 2018 р. – Харків: ХНЕУ. – 2018. – С. 38.

57. Коваленко О.В. Розробка методу передтестової компіляції й розподілу доступу / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницькй. 19-20 квітня 2018 р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215.

58. Коваленко О.В. Оцінка ефективності технологій тестування безпеки уразливостей DOM XSS й SQL-ін'єкцій / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Сборник тезисов XIV международной конференции "Стратегия качества в промышленности и образовании", Варна, Болгария. 04-07 июня 2018 г – Варна. ТУВ. – 2018. – С. 271-274.

59. Коваленко О.В. Аналіз основних підходів математичного моделювання та методологій для забезпечення максимальних показників безпеки програмного забезпечення / О.В. Коваленко, А.С. Коваленко // Збірник наукових праць всеукр. наук.-практ. конф. здобувачів вищої освіти й молодих учених «Комп’ютерна інженерія і кібербезпека : досягнення та інновації, м. Кропивницькй. 27-29 листопада 2018

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи візуалізації роботи багатоядерних платформ.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 7-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи візуалізації роботи багатоядерних платформ.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи візуалізації роботи багатоядерних платформ;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Delphi 10.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 86 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

11.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 7.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Коваленко А.С.

Програмне забезпечення системи візуалізації роботи багатоядерних платформ

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 38

Літера: РП

Кропивницький – 2023 року

Основний файл проекту розробленого програмного забезпечення (Program_Model_MultiCore.dpr)

```
program ProgramModelMultiCore;

uses
  Forms,
  Messages,Graphics,
  Windows,Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Report.pas' {Form2},
  Unit3 in 'fset.pas' {Form3},
  Unit4 in 'data.pas' {Form4},
  Unit5 in 'MultiCoreEmulation.pas' {Form5},
  Unit6 in 'U6.pas' {Form6},
  U_splash in 'U7.pas' {U_Form_Splash}.

const
WM_CONST1=WM_USER+322; WM_CONST2=WM_USER+105; WM_CONST3=WM_USER+99;

{$R *.res}

///Розробив Федюк Ярослав Вячеславович, ЦНТУ, 2023

begin
  Application.HintPause:=400; Application.HintHidePause:=1000;
  Application.Title:= ProgramModelMultiCore';
  Application.Initialize;
try
    U_Form_Splash:=TU_Form_Splash.Create(Application);
    U_Form_Splash.Show;
    U_Form_Splash.Update;
  Sendmessage(U_Form_Splash.Handle,WM_MY,0,'Start');
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.CreateForm(TForm4, Form4);
  Sendmessage(U_Form_Splash.Handle,WM_MY,0,'End');
  Application.CreateForm(TForm5, Form5);
Finally U_Form_Splash.free; end;
  Application.Run;
end.
```

Файл форми взаємодії та відображення звіту роботи
(Report.pas)

```
unit Report;

interface

uses
Windows, glReport, glCapt, glBevel, glPage, Printers,
glLabel, glRuler, Mask, glLBox, dsgnintf, Spin, geRPEdit,
Menus, ExtCtrls, StdCtrls, Buttons, ComCtrls, Controls, Dialogs,
Forms, Classes,Sysutils;
///Розробив Федюк Ярослав Вячеславович, ЦНТУ, 2023

type

tmyRep_Property = class( TPropertyEditor )
function GetAttributes: TPropertyAttributes; override;
function GetValue: string; override;
procedure Edit; override;
end;

tmyRep_Editor = class(TComponentEditor)
procedure ExecuteVerb(Index: Integer); override;
function GetVerb(Index: Integer): string; override;
function GetVerbCount: Integer; override;
end;

tmyReportEditor = class(TComponent)
FReport: tmyReport;
protected
procedure Notification(AComponent: TComponent; Operation: TOperation); override;
public
procedure Preview;
procedure Edit;
published
property Report: tmyReport read FReport write FReport;
end;

tmyRepEditor = class(TForm) //основний класс взаємодії
OpenDialog1: TOpenDialog; SaveDialog1: TSaveDialog;
PM_Control: TPopupMenu; N_Linktofile: TMenuItem;
N1: TMenuItem; N2: TMenuItem;
PC: tmyPageControl; Panel1: TPanel;
Bevel4: TBevel; P_Sides: TPanel;
Panel2: TPanel; glBevel1: tmyBevel;
Bevel2: TBevel; Bevel1: TBevel;
B_Label: TSpeedButton; sb_Open: TSpeedButton;
sb_Save: TSpeedButton; sb_Preview: TSpeedButton;
sb_Book: TSpeedButton; sb_Album: TSpeedButton;
Bevel3: TBevel; sb_OLE: TSpeedButton;
sb_SnapToGrid: TSpeedButton; b_Bevel: TSpeedButton;
sb_Print: TSpeedButton; TabSheet1: TTabSheet;
TabSheet2: TTabSheet; Memo1: TMemo;
P_Font: TPanel; ColorDialog1: TColorDialog;
N3: TMenuItem; N_DeleteObject: TMenuItem;
P_HRuler: TPanel; P_Main: TPanel;
ScrollBox_: TScrollBox; ShapeSize: TShape;
P_VRuler: TPanel; TabSheet3: TTabSheet;
ImageList1: TImageList; HRuler: tmyRuler;
VRuler: tmyRuler; glBevel4: tmyBevel;
sb_FixAllMoving: TSpeedButton; sb_FixMoving: TSpeedButton;
glLabel3: TLabel; N_OLESize: TMenuItem;
N_Clip: TMenuItem; N_Center: TMenuItem;
N_Scale: TMenuItem; N_Stretch: TMenuItem;
N_AutoSize: TMenuItem; P_SBar: TPanel;
Panel5: TPanel; glLabel4: TLabel;
```

```
glLabel5: TLabel; glLabel7: TLabel;
se_Width: TSpinEdit; se_Top: TSpinEdit;
se_Left: TSpinEdit; glLabel6: TLabel;
se_Height: TSpinEdit; cb_Components: TComboBox;
glLabel8: TLabel; N4: TMenuItem;
Bevel5: TBevel; SB_Left: TSpeedButton;
SB_Bottom: TSpeedButton; SB_Right: TSpeedButton;
SB_Top: TSpeedButton; sb_AlignL: TSpeedButton;
sb_AlignC: TSpeedButton; sb_AlignR: TSpeedButton;
sb_AlignW: TSpeedButton; RxSpeedButton8: TSpeedButton;
RxSpeedButton9: TSpeedButton; sb_BevelBold: TSpeedButton;
glBevel2: tmyBevel; Panel3: TPanel;
RxSpinEdit1: TSpinEdit; Panel6: TPanel;
sbFontColor: TSpeedButton; glBevel3: tmyBevel;
Panel7: TPanel; Edit1: TMemo;
FE_OLE: TEdit; SpeedButton1: TSpeedButton;
OpenOLEFile: TOpenDialog; sb_FontUnderline: TSpeedButton;
sb_FontItalic: TSpeedButton; sb_FontBold: TSpeedButton;
TabSheet4: TTabSheet; glLabel1: TLabel;
lb_Params: tmyListBox; Panel4: TPanel;
SpeedButton2: TSpeedButton; sbBackColor: TSpeedButton;
SpeedButton3: TSpeedButton; CheckBox1: TCheckBox;
FontComboBox1: TComboBox; ColorComboBox1: TComboBox;
procedure OpenClick(Sender: TObject);
procedure Save1Click(Sender: TObject);
procedure ScrollBox_MouseDown(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
procedure FormCreate(Sender: TObject);
procedure Edit1Change(Sender: TObject);
procedure SB_LeftClick(Sender: TObject);
procedure FontComboBox1Change(Sender: TObject);
procedure RxSpinEdit1Change(Sender: TObject);
procedure ColorComboBox1Change(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure sb_BookClick(Sender: TObject);
procedure N1Click(Sender: TObject);
procedure sb_FontBoldClick(Sender: TObject);
procedure sb_AlignLClick(Sender: TObject);
procedure Memo1Change(Sender: TObject);
procedure sbFontColorClick(Sender: TObject);
procedure N_DeleteObjectClick(Sender: TObject);
procedure ScrollBox_MouseMove(Sender: TObject; Shift: TShiftState; X,
Y: Integer);
procedure ScrollBox_MouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
procedure FormDestroy(Sender: TObject);
procedure FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
procedure sb_FixMovingClick(Sender: TObject);
procedure N_AutoSizeClick(Sender: TObject);
procedure sb_SnapToGridClick(Sender: TObject);
procedure se_SizeChange(Sender: TObject);
procedure cb_ComponentsChange(Sender: TObject);
procedure N4Click(Sender: TObject);
procedure sb_BevelBoldClick(Sender: TObject);
procedure se_TopChange(Sender: TObject);
procedure se_WidthChange(Sender: TObject);
procedure se_HeightChange(Sender: TObject);
procedure FormClose(Sender: TObject; var Action: TCloseAction);
procedure FE_OLEChange(Sender: TObject);
procedure SpeedButton1Click(Sender: TObject);
procedure sb_PrintClick(Sender: TObject);
procedure sb_PreviewClick(Sender: TObject);
procedure SpeedButton2Click(Sender: TObject);
procedure cb_ComponentsKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
procedure CheckBox1Click(Sender: TObject);
private
ScrollBox: TlaScrollBox;
```

```
SelectedControl: tmyReportItem;
fSelection: boolean;
SelectionRect: TRect;
procedure RemakeComponentsList;
procedure read( FileName: string; ParentWnd: TWinControl );
procedure Save( FileName: string );
procedure OnMouseDown_(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
procedure OnMouseUp_(Sender: TObject; Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
procedure OnMouseMove_(Sender: TObject; Shift: TShiftState; X, Y: Integer);
procedure OnResize_(Sender: TObject);
procedure ShowComponentPos(Control: TControl);
procedure AssignEventsToAllComponents;
procedure UpdateToolBar( Control: tmyReportItem);
public
Component: tmyReport;
procedure Preview(glReport: tmyReport);
procedure Edit(glReport: tmyReport);
end;

TPublicControl = class(TControl)
public
property Caption;
end;

TPublicControlClass = class of TPublicControl;

const
IGNORE_VALUE = 65536;
var
glRepEditor: tmyRepEditor;
Form2: TComponent;

implementation
uses glTypes, glUtils;
{$R *.DFM}

procedure tmyRepEditor.OnMouseMove_(Sender: TObject; Shift: TShiftState; X, Y:
Integer);
var
DC: HDC;
i: integer;
begin
if fSelection then ScrollBox_MouseMove(Sender, Shift, X, Y);
if sb_FixAllMoving.Down then exit;
if TControl(Sender).Tag = 0 then exit;
if not fMouseDown then exit;
with ScrollBox do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyReportItem) then with tmyReportItem(Controls[i]) do
if Selected and not bool(Fixed) then
begin
Left:= ((Left + X - ControlPos.x)div Step.X)*Step.X;
Top:= ((Top + Y - ControlPos.y)div Step.Y)*Step.Y;
end;
fSkipSizeUpdate:= true;
ShowComponentPos(SelectedControl);
TControl(Sender).Left:= TControl(Sender).Left + X - ControlPos.x;
TControl(Sender).Top:= TControl(Sender).Top + Y - ControlPos.y;
{
TControl(Sender).Tag:= 2;//...переміщення
DC:= GetDC( TControl(Sender).Parent.Handle );
DrawFocusRect( DC, FocusRect );
FocusRect:= Bounds( TControl(Sender).Left+X-ControlPos.x,
TControl(Sender).Top+Y-ControlPos.y,  SelectedControl.Width,
SelectedControl.Height );
DrawFocusRect( DC, FocusRect );
ReleaseDC( TControl(Sender).Parent.Handle, DC );
```

```
}
end;

procedure tmyRepEditor.OnResize_(Sender: TObject);
begin
fSkipSizeUpdate:= true;
if Sender = SelectedControl then ShowComponentPos(TControl(Sender));
end;

procedure tmyRepEditor.read( FileName: string; ParentWnd: TWinControl );
begin
ScrollBox.HorzScrollBar.Position:= 0;
ScrollBox.VertScrollBar.Position:= 0;
SelectedControl:= nil;
UpdateToolBar( nil );
Component.LoadFromFile( FileName );
Component.CreateReport( ParentWnd, true );
AssignEventsToAllComponents;
RemakeComponentsList;
end;

procedure tmyRepEditor.Save( FileName: string );
begin
ScrollBox.HorzScrollBar.Position:= 0;
ScrollBox.VertScrollBar.Position:= 0;
Component.SaveToFile( FileName );
end;

procedure tmyRepEditor.OpenClick(Sender: TObject);
begin
OpenDialog1.InitialDir:= ExtractFilePath(ParamStr(0));
if OpenDialog1.Execute then
Read( OpenDialog1.FileName, ScrollBox );
end;

procedure tmyRepEditor.Save1Click(Sender: TObject);
begin
SaveDialog1.InitialDir:= ExtractFilePath(ParamStr(0));
if SaveDialog1.Execute then Save( SaveDialog1.FileName );
end;

procedure tmyRepEditor.ScrollBox_MouseDown(Sender: TObject; Button:
TMouseButton;
Shift: TShiftState; X, Y: Integer);
var
l, Compon: tmyReportItem;
R: TRect;
pt: TPoint;
begin
if ssCtrl in Shift then
begin
SelectionRect:=  Rect( 0,0,0,0 );
SelPt.X:= X - ScrollBox.HorzScrollBar.Position;
SelPt.Y:= Y - ScrollBox.VertScrollBar.Position;
SelPt:= ScrollBox.ClientToScreen(SelPt);
fSelection:= true;
end;
if (B_Label.Down)or(B_Bevel.Down)or(sb_OLE.Down) then
begin
Compon:= Component.AddComponent;
with Compon do
begin
Left:= X - ScrollBox.HorzScrollBar.Position;
Top:= Y - ScrollBox.VertScrollBar.Position;
if B_Label.Down then
begin
SideLeft:= 0;
SideTop := 0;
SideRight:= 0;
```

```
SideBottom:= 0;
end;
OnMouseDown:= OnMouseDown_;
OnMouseUp:= OnMouseUp_;
OnMouseMove:= OnMouseMove_;
OnResize:= OnResize_;
ContainOLE:= sb_OLE.Down;
B_Label.Down:= false;
B_Bevel.Down:= false;
sb_OLE.Down:= false;
RemakeComponentsList;
end;
end else
begin
R:= ScrollBox.ClientRect;
pt.x:= 0; pt.y:= 0; pt:= ScrollBox.ClientToScreen(pt);
OffsetRect( R, pt.x, pt.y );
ClipCursor( @R );
end;
end;

procedure tmyRepEditor.ScrollBox_MouseMove(Sender: TObject; Shift: TShiftState;
X, Y: Integer);
var
DC: HDC;
pt: TPoint;
begin

if not fSelection then exit;
DC:= GetDC( 0 );
DrawFocusRect( DC, SelectionRect );
Pt.X:= X - ScrollBox.HorzScrollBar.Position;
Pt.Y:= Y - ScrollBox.VertScrollBar.Position;
pt:= ScrollBox.ClientToScreen(pt);
SelectionRect:= Bounds( min( SelPt.X, pt.X ), min( SelPt.Y, pt.Y ), abs(
SelPt.X-pt.X ), abs( SelPt.Y-pt.Y ) );
DrawFocusRect( DC, SelectionRect );
ReleaseDC( 0, DC );
end;

procedure tmyRepEditor.sb_BookClick(Sender: TObject);
begin
if TControl(Sender).Tag=1 then
f_PrintReport.CReport1.Orientation:=f_PrintReport.PrintWin1.Orientation else
f_PrintReport.CReport1.Orientation:= f_PrintReport.PrintWin2.Orientation;
if TControl(Sender).Tag=1 then Printer.Orientation:=poPortrait
      else Printer.Orientation:= poLandscape;
UpdatePageSize;
end;

procedure tmyRepEditor.N1Click(Sender: TObject);
begin
ScrollBox.RemoveControl( SelectedControl );
ScrollBox.InsertControl( SelectedControl );
end;

procedure tmyRepEditor.sb_FontBoldClick(Sender: TObject);
begin
if not Assigned(SelectedControl) then exit;
with SelectedControl do
case TControl(Sender).Tag of
1: FStyle:= FStyle xor 1;
2: FStyle:= FStyle xor 2;
3: FStyle:= FStyle xor 4;
end;
end;

procedure tmyRepEditor.sb_AlignLClick(Sender: TObject);
begin
```

```
if not Assigned(SelectedControl) then exit;
SelectedControl.Alignment:= TControl(Sender).Tag;
end;

procedure tmyRepEditor.sbFontColorClick(Sender: TObject);
var i: integer;
begin
if not Assigned(SelectedControl) then exit;
with ColorDialog1 do
begin
case TControl(Sender).Tag of
0: Color:= SelectedControl.FColor;
1: Color:= SelectedControl.BkColor;
else Color:= SelectedControl.BvColor;
end;

if Execute then
for i:=0 to ScrollBox.ControlCount-1 do
if ScrollBox.Controls[i] is tmyReportItem then with
tmyReportItem(ScrollBox.Controls[i]) do
if tmyReportItem(ScrollBox.Controls[i]).Selected then
case TControl(Sender).Tag of
0: tmyReportItem(ScrollBox.Controls[i]).FColor:= Color;
1: tmyReportItem(ScrollBox.Controls[i]).BkColor:= Color;
else tmyReportItem(ScrollBox.Controls[i]).BvColor:= Color;
end;
end;
end;

procedure tmyRepEditor.N_DeleteObjectClick(Sender: TObject);
begin
if Assigned(SelectedControl) then
begin
if Windows.MessageBox(0, 'Delete object?', 'Confirm', MB_OKCANCEL ) <> IDOK then
exit;

if SelectedControl.ContainOLE then
ScrollBox.RemoveControl( SelectedControl.OLEContainer );
ScrollBox.RemoveControl( SelectedControl );
SelectedControl.Free;
SelectedControl:= nil;
RemakeComponentsList;
end;
end;

procedure tmyRepEditor.OnDrawScrollBox(Sender: TObject);
begin
VRuler.Top:= ShapeSize.Top;
HRuler.Left:= ShapeSize.Left + P_VRuler.Width;
end;

procedure tmyRepEditor.RemakeComponentsList;
var i: integer;
begin
cb_Components.Items.Clear;
for i:= 0 to ScrollBox.ControlCount-1 do
if ScrollBox.Controls[i] is tmyReportItem then
cb_Components.Items.Add( tmyReportItem(ScrollBox.Controls[i]).CompName );
cb_Components.Text:= '';
lb_Params.Items.Clear;
for i:=0 to Component.ParamNames.Count-1 do
lb_Params.Items.Add( Component.ParamNames[i] );
end;

procedure tmyRepEditor.UpdatePageSize;
const
Sizes:array[boolean,1..2] of integer = ((21,29),(29,21));
begin
```

```
ShapeSize.Width:=round(Sizes[Printer.Orientation=
poLandscape][1]*GetDeviceCaps(Canvas.Handle,LOGPIXELSX)* 1.541*2.54/10);
ShapeSize.Height:=round( Sizes[Printer.Orientation=poLandscape][2]
*GetDeviceCaps(Canvas.Handle,LOGPIXELSY)*1.541*2.54/10);
HRuler.Width:=ShapeSize.Width+10;
VRuler.Height:=ShapeSize.Height+10;
end;

procedure tmyRepEditor.FormKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
var
l, t, w, h: integer;
begin
w:= 0; h:= 0;
if (Shift = [ssCtrl])and(chr(Key)='Z')and fCanUndo then
begin
fCanUndo:=false;
ResizeReportControls(UndoPosShift.X, UndoPosShift.Y,0,0,true);
end;

if Assigned(ActiveControl) then exit;
l:=0; t:=0;
case Key of
VK_UP: if Shift = [ssShift] then h:= -1 else if Shift = [ssCtrl] then t:= -1;
VK_DOWN: if Shift = [ssShift] then h:= 1 else if Shift = [ssCtrl] then t:= 1;
VK_LEFT: if Shift = [ssShift] then w:= -1 else if Shift = [ssCtrl] then l:= -1;
VK_RIGHT: if Shift = [ssShift] then w:= 1 else if Shift = [ssCtrl] then l:= 1;
else exit;
end;
ResizeReportControls( l, t, w, h, true );

end;

procedure tmyRepEditor.sb_FixMovingClick(Sender: TObject);
var i: integer;
begin
with ScrollBox do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyReportItem)and tmyReportItem(Controls[i]).Selected then
tmyReportItem(Controls[i]).Fixed:= integer(sb_FixMoving.Down);
end;

procedure tmyRepEditor.N_AutoSizeClick(Sender: TObject);
begin
SelectedControl.OLESizeMode:= TMenuItem(Sender).Tag;
end;

procedure tmyRepEditor.sb_SnapToGridClick(Sender: TObject);
begin
if sb_SnapToGrid.Down then
begin Step.X:= Grid.X; Step.Y:= Grid.Y; end else
begin Step.X:= 1; Step.Y:= 1; end;
end;

procedure tmyRepEditor.se_SizeChange(Sender: TObject);
begin
if (not fMouseDown)and not fSkipSizeUpdate then
ResizeReportControls( se_Left.Value, IGNORE_VALUE,
  IGNORE_VALUE, IGNORE_VALUE,
  false{fUseParamsAsShifts} );
ShowComponentPos(SelectedControl);
fSkipSizeUpdate:= false;
end;

procedure tmyRepEditor.se_TopChange(Sender: TObject);
begin
if (not fMouseDown)and not fSkipSizeUpdate then
ResizeReportControls( IGNORE_VALUE, se_Top.Value,
  IGNORE_VALUE, IGNORE_VALUE,
```

```
  false{fUseParamsAsShifts} );
ShowComponentPos(SelectedControl);
fSkipSizeUpdate:= false;
end;

procedure tmyRepEditor.se_WidthChange(Sender: TObject);
begin
if (not fMouseDown)and not fSkipSizeUpdate then
ResizeReportControls( IGNORE_VALUE, IGNORE_VALUE,
  se_Width.Value, IGNORE_VALUE,
  false{fUseParamsAsShifts} );
ShowComponentPos(SelectedControl);
fSkipSizeUpdate:= false;
end;

procedure tmyRepEditor.se_HeightChange(Sender: TObject);
begin
if (not fMouseDown)and not fSkipSizeUpdate then
ResizeReportControls( IGNORE_VALUE, IGNORE_VALUE,
  IGNORE_VALUE, se_Height.Value,
  false{fUseParamsAsShifts} );
ShowComponentPos(SelectedControl);
fSkipSizeUpdate:= false;
end;

procedure tmyRepEditor.ResizeReportControls( l, t, w, h: integer;
fUseParamsAsShifts: boolean );
var i: integer;
begin
with ScrollBox do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyReportItem) then with tmyReportItem(Controls[i]) do
if Selected and not bool(Fixed) then
if fUseParamsAsShifts then
begin
if l < IGNORE_VALUE then Left:= Left + l; if t < IGNORE_VALUE then Top:= Top +
t;
if w < IGNORE_VALUE then Width:= Width + w; if h < IGNORE_VALUE then Height:=
Height + h;
end else
begin
if l < IGNORE_VALUE then Left:= l; if t < IGNORE_VALUE then Top:= t;
if w < IGNORE_VALUE then Width:= w; if h < IGNORE_VALUE then Height:= h;
end;
end;

procedure tmyRepEditor.cb_ComponentsChange(Sender: TObject);
var i: integer;
begin
if Assigned(SelectedControl) then
if SelectedControl.CompName = cb_Components.Text then exit;
with ScrollBox do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyReportItem) then
if tmyReportItem(Controls[i]).CompName = cb_Components.Text then
begin
OnMouseDown_( Controls[i], mbLeft, [], 0, 0 );
OnMouseUp_( Controls[i], mbLeft, [], 0, 0 );
exit;
end;
end;

procedure tmyRepEditor.ShowComponentPos(Control: TControl);
begin
if Component = nil then exit;
se_Left.Value:= Control.Left;
se_Top.Value:= Control.Top;
se_Width.Value:= Control.Width;
se_Height.Value:= Control.Height;
```

```
end;

procedure tmyRepEditor.AssignEventsToAllComponents;
var i: integer;
begin
with Component.ParentWnd do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyReportItem) then with tmyReportItem(Controls[i]) do
begin
OnMouseDown:= OnMouseDown_;
OnMouseUp:= OnMouseUp_;
OnMouseMove:= OnMouseMove_;
OnResize:= OnResize_;
end;
end;

function CanAlignControl(Control: TControl): boolean;
begin
Result:= (Control is tmyReportItem)and(bool(tmyReportItem(Control).Selected))
and(not bool(tmyReportItem(Control).Fixed));
end;

procedure tmyRepEditor.N4Click(Sender: TObject);
begin
if not Assigned(AlignForm) then AlignForm:= TAlignForm.Create(nil);
if AlignForm.ShowModal = mrOK then
AlignControlsInWindow( Component.ParentWnd, CanAlignControl, AlignForm.Horz,
AlignForm.Vert );
end;

procedure tmyRepEditor.sb_BevelBoldClick(Sender: TObject);
var i: integer;
begin
with Component.ParentWnd do
for i:=0 to ControlCount-1 do
if (Controls[i] is tmyReportItem)and bool(tmyReportItem(Controls[i]).Selected)
then
tmyReportItem(Controls[i]).PenWidth:= 1+integer(sb_BevelBold.Down);

end;

procedure tmyRepEditor.UpdateToolBar( Control: tmyReportItem);
begin
with Control do
begin
{
se_Left.Enabled:= Assigned(Control);
se_Top.Enabled:= Assigned(Control);
se_Width.Enabled:= Assigned(Control);
se_Height.Enabled:= Assigned(Control);}
P_Sides.Enabled:= Assigned(Control);
P_Font.Enabled:= Assigned(Control);
P_SBar.Enabled:= Assigned(Control);
if not Assigned(Control) then exit;
Edit1.Text:= Text;
Memo1.Text:= Text;
FontComboBox1.Text:= FName;
RxSpinEdit1.Value:= FSize;
ColorComboBox1.Color:= FColor;

SB_Left.Down := SideLeft = 1;
SB_Top.Down  := SideTop = 1;
SB_Right.Down:= SideRight = 1;
SB_Bottom.Down:= SideBottom = 1;
case Alignment of
1: sb_AlignL.Down:= true;
2: sb_AlignR.Down:= true;
3: sb_AlignC.Down:= true;
4: sb_AlignW.Down:= true;
```

```
end;
sb_FixMoving.Down:= bool(Fixed);
sb_FontBold.Down:= boolean(FStyle and 1);
sb_FontItalic.Down:= boolean(FStyle and 2);
sb_FontUnderline.Down:= boolean(FStyle and 4);
FE_OLE.Text:= OLELinkToFile;
FE_OLE.Enabled:= ContainOLE;
sb_BevelBold.Down:= bool(PenWidth-1);
fSkipSizeUpdate:= true;
ShowComponentPos( Control );

cb_Components.Text:= CompName;

end;
end;

procedure tmyRepEditor.FormClose(Sender: TObject; var Action: TCloseAction);
var  msS, msT: TMemoryStream;
begin
if Assigned(AlignForm) then AlignForm.Free;
if Assigned(ReportParamEditor) then ReportParamEditor.Free;
ScrollBox.HorzScrollBar.Position:= 0;
ScrollBox.VertScrollBar.Position:= 0;
Component.Save;
end;

procedure tmyRepEditor.FE_OLEChange(Sender: TObject);
var
str: string;
begin
if (not Assigned(SelectedControl))or (not FileExists(FE_OLE.Text)) then exit;
str:= FE_OLE.Text;
if ExtractFilePath(Name) = ExtractFilePath(ParamStr(0)) then
str:= ExtractFileName(Name);
if SelectedControl.OLELinkToFile <> str then
SelectedControl.OLELinkToFile:= str;
end;

procedure tmyRepEditor.SpeedButton1Click(Sender: TObject);
begin
if OpenOLEFile.Execute then FE_OLE.Text:= OpenOLEFile.FileName;
end;

procedure tmyRepEditor.sb_PrintClick(Sender: TObject);
begin
if Assigned(Component)and (Component.ComponentList.Count > 0) then
Component.Print;
Component.OwnerWnd:= self;
Component.ParentWnd:= ScrollBox;
end;

procedure tmyRepEditor.sb_PreviewClick(Sender: TObject);
var
Form: TForm;
Image: TImage;
bmp: TBitmap;
R: TRect;
i, W, H: integer;
begin
if not Assigned(Component) then exit;
Form:= TForm.Create(nil);
Form.Caption:= 'Page Preview';
Image:= TImage.Create(Form);
bmp:= TBitmap.Create;
Image.Parent:= Form;
H:= SantimsToPixels( Form.Canvas.Handle, 29, true );
W:= SantimsToPixels( Form.Canvas.Handle, 21, false );
//  Image.Width:= W+8;
Image.Left:= 0;  Image.Top:= 0;
```

```
bmp.Width:= W+7;
bmp.Height:= H+7;
try


with Component do
for i:=0 to ComponentList.Count-1 do with tmyReportItem(ComponentList[i]) do
begin
PaintTo(bmp.Canvas);
if ContainOle then OLEContainer.PaintTo( bmp.Canvas.Handle, Left, Top );
end;

bmp.Canvas.Brush.Color:= clBtnFace;
R:= Bounds(bmp.Width-7, 0, 7, bmp.Height-7);
bmp.Canvas.FillRect( R );
R:= Bounds(0, bmp.Height-7, bmp.Width-7, 7);
bmp.Canvas.FillRect( R );
bmp.Canvas.Brush.Color:= 0;
R:= Bounds(bmp.Width-7, 7, 7, bmp.Height-7);
bmp.Canvas.FillRect( R );
R:= Bounds(7, bmp.Height-7, bmp.Width-7, 7);
bmp.Canvas.FillRect( R );

Image.Picture.bitmap:= bmp;
Image.Stretch:= true;
Image.Width:= W div 2;
Image.Height:= H div 2;
Form.ClientWidth:= Image.Width;
Form.ClientHeight:= Image.Height;

bmp.Free; bmp:= nil;
Form.ShowModal;
finally
if Assigned(bmp) then bmp.Free;
Form.Free;
end;
end;

procedure tmyRepEditor.SpeedButton2Click(Sender: TObject);
begin
if not Assigned(ReportParamEditor) then
ReportParamEditor:= TReportParamEditor.Create(nil);
ReportParamEditor.ShowModal;
end;

procedure tmyRepEditor.cb_ComponentsKeyDown(Sender: TObject; var Key: Word;
Shift: TShiftState);
begin
if (Key=VK_RETURN) then
begin
SelectedControl.CompName:= trim(cb_Components.Text);
RemakeComponentsList;
cb_Components.Text:= SelectedControl.CompName;
end;
if (Key=VK_ESCAPE) then cb_Components.Text:= SelectedControl.CompName;

end;

procedure tmyRepEditor.CheckBox1Click(Sender: TObject);
begin
if Assigned(SelectedControl) then SelectedControl.Transparent:=
integer(TCheckBox(Sender).Checked);
end;

end.
```

Файл форми виведення графічного представлення (fset.pas)

```
unit SET;

interface

uses Windows, Messages, Classes, Forms, dialogs;

///Розробив Федюк Ярослав Вячеславович, ЦНТУ, 2023

type

TEMUL_MULTICORE = class ( TComponent )
private
FResult: boolean;
FFileName: string;
FOnTermainated: TNotifyEvent;
si: TStartupInfo;
public
pi: TProcessInformation;
function Run: boolean;
function Kill: boolean;
destructor Destroy; override;
published
property FileName: string read FFileName write FFileName;
property Result: boolean read FResult stored false;
property OnTermainated: TNotifyEvent read FOnTermainated write FOnTermainated;
end;

procedure Register;

implementation

procedure Register;
begin
RegisterComponents('Proba', [TEMUL_MULTICORE]);
end;


destructor TEMUL_MULTICORE.Destroy;
begin
Kill;
inherited;
end;

function TEMUL_MULTICORE.Run: boolean;
begin
GetStartupInfo(si);
si.wShowWindow:= SW_NORMAL;
FResult:= CreateProcess( PChar(FFileName), nil, nil, nil, false,
NORMAL_PRIORITY_CLASS, nil, nil, si, pi);
Run:= FResult;
if Result then
begin
while WaitForSinObjectMultiCore(pi.hProcess, 100) = WAIT_TIMEOUT do
Application.ProcessMessages;
if Assigned(OnTermainated) then OnTermainated(self);
end;
end;

function TEMUL_MULTICORE.Kill: boolean;
begin
if FResult {and(WaitForSingleObject(pi.hProcess, 100) <> WAIT_TIMEOUT)}
then Kill:= TerminateProcess( pi.hProcess, 0 ) else Kill:= false;
end;
```

```
end.
```

Файл форми даних розробленої програми
(data.pas)

```
unit DATA;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
dsgnintf,
  StdCtrls, glPropCn, ComCtrls, ExtCtrls, TypInfo, Buttons;

type

  Ttdata_MultiCore1 = class( TPropertyEditor )
    function GetAttributes: TPropertyAttributes; override;
    function GetValue: string; override;
    procedure Edit; override;
  end;

  Ttdata_MultiCore2 = class(TComponentEditor)
    procedure ExecuteVerb(Index: Integer); override;
    function GetVerb(Index: Integer): string; override;
    function GetVerbCount: Integer; override;
  end;


  Ttdata_MultiCore3 = class(TForm)
    lvAll: TListView;
    Label1: TLabel;
    Bevel1: TBevel;
    Bevel2: TBevel;
    Label2: TLabel;
    Bevel3: TBevel;
    Bevel4: TBevel;
    lvSel: TListView;
    pbAdd: TBitBtn;
    pbRemove: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    ImageList1: TImageList;
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure lvAllDblClick(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure lvSelDblClick(Sender: TObject);
    procedure lvSelChange(Sender: TObject; Item: TListItem;
      Change: TItemChange);
    procedure lvAllChange(Sender: TObject; Item: TListItem;
      Change: TItemChange);
  private
    ComponentList: TStringList;
  end;

type
  TDesigner = IDesigner;
  TFormDesigner = IFormDesigner;

var
  dataCompListEditor: Ttdata_MultiCore3;

implementation
{$R *.DFM}
```

```
procedure ShowCompListEditor(Designer: TDesigner; dataPropertyCenter:
TdataPropertyCenter);
var
  Dialog:Ttdata_MultiCore3;
  I:Integer;
begin
  Dialog:=Ttdata_MultiCore3.Create( Application );
  Dialog.Component:=dataPropertyCenter;
  Dialog.ShowModal;
  Dialog.free;
end;

function Ttdata_MultiCore1.GetAttributes: TPropertyAttributes;
begin
  Result:=[paDialog];
end;

function Ttdata_MultiCore1.GetValue: string;
begin
  Result:= Format( '(%s)', [ GetPropType^.Name ] );
end;

procedure Ttdata_MultiCore1.Edit;
begin
  ShowCompListEditor(Designer, dataPropertyCenter(GetComponent(0)));
  GetComponent(0).Owner.Name
end;

procedure Ttdata_MultiCore2.ExecuteVerb(Index: Integer);
begin
  case Index of
    0: ShowCompListEditor(Designer, dataPropertyCenter(Component));
  end;
end;

function Ttdata_MultiCore2.GetVerbCount: Integer;
begin
  Result:= 1;
end;

procedure Ttdata_MultiCore3.FormCreate(Sender: TObject);
begin
  ControlsList:= TList.create;
end;

procedure Ttdata_MultiCore3.FormDestroy(Sender: TObject);
begin
  ControlsList.Free;
end;

procedure Ttdata_MultiCore3.FormShow(Sender: TObject);
var
  i, j: integer;
  ListItem: TListItem;
  Comp: TComponent;
  ColorPropInfo, FontPropInfo: PPropInfo;
const
  aSigns: array [boolean] of string = ('-', '+');
begin
  lvAll.Items.Clear;
  lvSel.Items.Clear;

  for i:= 0 to Component.ComponentList.Count-1 do
  begin
    ListItem:= lvSel.Items.Add;
    ListItem.Caption:= Component.ComponentList[i];
    Comp:= Component.Owner.FindComponent( Component.ComponentList[i] );
    if Comp = nil then continue;
    ColorPropInfo:= GetPropInfo( Comp.ClassInfo, 'Color');
```

```
    FontPropInfo:= GetPropInfo( Comp.ClassInfo, 'Font');
    ListItem.SubItems.Add(aSigns[Assigned(ColorPropInfo)]);
    ListItem.SubItems.Add(aSigns[Assigned(FontPropInfo)]);
    ListItem.ImageIndex:= 1;
  end;

  with Component.Owner do
  for i:=0 to ComponentCount-1 do
  begin
    ColorPropInfo:= GetPropInfo( Components[i].ClassInfo, 'Color');
    FontPropInfo:= GetPropInfo( Components[i].ClassInfo, 'Font');
    if (ColorPropInfo <> nil)or(FontPropInfo <> nil) then
     begin
       ListItem:= lvAll.Items.Add;
       ListItem.Caption:= Components[i].Name;
       ListItem.SubItems.Add(aSigns[Assigned(ColorPropInfo)]);
       ListItem.SubItems.Add(aSigns[Assigned(FontPropInfo)]);

       for j:=0 to lvSel.Items.Count - 1 do
         if lvSel.Items[j].Caption = ListItem.Caption then
         begin ListItem.ImageIndex:= 1; break; end;

     end;
     SetOrdProp( FormX.Components[i], PropInfo, clGreen );
  end;
end;

procedure Ttdata_MultiCore3.lvAllDblClick(Sender: TObject);
var ListItem: TListItem;
begin
  if (lvAll.Selected = nil)or(lvAll.Selected.ImageIndex = 1) then exit;

  ListItem:= lvSel.Items.Add;
  ListItem.Caption:= lvAll.Selected.Caption;
  ListItem.SubItems.Add(lvAll.Selected.SubItems[0]);
  ListItem.SubItems.Add(lvAll.Selected.SubItems[1]);
  lvAll.Selected.ImageIndex:= 1;
  ListItem.ImageIndex:= 1;
end;

procedure Ttdata_MultiCore3.BitBtn4Click(Sender: TObject);
begin close; end;

procedure Ttdata_MultiCore3.BitBtn3Click(Sender: TObject);
var
  i:integer;
  Comp:TComponent;
begin
  Component.ComponentList.Clear;
  Component.CompList.Clear;
  for i:=0 to lvSel.Items.Count-1 do
  begin
  Comp:=Component.Owner.FindComponent(lvSel.Items[i].Caption);
    if Comp = nil then continue;
    Component.CompList.Add(Comp);
    Component.ComponentList.Add(lvSel.Items[i].Caption);
  end;
  close;
end;

procedure Ttdata_MultiCore3.lvSelDblClick(Sender: TObject);
var i: integer;
begin
  if not Assigned(lvSel.Selected) then exit;
  for i:=0 to lvAll.Items.Count - 1 do
  if lvAll.Items[i].Caption = lvSel.Selected.Caption then break;
  lvAll.Items[i].ImageIndex:= 0;
  lvSel.Items.Delete(lvSel.Selected.Index);
end;
```

```
procedure Ttdata_MultiCore3.lvSelChange(Sender:
TObject;Item:TListItem;Change:TItemChange);
begin
  pbRemove.Enabled:=Assigned(lvSel.Selected);
end;

procedure Ttdata_MultiCore3.lvAllChange(Sender: TObject; Item: TListItem;
Change: TItemChange);
begin
pbAdd.Enabled:=Assigned(lvAll.Selected)and(lvAll.Selected.ImageIndex=0);
end;

end.
```

Файл форми роботи программної моделі
(MultiCoreEmulation.pas)

```
unit MultiCoreEmulation;
///Розробив Федюк Ярослав Вячеславович, ЦНТУ, 2023
interface
uses Windows,Graphics,Controls,Classes,ExtCtrls,glTypes,SysUtils;

type
TTwainColors = class; Tem_MultiCore_CustomLabelColors = class;
Tem_MultiCore_LabelColors = class; TCustomGradient = class;
TGradient = class; TThreeDGradient = class;
T2DAlign = class; Tem_MultiCore_CustomBoxStyle= class;
Tem_MultiCore_CustomTextBoxStyle= class;
Tem_MultiCore_TextBoxStyle = class;
TPointClass = class; Tem_MultiCore_Bevel = class;
Tem_MultiCore_ExtBevel = class; TIllumination = class;
Tem_MultiCore_LabelTextStyles = class;
Tem_MultiCore_CustomTextColors = class;
Tem_MultiCore_SimleLabelColors = class;
Tem_MultiCore_BevelLines = class; TTwainColors = class(TPersistent)
Tem_MultiCore_GrBoxColors = class; Tem_MultiCore_ListBoxItemStyle  = class;
Tem_MultiCore_AskListBoxItemStyle= class;
private
  FFromColor:TColor; FToColor:TColor;
  procedure SeTem_MultiCore_omColor( Value:TColor );
  procedure SetToColor( Value:TColor );
public
  FRGBFromColor:Longint;
  FRGBToColor:Longint;
  OnChanged:TNotifyEvent;
  constructor Create; virtual;
published
  property FromColor:TColor read FFromColor write SeTem_MultiCore_omColor
default $00808080;
   property ToColor:TColor read FToColor write SetToColor
default 0;
end;

TCustomGradient = class(TTwainColors)
private
  FBufferedDraw: boolean;
  FSteps: integer;
  FPercentFilling: Tem_MultiCore_Percent;
  FBrushStyle: TBrushStyle;
procedure SetActive( Value:boolean );
procedure SetOrientation( Value:Tem_MultiCore_GradientDir );
procedure SetSteps( Value: integer );
procedure SetPercentFilling( Value: Tem_MultiCore_Percent );
procedure SetBrushStyle( Value: TBrushStyle );
public
FOrientation: Tem_MultiCore_GradientDir;
FActive: boolean;
fReverse: boolean;
procedure TextOut(DC:HDC;Str:string; TextR:TRect; x, y:integer);
function GetColorFromGradientLine( GradientLineWidth, Position: word ):
COLORREF;
constructor Create; override;
destructor Destroy;override;
protected
property Active:boolean read FActive write SetActive;
property BufferedDraw:boolean read FBufferedDraw write FBufferedDraw
default false;
property Orientation:Tem_MultiCore_GradientDir read FOrientation write
SetOrientation;
property Steps: integer read FSteps write SetSteps default 255;
```

```
property PercentFilling: Tem_MultiCore_Percent read FPercentFilling write
SetPercentFilling
default 100;
property BrushStyle: TBrushStyle read FBrushStyle write SetBrushStyle
default bsSolid;
end;
TGradient = class(TCustomGradient)
private
public
procedure Draw(DC: HDC; r: TRect; PenStyle, PenWidth: integer);
published
property Active;
property BufferedDraw;
property Orientation;
property Steps;
property PercentFilling;
property BrushStyle;
end;
TThreeDGradient = class(TCustomGradient)
private
FDepth:word;
FGType:TThreeDGradientType;
procedure SetDepth(Value: word);
procedure SetGType(Value: TThreeDGradientType);
public
constructor Create; override;
published
property Depth: word
read FDepth write SetDepth default 16;
property GType:TThreeDGradientType read FGType write SetGType default fgtFlat;
end;
T2DAlign = class(TPersistent)
private
FHorizontal: Tem_MultiCore_HorAlign;
FVertical: Tem_MultiCore_VertAlign;
procedure SetHorizontal(Value: Tem_MultiCore_HorAlign);
procedure SetVertical(Value: Tem_MultiCore_VertAlign);
public
OnChanged:TNotifyEvent;
constructor Create;
published
property Horizontal:Tem_MultiCore_HorAlign read FHorizontal write SetHorizontal
default fhaLeft;
property Vertical:Tem_MultiCore_VertAlign read FVertical write SetVertical
default fvaTop;
end;
TPointClass = class(TPersistent)
private
FX: integer;
FY: integer;
procedure SetX(Value: integer);
procedure SetY(Value: integer);
public
OnChanged:TNotifyEvent;
published
property X:integer read FX write SetX;
property Y:integer read FY write SetY;
end;
Tem_MultiCore_Bevel = class(TPersistent)
private
FInner: TPanelBevel;
FOuter: TPanelBevel;
FSides: Tem_MultiCore_Sides;
FBold: boolean;
procedure SetInner(Value: TPanelBevel);
procedure SetOuter(Value: TPanelBevel);
procedure SetSides(Value: Tem_MultiCore_Sides);
procedure SetBold(Value: boolean);
public
```

```
OnChanged:TNotifyEvent;
constructor Create; virtual;
function BordersHeight: integer;
function BordersWidth: integer;
published
property Inner: TPanelBevel read FInner write SetInner stored true;//    default
bvLowered;
property Outer: TPanelBevel read FOuter write SetOuter stored true;//    default
bvNone;
property Sides: Tem_MultiCore_Sides read FSides write SetSides stored true
default ALLGLSIDES;
property Bold: boolean read FBold write SetBold stored true;//  default false;
end;
Tem_MultiCore_ExtBevel = class(Tem_MultiCore_Bevel)
private
FActive: boolean;
FBevelPenStyle: TPenStyle;
FBevelPenWidth: word;
FInteriorOffset: word;
procedure SetActive(Value: boolean);
procedure SetBevelPenStyle(Value: TPenStyle);
procedure SetBevelPenWidth(Value: word);
procedure SetInteriorOffset(Value: word);
public
constructor Create; override;
published
property Active: boolean read FActive write SetActive
default true;
property BevelPenStyle: TPenStyle read FBevelPenStyle write SetBevelPenStyle
default psSolid;
property BevelPenWidth: word read FBevelPenWidth write SetBevelPenWidth
default 1;
property InteriorOffset: word read FInteriorOffset write SetInteriorOffset
default 0;
end;
TIllumination = class(T2DAlign)
private
procedure SetShadowDepth(Value: integer);
public
FShadowDepth: integer;
OnChanged:TNotifyEvent;
constructor Create;
published
property ShadowDepth: integer
read FShadowDepth write SetShadowDepth default 2;
end;
Tem_MultiCore_LabelTextStyles = class(TPersistent)
private
FPassive: Tem_MultiCore_TextStyle;
FActive: Tem_MultiCore_TextStyle;
FDisabled: Tem_MultiCore_TextStyle;
procedure SetPassive(Value: Tem_MultiCore_TextStyle);
procedure SetActive(Value: Tem_MultiCore_TextStyle);
procedure SetDisabled(Value: Tem_MultiCore_TextStyle);
public
OnChanged:TNotifyEvent;
constructor Create;
published
property Passive: Tem_MultiCore_TextStyle read FPassive write SetPassive
default fstRaised;
property Active: Tem_MultiCore_TextStyle read FActive write SetActive
default fstRaised;
property Disabled: Tem_MultiCore_TextStyle read FDisabled write SetDisabled
default fstPushed;
end;
Tem_MultiCore_CustomTextColors = class(TPersistent)
private
FText: TColor;
FTextDisabled: TColor;
```

```
FDelineate: TColor;
FBackground: TColor;
public
FHighlight: TColor;
FShadow: TColor;
private
procedure SetText(Value: TColor);
procedure SetTextDisabled(Value: TColor);
procedure SetDelineate(Value: TColor);
procedure SetBackground(Value: TColor);
procedure SetHighlight(Value: TColor);
procedure SetShadow(Value: TColor);
public
OnChanged:TNotifyEvent;
constructor Create; virtual;
protected
property Text: TColor read FText write SetText
default clBlack;
property TextDisabled: TColor read FTextDisabled write SetTextDisabled
default clGray;
property Delineate: TColor read FDelineate write SetDelineate
default clWhite;
property Shadow:TColor read FShadow write SetShadow default clBtnShadow;
property Highlight:TColor read FHighlight write SetHighlight default
clBtnHighlight;
property Background:TColor read FBackground write SetBackground default
clBtnFace;
end;
Tem_MultiCore_SimleLabelColors = class(Tem_MultiCore_CustomTextColors)
published
property Text stored true;
property Delineate stored true;
property Shadow stored true;
property Highlight;
property Background stored true;
end;
Tem_MultiCore_CustomLabelColors = class(Tem_MultiCore_CustomTextColors)
private
FTextActive: TColor;
FDelineateActive: TColor;
FAutoHighlight: boolean;
FAutoShadow: boolean;
FBackgroundActive: TColor;
public
FColorHighlightShift: integer;
FColorShadowShift: integer;
private
procedure SetTextActive(Value: TColor);
procedure SetDelineateActive(Value: TColor);
procedure SetBackgroundActive(Value: TColor);
procedure SetAutoHighlight(Value: boolean);
procedure SetAutoShadow(Value: boolean);
procedure SetColorHighlightShift(Value: integer);
procedure SetColorShadowShift(Value: integer);
public
OnChanged:TNotifyEvent;
constructor Create; override;
protected
property TextActive: TColor read FTextActive write SetTextActive
default clBlack;
property DelineateActive: TColor read FDelineateActive write SetDelineateActive
default clWhite;
property AutoHighlight: boolean
read FAutoHighlight write SetAutoHighlight default false;
property AutoShadow: boolean
read FAutoShadow write SetAutoShadow default false;
property ColorHighlightShift: integer
read FColorHighlightShift write SetColorHighlightShift default 40;
property ColorShadowShift: integer
```

```
read FColorShadowShift write SetColorShadowShift default 60;
property BackgroundActive: TColor
read FBackgroundActive write SetBackgroundActive default clBtnFace;
end;
Tem_MultiCore_LabelColors = class(Tem_MultiCore_CustomLabelColors)
published
property Text;
property TextDisabled;
property Delineate;
property Shadow;
property Highlight;
property Background;
property TextActive;
property DelineateActive;
property AutoHighlight;
property AutoShadow;
property ColorHighlightShift;
property ColorShadowShift;
property BackgroundActive;
end;
Tem_MultiCore_GrBoxColors = class(Tem_MultiCore_CustomLabelColors)
private
FCaption: TColor;
FCaptionActive: TColor;
FClient: TColor;
FClientActive: TColor;
procedure SetCaption(Value: TColor);
procedure SetCaptionActive(Value: TColor);
procedure SetClient(Value: TColor);
procedure SetClientActive(Value: TColor);
public
constructor Create; override;
published
property Text;
property Delineate;
property Shadow;
property Highlight;
property Background;
property TextActive;
property DelineateActive;
property AutoHighlight;
property AutoShadow;
property ColorHighlightShift;
property ColorShadowShift;
property BackgroundActive;
property Caption: TColor read FCaption write SetCaption;
property CaptionActive: TColor read FCaptionActive write SetCaptionActive;
property Client: TColor read FClient write SetClient;
property ClientActive: TColor read FClientActive write SetClientActive;
end;
Tem_MultiCore_CustomListBoxItemStyle  = class(TPersistent)
private
FColor: TColor;
FDelineateColor: TColor;
FFont: TFont;
FBevel: Tem_MultiCore_Bevel;
FTextStyle: Tem_MultiCore_TextStyle;
FOnChanged: TNotifyEvent;
procedure SetColor(Value: TColor);
procedure SetDelineateColor(Value: TColor);
procedure SetFont(Value: TFont);
procedure SetTextStyle(Value: Tem_MultiCore_TextStyle);
protected
procedure SetOnChanged(Value: TNotifyEvent); virtual;
public
property OnChanged: TNotifyEvent read FOnChanged write SetOnChanged;
constructor Create; virtual;
destructor Destroy; override;
function HighlightColor: TColor;
```

```
function ShadowColor: TColor;
published
property Color: TColor read FColor write SetColor;
property DelineateColor: TColor read FDelineateColor write SetDelineateColor;
property Font: TFont read FFont write SetFont;
property Bevel: Tem_MultiCore_Bevel read FBevel write FBevel;
property TextStyle: Tem_MultiCore_TextStyle read FTextStyle write SetTextStyle;
end;
Tem_MultiCore_ListBoxItemStyle=class(Tem_MultiCore_CustomListBoxItemStyle)
private
FGradient: TGradient;
FTextGradient: TGradient;
protected
property TextGradient: TGradient read FTextGradient write FTextGradient;
procedure SetOnChanged(Value: TNotifyEvent); override;
public
constructor Create; override;
destructor Destroy; override;
published
property Gradient: TGradient read FGradient write FGradient;
end;

Tem_MultiCore_HintStyle=class(Tem_MultiCore_ListBoxItemStyle)
end;

Tem_MultiCore_SpeedButtonStyle=class(Tem_MultiCore_ListBoxItemStyle)
published
property TextGradient;
end;
Tem_MultiCore_AskListBoxItemStyle= class(Tem_MultiCore_CustomListBoxItemStyle)
private
FBtnColor: TColor;
FBtnFont: TFont;
FBtnTextStyle: Tem_MultiCore_TextStyle;
procedure SetBtnColor(Value: TColor);
procedure SetBtnFont(Value: TFont);
procedure SetBtnTextStyle(Value: Tem_MultiCore_TextStyle);
public
constructor Create; override;
destructor Destroy; override;
published
property BtnColor: TColor read FBtnColor write SetBtnColor;
property BtnFont: TFont read FBtnFont write SetBtnFont;
property BtnTextStyle: Tem_MultiCore_TextStyle read FBtnTextStyle write
SetBtnTextStyle;
end;
Tem_MultiCore_CustomBoxStyle= class(Tem_MultiCore_Bevel)
private
FPenStyle: TPenStyle;
FHighlightColor: TColor;
FShadowColor: TColor;
procedure SetPenStyle(Value: TPenStyle);
procedure SetHighlightColor(Value: TColor);
procedure SetShadowColor(Value: TColor);
public
OnChanged: TNotifyEvent;
constructor Create; override;
protected
property PenStyle: TPenStyle read FPenStyle write SetPenStyle
default psSolid;
property HighlightColor: TColor read FHighlightColor write SetHighlightColor
default clBtnHighlight;
property ShadowColor: TColor read FShadowColor write SetShadowColor
default clBtnShadow;
end;
Tem_MultiCore_CustomTextBoxStyle= class(Tem_MultiCore_CustomBoxStyle)
private
FTextColor: TColor;
FBackgroundColor: TColor;
```

```
procedure SetTextColor(Value: TColor);
procedure SetBackgroundColor(Value: TColor);
public
constructor Create; override;
protected
property TextColor: TColor read FTextColor write SetTextColor
default clBlack;
property BackgroundColor: TColor read FBackgroundColor write SetBackgroundColor
default clWindow;
end;
Tem_MultiCore_TextBoxStyle = class(Tem_MultiCore_CustomTextBoxStyle)
published
property Inner;
property Outer;
property Sides;
property Bold;
property PenStyle;
property TextColor;
property BackgroundColor;
property HighlightColor;
property ShadowColor;
end;
Tem_MultiCore_BevelLines = class(TPersistent)
private
FCount: cardinal;
FStep: cardinal;
FOrigin: Tem_MultiCore_Origin;
FStyle: TPanelBevel;
FBold: boolean;
FThickness: byte;
FIgnoreBorder: boolean;

procedure SetCount(Value: cardinal);
procedure SetStep(Value: cardinal);
procedure SetOrigin(Value: Tem_MultiCore_Origin);
procedure SetStyle(Value: TPanelBevel);
procedure SetBold(Value: boolean);
procedure SetThickness(Value: byte);
procedure SetIgnoreBorder(Value: boolean);
public
OnChanged: TNotifyEvent;
constructor Create;
published
property Count: cardinal read FCount write SetCount
default 0;
property Step: cardinal read FStep write SetStep
default 0;
property Origin: Tem_MultiCore_Origin read FOrigin write SetOrigin
default forLeftTop;
property Style: TPanelBevel read FStyle write SetStyle
default bvLowered;
property Bold: boolean read FBold write SetBold
default false;
property Thickness: byte read FThickness write SetThickness
default 1;
property IgnoreBorder: boolean read FIgnoreBorder write SetIgnoreBorder
default false;
end;
implementation

{$I Debug.inc}

constructor TTwainColors.Create;
begin
inherited Create;
FFromColor:= $00808080; FRGBFromColor:= ColorToRGB( FFromColor );
FToColor:= 0;FRGBToColor:= ColorToRGB( FToColor );
end;
```

```
procedure TTwainColors.SeTem_MultiCore_omColor( Value:TColor );
begin
if FFromColor = Value then exit;
FFromColor:= Value; FRGBFromColor:= ColorToRGB( Value );
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TTwainColors.SetToColor( Value:TColor );
begin
if FToColor = Value then exit;
FToColor:= Value; FRGBToColor:= ColorToRGB( Value );
if Assigned(OnChanged) then OnChanged(self);
end;
constructor TCustomGradient.Create;
begin
inherited Create;
FActive:= false;
FBufferedDraw:= false;
FOrientation:= fgdHorizontal;
FSteps:= 255;
FPercentFilling:= 100;
FBrushStyle:= bsSolid;
end;

destructor TCustomGradient.Destroy;
begin inherited Destroy; end;

procedure TCustomGradient.Free;
begin if self<>nil then Destroy; end;

procedure TCustomGradient.SetActive( Value:boolean );
begin
if FActive = Value then exit;
FActive:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TCustomGradient.SetOrientation( Value:Tem_MultiCore_GradientDir );
begin
if FOrientation = Value then exit;
FOrientation:= Value;
if FActive and Assigned(OnChanged) then OnChanged(self);
end;

procedure TCustomGradient.SetSteps( Value: integer );
begin
if Value > 255 then Value:= 255
else if Value < 1 then Value:= 1;
if FSteps = Value then exit;
FSteps:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TCustomGradient.SetPercentFilling( Value: Tem_MultiCore_Percent );
begin
if FPercentFilling = Value then exit;
FPercentFilling:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TCustomGradient.SetBrushStyle( Value: TBrushStyle );
begin
FBrushStyle:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

function TCustomGradient.GetColorFromGradientLine
( GradientLineWidth, Position: word ): COLORREF;
var
```

```
c1F,c2F,c3F:byte;
c1T,c2T,c3T:byte;
Step1,Step2,Step3:SinEm_MultiCore;
begin
c1F:= Byte( self.FRGBFromColor ) ;
c2F:= Byte( WORD(self.FRGBFromColor) shr 8);
c3F:= Byte( self.FRGBFromColor shr 16 );
c1T:= Byte( self.FRGBToColor ) ;
c2T:= Byte( WORD(self.FRGBToColor) shr 8);
c3T:= Byte( self.FRGBToColor shr 16 );

Step1:=(c1T-c1F)/GradientLineWidth;
Step2:=(c2T-c2F)/GradientLineWidth;
Step3:=(c3T-c3F)/GradientLineWidth;

Result:=RGB( trunc(c1F+Step1*Position),
trunc(c2F+Step2*Position),
trunc(c3F+Step3*Position) );
end;


procedure TCustomGradient.TextOut( DC: HDC; Str:string; TextR:TRect; x,
y:integer );
var
i,Steps:integer;
r:TRect;
c1F,c2F,c3F:byte;
c1T,c2T,c3T:byte;
c1,c2,c3:SinEm_MultiCore;
OldTextColor:TCOLORREF;
begin
if (not Active)or(GetDeviceCaps( DC, BITSPIXEL )<16) then
begin Windows.TextOut( DC, x,y, PChar(str), Length(str)); exit; end;
r:= TextR;
c1F:= Byte( FRGBFromColor ) ;
c2F:= Byte( WORD(FRGBFromColor) shr 8);
c3F:= Byte( FRGBFromColor shr 16 );
c1T:= Byte( FRGBToColor ) ;
c2T:= Byte( WORD(FRGBToColor) shr 8);
c3T:= Byte( FRGBToColor shr 16 );

c1:=c1F; c2:=c2F; c3:=c3F;
if FOrientation = fgdVertical then Steps:=r.right-r.left
else Steps:=r.bottom-r.top;
Step1:=(c1T-c1F)/Steps; Step2:=(c2T-c2F)/Steps; Step3:=(c3T-c3F)/Steps;

OldTextColor:= SetTextColor( DC, 0 );
Steps:= MulDiv( Steps, PercentFilling, 100 );
for i:=0 to Steps do
begin
SetTextColor( DC, RGB( trunc(c1),trunc(c2),trunc(c3)) );

if FOrientation = fgdVertical then
begin r.left:=i; r.right:=r.left+1; end
else
begin r.top:=i; r.bottom:=r.top+1; end;

Windows.ExtTextOut( DC, x, y,  ETO_CLIPPED, @r,
PChar(str), Length(str), nil);
c1:= c1+ Step1; c2:= c2+ Step2; c3:= c3+ Step3;
end;
SetTextColor( DC, OldTextColor );
end;
constructor TThreeDGradient.Create;
begin
inherited Create;
Depth:=16;
FGType:=fgtFlat;
FActive:=true;
```

```
end;

procedure TThreeDGradient.SetGType(Value: TThreeDGradientType);
begin
if FGType = Value then exit;
FGType:= Value; if FActive and Assigned(OnChanged) then OnChanged(self);
end;

procedure TThreeDGradient.SetDepth(Value: word);
begin
if FDepth = Value then exit;
FDepth:= Value; if FActive and Assigned(OnChanged) then OnChanged(self);
end;

constructor T2DAlign.Create;
begin
inherited Create;
FHorizontal:= fhaLeft;
FVertical:= fvaTop;
end;

procedure T2DAlign.SetHorizontal(Value: Tem_MultiCore_HorAlign);
begin
if FHorizontal = Value then exit; FHorizontal:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure T2DAlign.SetVertical(Value: Tem_MultiCore_VertAlign);
begin
if FVertical = Value then exit; FVertical:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TPointClass.SetX(Value: integer);
begin
if FX = Value then exit; FX:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure TPointClass.SetY(Value: integer);
begin
if FY = Value then exit; FY:= Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_Bevel.Create;
begin
inherited;
FSides:= ALLGLSIDES;
end;

procedure Tem_MultiCore_Bevel.SetOuter(Value: TPanelBevel);
begin
if FOuter=Value then exit; FOuter:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_Bevel.SetInner(Value: TPanelBevel);
begin
if FInner=Value then exit; FInner:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_Bevel.SetSides(Value: Tem_MultiCore_Sides);
begin
if FSides=Value then exit; FSides:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;
```

```
procedure Tem_MultiCore_Bevel.SetBold(Value: boolean);
begin
if FBold=Value then exit; FBold:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor TIllumination.Create;
begin
inherited;
FShadowDepth:= 2;
end;

procedure TIllumination.SetShadowDepth(Value: integer);
begin
if Value < 0 then Value:=0;
if FShadowDepth=Value then exit; FShadowDepth:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_LabelTextStyles.Create;
begin
inherited;
FActive:= fstRaised;
FPassive:= fstRaised;
FDisabled:= fstPushed;
end;

procedure Tem_MultiCore_LabelTextStyles.SetPassive(Value:
Tem_MultiCore_TextStyle);
begin
if FPassive=Value then exit; FPassive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_LabelTextStyles.SetActive(Value:
Tem_MultiCore_TextStyle);
begin
if FActive=Value then exit; FActive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_LabelTextStyles.SetDisabled(Value:
Tem_MultiCore_TextStyle);
begin
if FDisabled=Value then exit; FDisabled:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_CustomTextColors.Create;
begin
inherited;
FText:= clBlack;
FTextDisabled:= clGray;
FDelineate:= clWhite;
FHighlight:= clBtnHighlight;
FShadow:= clBtnShadow;
FBackground:= clBtnFace;
end;

procedure Tem_MultiCore_CustomTextColors.SetText(Value: TColor);
begin
if FText=Value then exit; FText:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomTextColors.SetTextDisabled(Value: TColor);
begin
if FTextDisabled=Value then exit; FTextDisabled:=Value;
if Assigned(OnChanged) then OnChanged(self);
```

```
end;

procedure Tem_MultiCore_CustomTextColors.SetDelineate(Value: TColor);
begin
if FDelineate=Value then exit; FDelineate:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomTextColors.SetHighlight(Value: TColor);
begin
if FHighlight=Value then exit; FHighlight:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomTextColors.SetShadow(Value: TColor);
begin
if FShadow=Value then exit; FShadow:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomTextColors.SetBackground(Value: TColor);
begin
if FBackground=Value then exit; FBackground:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_CustomLabelColors.Create;
begin
inherited;
FTextActive:= clBlack;
FDelineateActive:= clWhite;
FAutoHighlight:= false;
FAutoShadow:= false;
FColorHighlightShift:= 40;
FColorShadowShift:= 60;
FBackgroundActive:= clBtnFace;
end;

procedure Tem_MultiCore_CustomLabelColors.SetTextActive(Value: TColor);
begin
if FTextActive=Value then exit; FTextActive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomLabelColors.SetDelineateActive(Value: TColor);
begin
if FDelineateActive=Value then exit; FDelineateActive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomLabelColors.SetAutoHighlight(Value: boolean);
begin
if FAutoHighlight=Value then exit; FAutoHighlight:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomLabelColors.SetAutoShadow(Value: boolean);
begin
if FAutoShadow=Value then exit; FAutoShadow:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomLabelColors.SetColorHighlightShift(Value:
integer);
begin
if FColorHighlightShift=Value then exit; FColorHighlightShift:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;
```

```
procedure Tem_MultiCore_CustomLabelColors.SetColorShadowShift(Value: integer);
begin
if FColorShadowShift=Value then exit; FColorShadowShift:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomLabelColors.SetBackgroundActive(Value: TColor);
begin
if FBackgroundActive=Value then exit; FBackgroundActive:=Value;
if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_GrBoxColors.Create;
begin
inherited;
FCaption:= clBtnFace;
FCaptionActive:= clBtnFace;
FClient:= clBtnFace;
FClientActive:= clBtnFace;
end;

procedure Tem_MultiCore_GrBoxColors.SetCaption(Value: TColor);
begin FCaption:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_MultiCore_GrBoxColors.SetCaptionActive(Value: TColor);
begin FCaptionActive:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_MultiCore_GrBoxColors.SetClient(Value: TColor);
begin FClient:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_MultiCore_GrBoxColors.SetClientActive(Value: TColor);
begin FClientActive:= Value; if Assigned(OnChanged) then OnChanged(self); end;

constructor Tem_MultiCore_ExtBevel.Create;
begin
inherited;
FActive:= true;
FBevelPenStyle:= psSolid;
FBevelPenWidth:= 1;
end;

procedure Tem_MultiCore_ExtBevel.SetActive(Value: boolean);
begin
if FActive = Value then exit;
FActive:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_ExtBevel.SetBevelPenStyle(Value: TPenStyle);
begin
if FBevelPenStyle = Value then exit;
FBevelPenStyle:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_ExtBevel.SetBevelPenWidth(Value: word);
begin
if FBevelPenWidth = Value then exit;
FBevelPenWidth:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_ExtBevel.SetInteriorOffset(Value: word);
begin
if FInteriorOffset = Value then exit;
FInteriorOffset:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_CustomListBoxItemStyle.Create;
begin
inherited Create;
FBevel:= Tem_MultiCore_Bevel.Create;
```

```
FFont:= TFont.Create;
end;

destructor Tem_MultiCore_CustomListBoxItemStyle.Destroy;
begin
FFont.Free; FBevel.Free; inherited;
end;

procedure Tem_MultiCore_CustomListBoxItemStyle.SetOnChanged(Value:
TNotifyEvent);
begin
FOnChanged:= Value; FBevel.OnChanged:= Value;
end;

procedure Tem_MultiCore_CustomListBoxItemStyle.SetColor(Value: TColor);
begin
if FColor = Value then exit;
FColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomListBoxItemStyle.SetDelineateColor(Value: TColor);
begin
if FDelineateColor = Value then exit;
FDelineateColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomListBoxItemStyle.SetFont(Value: TFont);
begin
FFont.Assign(Value); if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomListBoxItemStyle.SetTextStyle(Value:
Tem_MultiCore_TextStyle);
begin
FTextStyle:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_ListBoxItemStyle.Create;
begin
inherited Create;
FGradient:= TGradient.Create;
FTextGradient:= TGradient.Create;
end;

destructor Tem_MultiCore_ListBoxItemStyle.Destroy;
begin
FGradient.Free;
FTextGradient.Free;
inherited;
end;

procedure Tem_MultiCore_ListBoxItemStyle.SetOnChanged(Value: TNotifyEvent);
begin
inherited SetOnChanged(Value);
FGradient.OnChanged:= Value;
end;

constructor Tem_MultiCore_AskListBoxItemStyle.Create;
begin
inherited Create;
FBtnFont:= TFont.Create;
end;

destructor Tem_MultiCore_AskListBoxItemStyle.Destroy;
begin
FBtnFont.Free; inherited;
end;

procedure Tem_MultiCore_AskListBoxItemStyle.SetBtnColor(Value: TColor);
```

```
begin
if FBtnColor = Value then exit;
FBtnColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_AskListBoxItemStyle.SetBtnFont(Value: TFont);
begin
FBtnFont.Assign(Value); if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_AskListBoxItemStyle.SetBtnTextStyle(Value:
Tem_MultiCore_TextStyle);
begin
FBtnTextStyle:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_CustomBoxStyle.Create;
begin
inherited;
FPenStyle:= psSolid;
FHighlightColor:= clBtnHighlight;
FShadowColor:= clBtnShadow;
end;

procedure Tem_MultiCore_CustomBoxStyle.SetPenStyle(Value: TPenStyle);
begin  FPenStyle:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_MultiCore_CustomBoxStyle.SetHighlightColor(Value: TColor);
begin  FHighlightColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

procedure Tem_MultiCore_CustomBoxStyle.SetShadowColor(Value: TColor);
begin  FShadowColor:= Value; if Assigned(OnChanged) then OnChanged(self); end;

constructor Tem_MultiCore_CustomTextBoxStyle.Create;
begin
inherited;
FTextColor:= clBlack;
FBackgroundColor:= clWindow;
end;

procedure Tem_MultiCore_CustomTextBoxStyle.SetTextColor(Value: TColor);
begin  FTextColor:= Value; if Assigned(OnChanged) then OnChanged(self); end;

procedure Tem_MultiCore_CustomTextBoxStyle.SetBackgroundColor(Value: TColor);
begin  FBackgroundColor:= Value; if Assigned(OnChanged) then OnChanged(self);
end;

constructor Tem_MultiCore_BevelLines.Create;
begin
inherited;
FStyle:= bvLowered;
FThickness:= 1;
end;

procedure Tem_MultiCore_BevelLines.SetCount(Value: cardinal);
begin FCount:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_MultiCore_BevelLines.SetStep(Value: cardinal);
begin FStep:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_MultiCore_BevelLines.SetOrigin(Value: Tem_MultiCore_Origin);
begin FOrigin:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_MultiCore_BevelLines.SetStyle(Value: TPanelBevel);
begin FStyle:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_MultiCore_BevelLines.SetBold(Value: boolean);
begin FBold:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_MultiCore_BevelLines.SetThickness(Value: byte);
begin FThickness:= Value; if Assigned(OnChanged) then OnChanged(self); end;
procedure Tem_MultiCore_BevelLines.SetIgnoreBorder(Value: boolean);
begin FIgnoreBorder:= Value; if Assigned(OnChanged) then OnChanged(self); end;
```

```
{ TGradient }

procedure TGradient.Draw(DC: HDC; r: TRect; PenStyle, PenWidth: integer);
var
i,j,x,y,x2,y2,h,w,NumberOfColors:integer;
c1F,c2F,c3F:byte;
c1T,c2T,c3T:byte;
c1D,c2D,c3D:integer;
_R,_G,_B:byte;
Pen,OldPen:HPen;
FillBrush:HBrush;
BufferBmp, OldBMP:HBITMAP;
BufferDC,TargetDC:HDC;
ColorR:TRect;
LOGBRUSH:TLOGBRUSH;

procedure SwapColors;
var TempColor: Longint;
begin
TempColor:= FRGBFromColor; FRGBFromColor:= FRGBToColor; FRGBToColor:= TempColor;
end;
begin
if (Steps = 1)or(GetDeviceCaps( DC, BITSPIXEL )<16) then
begin exit;
FillBrush:= CreateSolidBrush( ColorToRGB( FromColor ) );
FillRect( DC, r, FillBrush );
DeleteObject( FillBrush );
exit;
end;
x:=r.left; y:=r.top; h:=r.bottom-r.top; w:=r.right-r.left;
x2:= 0; y2:= 0; pen:= 0; oldpen:= 0; BufferDC:= 0;

if Orientation = fgdHorzConvergent then
begin
FOrientation:= fgdHorizontal;
Draw(DC, Rect(R.Left, R.Top, R.Right, R.Bottom- h div 2), PenStyle, PenWidth);
SwapColors;
Draw(DC, Rect(R.Left, R.Top+ h div 2, R.Right, R.Bottom), PenStyle, PenWidth);
SwapColors;
FOrientation:= fgdHorzConvergent;
exit;
end;
if Orientation = fgdVertConvergent then
begin
FOrientation:= fgdVertical;
Draw(DC, Rect(R.Left, R.Top, R.Right- w div 2, R.Bottom), PenStyle, PenWidth);
SwapColors;
Draw(DC, Rect(R.Left+ w div 2, R.Top, R.Right, R.Bottom), PenStyle, PenWidth);
SwapColors;
FOrientation:= fgdVertConvergent;
exit;
end;

c1F:=Byte(FRGBFromColor );
c2F:=Byte(WORD(FRGBFromColor) shr 8);
c3F:=Byte(FRGBFromColor shr 16 );
c1T:=Byte(FRGBToColor );
c2T:=Byte(WORD(FRGBToColor) shr 8);
c3T:=Byte(FRGBToColor shr 16 );
c1D:=c1T- c1F;
c2D:=c2T- c2F;
c3D:=c3T- c3F;

if BufferedDraw then
begin
BufferDC:= CreateCompatibleDC(DC);
BufferBmp:= CreateBitmap( w, h, GetDeviceCaps( DC, PLANES ), GetDeviceCaps( DC,
BITSPIXEL ), nil );
```

```
OldBMP:= SelectObject( BufferDC, BufferBmp );
SetMapMode( BufferDC, GetMapMode(DC) );
TargetDC:= BufferDC;
end else TargetDC:= DC;

case Orientation of
fgdHorizontal:
begin
NumberOfColors:= min( Steps, h );
ColorR.Left:= r.left; ColorR.Right:= r.right;
end;
fgdVertical:
begin
NumberOfColors:= min( Steps, w );
ColorR.Top:= r.top; ColorR.Bottom:= r.bottom;
end;
fgdLeftBias, fgdRightBias:
begin
NumberOfColors:= min( Steps, w+h );
if PenStyle=0 then PenStyle:=PS_SOLID; if PenWidth=0 then PenWidth:=1;
y2:= y;
if Orientation = fgdLeftBias then x2:= x else
begin x:= r.right; x2:= r.right; end;
end;
else{fgdRectanEm_MultiCore}
begin
h:=h div 2; w:=w div 2;
NumberOfColors:= min( Steps, min(w,h) );
end;
end;
LOGBRUSH.lbStyle:= BS_HATCHED;
LOGBRUSH.lbHatch:= Ord(BrushStyle)- Ord(bsHorizontal);
for i:=0 to NumberOfColors-1 do
begin
_R:= c1F+ MulDiv(i, c1D, NumberOfColors- 1);
_G:= c2F+ MulDiv(i, c2D, NumberOfColors- 1);
_B:= c3F+ MulDiv(i, c3D, NumberOfColors- 1);

case Orientation of
fgdHorizontal, fgdVertical, fgdRectanEm_MultiCore:
begin
if BrushStyle = bsSOLID then
FillBrush:= CreateSolidBrush( RGB( _R, _G, _B ) )
else
begin
LOGBRUSH.lbColor:= RGB( _R, _G, _B );
FillBrush:= CreateBrushIndirect(LOGBRUSH);
end;

case Orientation of
fgdHorizontal:
begin
if fReverse then begin
ColorR.Top:= r.bottom- MulDiv( i, h, NumberOfColors);
ColorR.Bottom:= r.bottom- MulDiv( i+ 1, h, NumberOfColors);
end else begin
ColorR.Top:= r.top+ MulDiv( i , h, NumberOfColors);
ColorR.Bottom:= r.top+ MulDiv( i+ 1, h, NumberOfColors);
end;
end;
fgdVertical:
begin
if fReverse then begin
ColorR.Left:= r.right- MulDiv( i,w, NumberOfColors);
ColorR.Right:= r.right- MulDiv( i+ 1, w, NumberOfColors);
end else begin
ColorR.Left:= r.left+ MulDiv( i,w, NumberOfColors);
ColorR.Right:= r.left+ MulDiv( i+ 1, w, NumberOfColors);
end;
```

```
end;
fgdRectanEm_MultiCore:
begin
ColorR.Top:= r.top+ MulDiv( i, h, NumberOfColors);
ColorR.Bottom:= r.bottom- MulDiv( i, h, NumberOfColors);
ColorR.Left:= r.left+ MulDiv( i, w, NumberOfColors);
ColorR.Right:= r.right - MulDiv( i, w, NumberOfColors);
end;
end;
FillRect( TargetDC, ColorR, FillBrush );
DeleteObject( FillBrush );
end;
else {fgdLeftBias, fgdRightBias:}
begin
if Pen <> 0 then
DeleteObject(SelectObject( TargetDC, OldPen ));
Pen:= CreatePen( PenStyle, PenWidth, RGB( _R, _G, _B ) );
OldPen:= SelectObject( TargetDC, Pen );
for j:= 1 to MulDiv( i+ 1, h+w, NumberOfColors)- MulDiv( i, h+w, NumberOfColors)
do
begin
case Orientation of
fgdLeftBias:
begin
if y >= r.bottom then inc( x, PenWidth ) else y:= y+ PenWidth;
if x2 >= r.right then inc( y2, PenWidth ) else x2:= x2+ PenWidth;
MoveToEx( TargetDC, x, y, nil );
LineTo( TargetDC, x2, y2 );
end;
else{fgdRightBias:}
begin
if x <= r.left then inc( y, PenWidth ) else x:= x- PenWidth;
if y2 >= r.bottom then dec( x2, PenWidth ) else y2:= y2+ PenWidth;
MoveToEx( TargetDC, x, y, nil );
LineTo( TargetDC, x2, y2 );
end;
end;
end;
DeleteObject( SelectObject( TargetDC, OldPen ) );
end;
end;
if NumberOfColors=0 then exit;
if i/NumberOfColors*100 > PercentFilling then break;
end;

if BufferedDraw then
begin
BitBlt( DC, 0, 0, r.right-r.left, r.bottom-r.top, BufferDC, 0, 0, SRCCOPY );
DeleteObject( SelectObject( BufferDC, OldBMP ) );
DeleteDC( BufferDC );
end;

end;

function Tem_MultiCore_Bevel.BordersHeight: integer;
begin
Result:= 0;
if Inner <> bvNone then
begin
if fsdTop in Sides then inc(Result);
if fsdBottom in Sides then
if Bold then inc(Result, 1) else inc(Result);
end;
if Outer <> bvNone then
begin
if fsdTop in Sides then inc(Result);
if fsdBottom in Sides then
if Bold then inc(Result, 1) else inc(Result);
end;
```

```
end;

function Tem_MultiCore_Bevel.BordersWidth: integer;
begin
Result:= 0;
if Inner <> bvNone then
begin
if fsdLeft in Sides then inc(Result);
if fsdRight in Sides then
if Bold then inc(Result, 1) else inc(Result);
end;
if Outer <> bvNone then
begin
if fsdLeft in Sides then inc(Result);
if fsdRight in Sides then
if Bold then inc(Result, 1) else inc(Result);
end;
end;

function Tem_MultiCore_CustomListBoxItemStyle.HighlightColor: TColor;
begin
Result:= incColor(Color, 60);
end;
function Tem_MultiCore_CustomListBoxItemStyle.ShadowColor: TColor;
begin
Result:= decColor(Color, 60);
end;

end.
```