МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Кваліфікаційна наукова праця на правах рукопису

ІГНАТЧЕНКО МАРІЯ СЕРГІЇВНА

УДК 519.688:519.6:514.752

ДИСЕРТАЦІЯ

ЛІНГВІСТИЧНЕ ЗАБЕЗПЕЧЕННЯ СКІНЧЕННО-ЕЛЕМЕНТНОГО МОДЕЛЮВАННЯ У ПАРАЛЕЛЬНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

Подається на здобуття наукового ступеня **доктора філософії**

Дисертація містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ М. С. Ігнатченко

Науковий керівник: **Кудін Олексій Володимирович**, кандидат фізико-математичних наук, доцент

Запоріжжя – 2021

АНОТАЦІЯ

Ігнатченко М. С. Лінгвістичне забезпечення скінченно-елементного моделювання у паралельних обчислювальних системах. – Кваліфікаційна наукова праця на правах рукопису. Дисертація на здобуття наукового ступеня доктора філософії за спеціальністю 122 “Комп’ютерні науки”. – Запорізький національний університет, Запоріжжя, 2021.

Дисертаційна робота присвячена розробці лінгвістичного та програмного забезпечення скінченно-елементного аналізу в паралельних обчислювальних системах.

На сьогодні створення нової техніки практично неможливе без широкого застосування сучасних інформаційних технологій та обчислювальних систем. Однією з найважливіших задач, які постають перед інженерами, є заміна фізичних випробувань дослідних зразків складних інженерно-технічних систем, що проєктуються, віртуальним комп’ютерним експериментом. Це дозволяє, з одного боку, значно зменшити витрати часу та ресурсів на проєктування, а з іншого, підвищити його якість.

Одним з найбільш поширених на практиці підходів до чисельного аналізу широких класів задач є застосування методу скінченних елементів. Для автоматизації розрахунків за його допомогою на сьогодні створено велику кількість різноманітного програмного забезпечення. Найбільш відомими серед комерційних систем скінченно-елементного аналізу є Abaqus, Ansys, MSC Nastran та інші. Серед програмного забезпечення з відкритим вихідним кодом можна виділити dial.II, FreeFEM, OpenCAD тощо. Кількість таких програм неухильно збільшується, оскільки постійно зростає складність задач, які постають перед інженерами та науковцями. Крім того, слід враховувати стрімкий розвиток можливостей сучасної обчислювальної техніки. Для ефективного застосування наявних обчислювальних ресурсів сучасних мультипроцесорів та

мультикомп’ютерів необхідно розробляти паралельні версії наявних алгоритмів скінченно-елементного аналізу.

Таким чином, проблема створення систем скінченно-елементного аналізу з відкритим програмним кодом, які б дозволяли використовувати можливості різних паралельних архітектур сучасних комп’ютерних систем, є актуальною.

На сьогодні найбільш поширеними класами паралельних комп’ютерів є: 1) мультипроцесори (обчислювальні системи зі спільною пам’яттю і декількома процесорами або одним процесором з багатьма ядрами) та 2) мультикомп’ютери (обчислювальні системи, що фактично є об’єднаними в єдину мережу окремими комп’ютерами). Програмна реалізація паралельних розрахунків в цих системах істотно різниться, оскільки, наприклад, в мультипроцесорних системах завдяки наявності спільної пам’яті не потрібно реалізовувати складний інтерфейс синхронізації даних.

Отже, на сьогодні актуальною є задача розробки таких програмних систем скінченно-елементного аналізу, які б можна було використовувати на різних типах паралельних архітектур. Для розв’язання цієї задачі в дисертаційній роботі пропонується застосування твірного патерну проєктування Prototype, за допомогою якого можна створювати копії об’єктів, не вдаючись у подробиці їхньої реалізації. Це дозволяє, з одного боку, паралельно запускати програмні потоки на обчислювальних вузлах різної природи (ядрах процесора; окремих процесорах або вузлах обчислювального кластеру), а з іншого, створювати стандартизований програмний код, зменшити час на його налагодження, а також мінімізувати можливість виникнення помилок в коді за рахунок використання верифікованих рішень.

Сучасні програмні системи скінченно-елементного аналізу за своєю архітектурою можуть можуть бути поділені на три окремі функціональні підсистеми: 1) препроцесор, який виконує автоматизацію побудови дискретної геометричної моделі об’єкту розрахунку; 2) процесор – ядро програми, яке безпосередньо реалізує чисельний розрахунок задачі із застосуванням методу

скінченних елементів; 3) постпроцесор, що реалізує автоматизацію аналізу отриманих числових результатів та їх візуалізацію.

Задача побудови скінченно-елементної моделі, яку вирішує препроцесор, є складною і може бути поділена на дві окремі складові: 1) створення формального опису геометрії об’єкту розрахунку у певній формі, зручній для подальшої автоматичної обробки і 2) генерація скінченно-елементної моделі на основі цього опису.

Найбільш природнім та універсальним способом опису довільних геометричних областей є функціональний підхід, який базується на побудові такої функції, яка приймає від’ємні значення за межами вихідного геометричного об’єкта, і невід’ємні в його середині та на границі. Можливість побудови таких функцій для будь-якого геометричного об’єкта була доведена академіком В. Л. Рвачовим.

Використання функціонального підходу в препроцесорі потребує розробки формального способу опису функціональних моделей геометричних об’єктів. В дисертаційній роботі з цією метою було розроблено проблемно-орієнтовану мову FORL-G, яка дозволяє описувати геометричні моделі об’єктів будь-якої форми. В роботі із застосуванням розширеної форми Бекуса-Наура наведено повний опис синтаксису та семантики цієї мови, а також приклади її застосування. Особливістю FORL-G є наявність в ній засобів, що керують процесом побудови скінченно-елементних моделей в паралельних обчислювальних системах. Також в дисертаційній роботі із застосуванням патерну проєктування Prototype програмно реалізовано паралельні алгоритми побудови скінченно-елементних моделей геометричних об’єктів, заданих функціонально. Було проведено обчислювальний експеримент, який підтвердив ефективність запропонованих підходів.

Програмна реалізація типового процесору системи скінченно-елементного аналізу передбачає створення окремих модулів, що реалізують певний тип розрахунку. Альтернативним підходом є автоматизація виведення необхідних розрахункових співвідношень безпосередньо з варіаційних принципів, що

дозволяє отримувати необхідні співвідношення для чисельного розв'язання широких класів задач. Його застосування потребує розробки формального способу опису варіаційних формул і правил виведення з них необхідних співвідношень. В дисертаційній роботі розроблено предметно-орієнтовану мову FORL-F, за допомогою якої користувач може описувати комп'ютерні моделі широких класів задач. В роботі наведено формальний опис цієї мови, а також приклади її застосування. За допомогою шаблону Prototype програмно реалізовано процесор, який виконує розрахунки програм, описаних на мові FORL-F, в паралельних обчислювальних системах. Було проведено низку обчислювальних експериментів, які підтвердили ефективність запропонованого підходу.

Також в дисертації наведено опис запропонованого паралельного алгоритму візуалізації чисельних розрахунків, на його основі програмно реалізовано постпроцесор, а також виконано відповідні обчислювальні експерименти.

Отже, у дисертаційній роботі вирішена актуальна науково-технічна проблема підвищення ефективності розробки систем скінченно-елементного аналізу з використанням паралельних обчислень.

Ключові слова: паралельні розрахунки, патерн проєктування Prototype, метод скінченних елементів, препроцесор, процесор, постпроцесор.

ABSTRACT

Ihnatchenko M. S. Linguistic support of finite element modeling in parallel computing systems. – Qualifying scientific work on the rights of the manuscript. The dissertation on competition of a scientific degree of the doctor of philosophy on a specialty 122 "Computer Science". – Zaporizhia National University, Zaporizhia, 2021.

The thesis is devoted to the development of linguistic and software finite element analysis in parallel computer systems.

Today, the creation of new technology is almost impossible without the widespread use of modern information technology and computer systems. One of the most important problems facing engineers is to replace the physical tests of prototypes of designed complex engineering systems a virtual computer experiment. This allows on the one hand to significantly reduce the cost of time and resources for design, and on the other – to improve its quality.

One of the most common in practice approaches to numerical analysis of wide classes of problems is the use of the finite element method. To automate computing with its help, a large number of different software has been created today. The most well-known among commercial programs of finite element analysis are Abaqus, Ansys, MSC Nastran and others. Among the open source software are dial.II, FreeFEM, OpenCAD, etc. The number of such programs is steadily increasing, as the complexity of the problems facing engineers and scientists is constantly increasing. In addition, the rapid development of modern computer technology should be taken into account. In order to effectively use the available computing resources of modern multiprocessors and multicomputers, it is necessary to develop parallel versions of existing finite element analysis algorithms.

Thus, the problem of development of systems of finite element analysis with open source software, which would allow to use the capabilities of different parallel architectures of modern computer systems, is relevant.

Today, the most common classes of parallel computers are: 1) multiprocessors (computing systems with shared memory and multiple processors or a single processor with multiple cores); 2) multicomputers (computing systems that are actually connected to a single network by separate computers). The software implementation of parallel computing in these systems differs significantly, because, for example, in multiprocessor systems, due to the presence of shared memory, it is not necessary to implement a complex data synchronization interface.

Thus, today the task of developing such software systems of finite element analysis, which could be used on different types of parallel architectures, is relevant. To

solve this problem, the dissertation proposes the use of a creative design pattern Prototype, with which you can create copies of objects without going into the details of their implementation. This allows, on the one hand, to run software streams in parallel on computing nodes of different nature (processor cores; individual processors or on nodes of a computing cluster), and on the other – to create standardized program code, reduce debugging time, and minimize the possibility of errors in code through the use of verified solutions.

Modern software systems of finite element analysis by their architecture can be divided into three separate functional subsystems: 1) preprocessor, which automates the construction of a discrete geometric model of the object of calculation; 2) processor – the core of the program, which directly implements the numerical calculation of the problem using the finite element method; 3) postprocessor, which implements automation of analysis of numerical results and their visualization.

The task of constructing a finite element model, which is solved by the preprocessor, is complex and can be divided into two separate components: 1) creating a formal description of the geometry of the object of calculation in a form convenient for further automatic processing and 2) generating a finite element model based on this description.

The most natural and universal way to describe arbitrary geometric domains is a functional approach, which is based on constructing a function that takes negative values outside the original geometric object and is non-negative in its middle and at the boundary. The possibility of constructing such functions for any geometric object was proved by Academician V. L. Rvachev.

Using a functional approach in a preprocessor requires the development of a formal way to describe functional models of geometric objects. For this purpose, the dissertation developed a problem-oriented language FORL-G, which allows you to describe geometric models of objects of any shape. In the work with the use of the extended Backus-Naur form, a complete description of the syntax and semantics of this language is given, as well as examples of its application. A feature of FORL-G is the

presence of tools that control the process of building finite-element models in parallel computing systems. Also in the dissertation work with the use of the design pattern Prototype programmatically implemented parallel algorithms for constructing finite-element models of geometric objects, given functionally. A computational experiment was conducted, which confirmed the effectiveness of the proposed approaches.

Software implementation of a typical processor of the finite element analysis system involves the creation of separate modules that implement a certain type of calculation. An alternative approach is to automate the derivation of the necessary calculation relations directly from the principles of variation, which allows to obtain the necessary relations for the numerical solution of wide classes of problems. Its application requires the development of a formal way to describe the variational formulas and rules for deriving the necessary relations from them. In the dissertation the subject-oriented language FORL-F is developed, with the help of which the user can describe computer models of wide classes of tasks. The paper provides a formal description of this language, as well as examples of its use. Using the Prototype template, a processor is implemented that performs calculations of programs described in the FORL-F language in parallel computer systems. A number of computational experiments were performed, which confirmed the effectiveness of the proposed approach.

Also in the dissertation the description of the offered parallel algorithm of visualization of numerical calculations is given, on its basis the postprocessor is programmatically realized, and also the corresponding computational experiments are executed.

Thus, in the dissertation the actual scientific and technical problem of increase of efficiency of development of systems of finite element analysis with use of parallel computing is solved.

Keywords: parallel computing, Prototype design pattern, finite element method, preprocessor, processor, postprocessor.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ, В ЯКИХ ОПУБЛІКОВАНІ ОСНОВНІ НАУКОВІ РЕЗУЛЬТАТИ ДИСЕРТАЦІЇ:

1) Математичне забезпечення інженерного аналізу об’єктів аерокосмічної техніки на базі хмарних технологій: монографія / С. В. Чопоров, О. В. Кудін, Є. В. Панасенко, Д. Д. Грищак, М. С. Ігнатченко [за наук. ред. С. В. Чопорова]. Херсон: Видавничий дім “Гельветика”, 2020. 300 с.

2) Mathematical and computer modelling of engineering systems : Collective monograph / In edition by Corresponding Member of the National Academy of Sciences of Ukraine V. S. Hudramovich. Riga, Latvia: “Baltija Publishing”, 2020. 164 p.

3) Ігнатченко М. С., Кудін О. В. Візуалізація геометричних областей складної форми в паралельних обчислювальних системах зі спільною пам’яттю. *Вісник Запорізького національного університету.* 2019. № 2. С. 48-54.

4) Игнатченко М. С., Кудин А. В., Гнездовский А. В. Объектно-ориентированная реализация библиотеки конечно-элементного анализа на языке программирования Python. *Вісник Запорізького національного університету. Фізико-математичні науки*. 2020. № 1. С. 138-147.

5) Игнатченко М. С. Параллельная реализация алгоритма marching cubes с использованием паттерна проєктирования Prototype. *Colloquium-journal.* 2021. №10 (97). P. 49-52.

6) А.с. № 100690. Комп’ютерна програма “Об’єктно-орієнтована бібліотека класів, що реалізують інтерпретацію складних арифметичних виразів “PARSER” / С. В. Чопоров, С. І. Гоменюк, М. С. Ігнатченко, М. І. Клименко, О. А. Головань. – опубл. 18.11.2020.

7) Игнатченко М. С., Гнездовский А. В. Объектно-ориентированное моделирование задач механики. *Сучасні проблеми машинобудування*. Конференція молодих вчених та спеціалістів: зб. тез доп. Харків: Інститут проблем машинобудування НАН України, 2016. С. 23-24.

8) Ігнатченко М. С. Паралельний алгоритм пошуку границі двовимірної геометричної області, заданої неявною функцією. *Інформаційні технології та взаємодії (IT&I – 2019)*. VI міжнародна науково-практична конференція. Матеріали доповідей. Київ: Київський національний університет імені Тараса Шевченка, 2019. С. 24-25.

9) Ігнатченко М. С., Кудін О. В. Моделювання складних геометричних областей із застосуванням функціонального підходу. *Актуальні проблеми математики та інформатики*: тези доповідей Десятої Всеукраїнської, сімнадцятої регіональної наукової конференції молодих дослідників. Запоріжжя: Запорізький національний університет, 2019. С. 97-98.

10) Ігнатченко М. С., Кудін О. В. Візуалізація R-функцій в паралельних обчислювальних системах зі спільною пам’яттю. *Інформаційні технології в металургії та машинобудуванні. ITMM’2021*: тези доповідей міжнародної науково-практичної конференції імені професора Михальова О. І. Дніпро: НМетАУ, 2020. С. 161-165.

11) Ігнатченко М. С., Кудін О. В. Застосування паралельних обчислювальних систем зі спільною пам’яттю для візуалізації R-функцій. *Актуальні проблеми математики та інформатики*: Збірка тез доповідей Одинадцятої Всеукраїнської, вісімнадцятої регіональної наукової конференції молодих дослідників (м. Запоріжжя, 23-24 квітня 2020 р.). Херсон: Видавничий дім “Гельветика”, 2020. С. 26-27.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

АСП – автоматизована система проєктування

ГЕ – граничний елемент

ЛМЖ – локальна матриця жорсткості

МСЕ – метод скінченних елементів

ООП – об’єктно-орієнтоване програмування

ПЗ – програмне забезпечення

РФБН – розширена форма Бекуса-Наура

САПР – система автоматизації проєктувальних робіт

СЕ – скінченний елемент

СЛАР – система лінійних алгебраїчних рівнянь

ASCII – American standard code for information interchange

BREP – boundary representation

CSG – constructive solid geometry

DSL – domain-specific language

FREP – function representation

FEA – finite element analysis

FEM – finite element method

GML – generative modelling language

GUI – graphical user interface

MVC – model-view-controller

UML – unified modeling language

ВСТУП

Актуальність теми. Розвиток сучасного машинобудування та будівництва в наш час практично неможливий без застосування сучасних інформаційних технологій і, в першу чергу, автоматизованих систем проєктування (АСП). Важливим аспектом їх застосування є заміна тривалого й дорогого фізичного випробування дослідного зразка (часто з його знищенням) віртуальним комп'ютерним експериментом [1].

Аналіз міцності й довговічності складних інженерно-технічних систем зазвичай полягає в аналізі їх напружено-деформованого стану, що вимагає розв'язання відповідних класів задач математичної фізики. На жаль, більшість таких актуальних задач аналітично не розв'язуються. Для цього на практиці застосовують різноманітні наближені чисельні методи. Одним з найбільш поширених та ефективних серед них є метод скінченних елементів (МСЕ) [2-7]. Його використання без застосування обчислювальної техніки практично неможливе, тому на сьогодні створено велику кількість різноманітних програмних засобів, які автоматизують ті чи інші аспекти розрахунків за допомогою МСЕ. До найбільш відомих комерційних автоматизованих систем скінченно-елементного аналізу належать Abaqus [8], Ansys [9], COMSOL [10], LS-DYNA [11], MSC Nastran [12], SOLIDWORKS [13] та ін. [14]. Серед вільного програмного забезпечення можна виділити, наприклад, такі програми, як dial.II [15], FreeFEM [16], OpenCAD [17] та інші [18, 19]. Їх кількість постійно збільшується, оскільки регулярно створюються нові композиційні конструктивні матеріали (вуглепластики, еластомери, металокераміка тощо), для врахування особливостей яких потрібно розробляти нові або вдосконалювати вже наявні моделі та методи розрахунку. Крім того, постійно зростає складність задач, які постають перед інженерами та науковцями. Також слід враховувати стрімкий розвиток можливостей сучасних комп'ютерів – останнім часом майже всі вони

облаштовуються декількома окремими процесорами (ядрами). Для ефективного застосування наявних обчислювальних ресурсів необхідно розробляти паралельні версії алгоритмів скінченно-елементного аналізу.

Таким чином, на сьогодні проблема створення систем скінченно-елементного аналізу з відкритим програмним кодом, які б дозволяли використовувати можливості паралельної архітектури сучасних комп'ютерних систем, є актуальною.

Зв'язок роботи з науковими програмами, планами, темами. Одержані в дисертаційній роботі результати повністю відповідають основним напрямам наукових досліджень, що виконуються у Запорізькому національному університеті. Зокрема, робота виконувалася у відповідності до плану науково-технічних робіт Запорізького національного університету при виконанні науково-дослідної теми "Математичне та програмне забезпечення автоматизованого проєктування аерокосмічної техніки" (номер державної реєстрації 0118U000210), яка фінансувалася за рахунок державного бюджету України, і де автор був одним з виконавців.

Мета і задачі дослідження. Метою роботи є вирішення актуальної проблеми підвищення ефективності розробки програмного забезпечення для скінченно-елементного аналізу складних інженерно-технічних систем, що проєктуються.

Для досягнення поставленої мети необхідно розв'язати наступні задачі:

1) провести аналіз відповідної предметної області, а саме виконати огляд наявних методів і підходів до процесу скінченно-елементного моделювання складних інженерно-технічних систем;

2) розробити спеціалізовану предметно-орієнтовану мову для формального опису:

– геометричної моделі досліджуваного об'єкту та параметрів її дискретизації на скінченні елементи (СЕ) заданої форми;

– математичної моделі і схеми чисельного розрахунку задачі із застосуванням МСЕ;

3) розробити паралельні алгоритми, що реалізують різні аспекти скінченно-елементного аналізу на основі формального опису задачі;

4) розробити відповідне програмне забезпечення із застосуванням сучасних технологій і інструментів програмування, таких, наприклад, як шаблони (патерни) проєктування;

5) виконати тестові розрахунки, які б довели ефективність запропонованого лінгвістичного забезпечення та алгоритмів.

Об'єктом дослідження є процес розробки програмного забезпечення для скінченно-елементного аналізу задач математичної фізики.

Предметом дослідження є лінгвістичне забезпечення скінченно-елементного моделювання у паралельних обчислювальних системах.

Методи дослідження базуються на застосуванні апарату математичного аналізу, аналітичної геометрії, теорії R-функцій, обчислювальної математики, об'єктно-орієнтованого аналізу та паралельних обчислень.

Наукова новизна одержаних результатів полягає у вирішенні актуальної науково-технічної проблеми підвищення ефективності розробки систем скінченно-елементного аналізу задач математичної фізики з використанням паралельних обчислень.

При виконанні дисертаційної роботи отримано такі наукові результати:

− *вперше* запропоновано проблемно-орієнтовану мову FORL-G, за допомогою якої користувач може здійснювати функціональне моделювання дво- та тривимірних геометричних областей довільної форми, і яка підтримує паралельні методи обчислень, що суттєво прискорює процес побудови скінченно-елементних моделей;

− *вперше* запропоновано проблемно-орієнтовану мову FORL-F, яка дозволяє із застосуванням варіаційних принципів описувати чисельні схеми

розрахунку широких класів задач математичної фізики в паралельних обчислювальних системах;

− *вперше* запропоновано методологію розробки систем скінченно-елементного аналізу в паралельних обчислювальних системах із застосуванням патерну проєктування Prototype;

− *отримав подальшого розвитку* метод функціонального моделювання геометричних областей із застосуванням теорії R-функцій і паралельних розрахунків;

− *отримав подальшого розвитку* паралельний алгоритм виведення розрахункових співвідношень методу скінченних елементів із варіаційних принципів;

− *отримав подальшого розвитку* метод візуалізації результатів чисельного розрахунку в паралельних обчислюваних системах;

− *вперше* створено автоматизовану систему проєктування, в якій програмно реалізовані запропоновані підходи та алгоритми.

Практичне значення одержаних результатів. Розроблені в дисертаційній роботі підходи дозволяють значно підвищити якість програмної реалізації систем скінченно-елементного аналізу крайових задач в паралельних обчислювальних системах.

Запропонований програмний засіб femsolver дозволяє автоматизувати всі аспекти застосування МСЕ: від побудови якісних дискретних моделей, до чисельного розрахунку й візуалізації отриманих результатів. Він реалізований у вигляді програми з відкритим вихідним кодом із застосуванням стандартної бібліотеки STL мови C++, а також безкоштовних бібліотек Boost та Eigen.

Апробація результатів дослідження. Результати дисертаційного дослідження було оприлюднено на наступних наукових семінарах і конференціях:

− Конференції молодих вчених та спеціалістів “Сучасні проблеми машинобудування” (Харків, 2016 р.);

− VI міжнародній науково-практичній конференції “Інформаційні технології та взаємодії (IT&I – 2019)” (Київ, 2019 р.);

− Десятій Всеукраїнській, сімнадцятій регіональній науковій конференції молодих дослідників “Актуальні проблеми математики та інформатики” (Запоріжжя, 2019 р.);

− Міжнародній науково-практичній конференції імені професора Михальова О. І. “Інформаційні технології в металургії та машинобудуванні ІТММ’2020” (м. Дніпро, 2020 р.);

− Одинадцятій Всеукраїнської, вісімнадцятій регіональній науковій конференції молодих дослідників “Актуальні проблеми математики та інформатики” (м. Запоріжжя, 2020 р.);

− VIII Міжнародній науково-технічній конференції «Актуальні проблеми прикладної механіки та міцності конструкцій» (м. Запоріжжя, 2020 р.).

Публікації. Основні положення роботи було опубліковано у 11 наукових працях, із них: 2 монографії (у тому числі 1 зарубіжна); 2 опубліковані у виданнях, що визнані як фахові з фізико-математичних наук; 1 стаття – у періодичному зарубіжному науковому виданні; 1 авторське свідоцтво, а також 5 тез доповідей на науково-практичних конференціях.

Структура та обсяг роботи. Дисертація складається зі вступу, п’яти розділів, які містять 22 підрозділи, висновків, списку використаних джерел та додатків. Загальний обсяг дисертації становить 137 сторінок, у т. ч. основного тексту – 117 сторінок. Список використаних джерел налічує 102 найменування (9 сторінок). Додатки – 12 сторінок.

1 АНАЛІЗ СТАНУ ПРОБЛЕМИ

1.1 Огляд сучасних автоматизованих систем проєктування

Розвиток сучасної техніки (машинобудування, будівництва, енергетики тощо) на сьогодні неможливий без широкого застосування АСП. Теперішній рівень можливостей обчислювальної техніки всіляко сприяє постійному підвищенню їх ролі. Без використання комп'ютерів наразі практично неможливе проєктування та створення складних машин, механізмів та споруд.

Автоматизація проєктувальних робіт завдяки використанню інформаційних технологій може застосовується в трьох основних напрямах:

1) при виконанні рутинних інженерних робіт, таких, наприклад, як створенні креслень, підготовці документації, автоматизації документообігу і таке інше;

2) для аналізу властивостей проєктованого об'єкта, який зазвичай полягає в дослідженні відповідності характеристик створюваного об'єкту вимогам замовника;

3) при виконанні таких задач проєктування, які не підлягають повній формалізації, наприклад, трасуванні друкованих плат; створенні різноманітних розкладів, інформаційній підтримці процесу прийняття рішень тощо) [1].

Найбільш складним з точки зору реалізації та важливим для кінцевого результату є другий напрям застосування АСП, а саме автоматизація аналізу властивостей об'єкту проєктування, яка дозволяє замінити дорогі й тривалі натурні випробування дослідного зразка віртуальним експериментом. Його суть полягає у побудові та дослідженні за допомогою обчислювальної техніки відповідної математичної моделі проєктованого об'єкта. Крім того, слід зазначити,

що фізичні випробування дослідних зразків можуть бути не тільки економічно затратними, а іноді й приводити до катастрофічних наслідків (наприклад, при створенні аерокосмічної техніки). Саме тому застосування АСП дозволяє не тільки істотно спростити й здешевити процес розробки нової техніки, а й зробити його більш безпечнішим. Крім того, слід зазначити, що посилення конкурентної боротьби постійно вимагає від промисловості зменшення витрат на розробку та випуск продукції. Саме тому, наприклад, у вітчизняній аерокосмічній галузі заміна фізичних руйнівних випробувань віртуальним комп'ютерним експериментом є конче актуальною задачею.

При проєктуванні складних інженерно-технічних систем в машинобудуванні та будівництві головним зазвичай є оцінка їх міцності та довговічності. Це вимагає проведення дослідження напружено-деформованого стану проєктованих об'єктів, що, в свою чергу, потребує розв'язання різних класів задач механіки. Оскільки актуальні задачі математичної фізики в тривимірній постановці аналітичними методами в більшості випадків не можуть бути розв'язані, на практиці застосовують різноманітні наближені чисельні методи, найбільш поширеним серед яких є МСЕ [2-7].

Застосування МСЕ без залучення обчислювальної техніки є практично неможливим. Тому для автоматизації розв'язання різних класів задач із застосуванням цього чисельного методу на сьогодні створено велику кількість різноманітного програмного забезпечення, яке виконує всі необхідні розрахунки: від генерації дво- або тривимірних дискретних (скінченно-елементних) моделей досліджуваного об'єкту, до візуалізації великих масивів отриманих чисельних результатів. Різновид АСП, що застосовуються для скінченно-елементного аналізу (FEA – Finite Element Analysis), зазвичай називають FEM-системами (FEM – Finite Element Method).

До найбільш відомих пропрієтарних програмних FEM-систем, що застосовуються для наближеного розв'язання широких класів крайових задач, належать Abaqus [8], Ansys [9], COMSOL [10], LS-DYNA [11], MSC Nastran [12],

SOLIDWORKS [13] та ін. [14]. Альтернативою їм є безкоштовні або умовно безкоштовні системи з відкритим програмним кодом, такі, наприклад, як dial.II [15], FreeFEM [16], OpenCAD [17] та ін. [18, 19].

Сучасні FEM-системи, що автоматизують застосування МСЕ для розв'язання різних класів задач, за функціональним призначенням можуть бути умовно поділеними на три окремі підсистеми:

− препроцесор – виконує автоматизацію підготовки вихідних для подальшого чисельного розрахунку даних (частіше за все на цій стадії створюється геометрична модель об'єкту розрахунку й виконується процес її дискретизації на заданий тип скінченних елементів);

− процесор – центральна частина (ядро) скінченно-елементного програмного забезпечення, яка безпосередньо виконує чисельний розрахунок певної задачі (побудову локальних матриць жорсткості, маси та демпфування для кожного СЕ та їх додавання (ансамблювання) до відповідних глобальних матриць; врахування крайових умов; формування та розв'язання системи лінійних алгебраїчних рівнянь (СЛАР) тощо);

− постпроцесор – підсистема, що автоматизує складний та трудомісткий аналіз великої кількості отриманих числових результатів (частіше за все шляхом їх візуалізації), а також синтез нової інформації на основі раніше отриманої (рис. 1.1).

Головною метою застосування препроцесору FEA-системи є створення дискретної скінченно-елементної моделі вихідної геометричної області. Вона поділяється на дві окремі задачі:

1) створення формального опису геометрії вихідного об'єкту, що проєктується або досліджується;

2) генерація на основі цього опису геометрії скінченно-елементної моделі, утвореної із СЕ заданого типу.

Найбільш непростою є перша задача, оскільки формальний опис геометричної моделі об'єкту складної форми є досить нетривіальної проблемою

[20]. Класичними підходами до побудови геометричних моделей складних об’єктів є такі методи, як каркасне моделювання, граничне подання і конструктивна блокова геометрія.

Рис. 1.1 – Типова архітектура сучасної FEA-системи

Каркасне моделювання (wireframe modelling) дозволяє описувати тривимірні геометричні об’єкти із застосуванням лише таких геометричних примітивів, як точки та лінії, а також певних операцій трансформації над ними, таких, наприклад, як витягування, обертання, зсув тощо [21]. Каркасна модель не містить інформації про грані об’єкту, що істотно ускладнює процедуру генерації дискретної моделі, придатної для чисельного розрахунку. Отже, такий тип

геометричного моделювання не дуже зручний для застосування в АСП, хоча й широко застосовується, наприклад, в такій відомій системі автоматизації проєктувальних робіт (САПР), як AutoCAD [22].

Граничне подання (BREP – Boundary Representation) описує модель геометричного об’єкту (дво- або тривимірного) як певну сукупності поверхонь, що обмежують його границю [23]. Граничне подання дозволяє однозначно описувати геометричні моделі областей довільної форми, проте практичне моделювання геометричних областей складної форми із застосуванням такого підходу є складною задачею [20]. Найбільш відомими програмними системами, що базується на застосуванні BREP, є Gmsh [24] і GRUMMP [25].

Конструктивна блокова геометрія (CSG – Constructive Solid Geometry) дозволяє будувати геометричні моделі у вигляді певної сукупності стандартних тривимірних геометричних примітивів (куби, сфери, циліндри, призми тощо) й логічних операцій об’єднання, перетину й доповнення (різниці) над ними [26]. Такий підхід є зручним й достатньо ефективним, але його практичне застосування істотно ускладнюється в разі необхідності моделювання об’єкту нетипової форми або відсутності в бібліотеці необхідного геометричного примітиву. Відомими програмними продуктами, що дозволяють будувати із застосуванням CSG дво- та тривимірні геометричні моделі та виконувати автоматичне їх розбиття на скінченні елементи, є Netgen [27] та Patran [28].

Таким чином, застосування класичних підходів до геометричного моделювання не завжди дозволяє ефективно й зручно описувати топологію геометричних областей складної форми у тривимірному випадку. Одним з можливих варіантів вирішення цієї проблеми є застосування апарату R-функцій, запропонованих В. Л. Рвачовим [29]. Він полягає в описі геометричної області у вигляді співвідношення $F(x,y,z)\geq 0$, яке однозначно й несуперечливо описує внутрішню частину області ($F(x,y,z)>0$) й її границю ($F(x,y,z)=0$). Такий підхід отримав назву функціонального подання (FREP – Function Representation) [30]. Він є вельми перспективним, але його практичне застосування пов’язано з

рядом певних складностей [20], в тому числі і необхідністю створення спеціальних проблемно-орієнтованих мов для опису FREP-моделей. Прикладом препроцесору, який базується на застосуванні FREP-підходу для геометричного моделювання, є система qzCAD [18, 31].

Центральною частиною (ядром) любої програмної системи скінченно-елементного аналізу є процесор або вирішувач. Його головною функцією є побудова для кожного СЕ вихідної скінченно-елементної моделі локальних матриць жорсткості, маси та демпфування; ансамблювання їх з відповідними глобальними матрицями; формування СЛАР з урахуванням крайових умов; розв'язання СЛАР тощо. Структура МСЕ-процесору залежить від типу задач, на розв'язання яких він орієнтований (статика, динаміка, в'язко-пружність, контактні задачі, акустика тощо). Серед великої кількості наявних на сьогодні МСЕ-процесорів можна (окрім вже згаданих) відзначити такі бібліотеки, як OFELI [32], GetFEM++ [33], Hermes [34], MFEM [35] та ін. [18, 19, 36-38].

Велика кількість наявних вирішувачів (значна частина з яких до того ж є пропрієтарними) робить незручним їх застосування при розв'язанні різних типів задач математичної фізики. Отже, виникає задача створення легковагих універсальних АСП, придатних для розв'язання широкого кола типів задач. Одним з можливих варіантів розв'язання цієї задачі є створення проблемно-орієнтованої мови для формального опису постановки задачі та методу її розв'язання.

Постпроцесор сучасної АСП призначений для виконання двох основних функцій:

1) підвищення ефективності та наочності аналізу великих масивів чисельних даних, отриманих в результаті застосування МСЕ;

2) автоматизація синтезу додаткової інформації на базі раніше отриманої [39].

Частіше за все головною функцією постпроцесора є графічне представлення (візуалізація) результатів розрахунку у зручній для сприйняття

людиною формі (епюри, графіки, ізолінії, напівтонові або кольорові зображення тощо).

У так званих “важких” (повнофункціональних) АСП та САПР (наприклад, Ansys, MSC Nastran, SOLIDWORKS та ін.) постпроцесор може бути реалізовано у вигляді невід’ємної складової FEA-програми. Але часто їх розробляють як окремі програмні модулі, що можуть підтримувати велику кількість вихідних форматів даних і працювати з різними процесорами. Так, наприклад, написаний на мові програмування Python [40-43] постпроцесор Plot3d [44] дозволяє виконувати візуалізацію результатів чисельного розрахунку в таких FEA-системах, як QFEM [19] та PyFEM [45].

Для аналізу результатів розрахунку широкого класу задач механіки еластомерів в системі МІРЕЛА+ [37] застосовується постпроцесор MIRELA [46]. Для візуалізації розрахунку різноманітних процесів втоми матеріалів та їх руйнування застосовується препроцесор P-FAT [47]. Для застосування спільно з системою MSC Nastran використовується пре- і постпроцесор Patran [28]. Крім того, в якості постпроцесорів можуть бути використані такі системи, як Gmsh [24], NGSolve [27] та ін. [48].

Як вже зазначалося, важливою функцією постпроцесору окрім візуалізації результатів розрахунку є синтез додаткових даних (наприклад, розрахунок компонент тензорів деформацій та напружень за раніше отриманими переміщеннями, або розрахунок інтенсивності напружень за відомими напруженнями). Такі розрахунки можуть бути достатньо складними й займати багато часу. Крім того, при аналізі результатів розрахунку нестаціонарних задач сучасний препроцесор повинен вміти візуалізувати дані в динаміці. Тому при реалізації постпроцесору виникає задача формального опису параметрів синтезу додаткових даних та підвищення ефективності їх розрахунку.

1.2 Постановка мети та задач дослідження

Не зважаючи на велику кількість наявного програмного забезпечення для скінченно-елементного аналізу різних класів задач механіки, часто виникає потреба у їх вдосконаленні або створенні нових програм. Це пояснюється, наприклад, тим фактом, що регулярно створюються конструктивні матеріали, які раніше не існували (вуглепластики, еластомери, металокераміка тощо), для врахування особливостей яких потрібно створювати нові або адаптувати наявні моделі та методи розрахунку. Крім того, останнім часом майже всі комп'ютерні системи обладнуються декількома окремими процесорами (або одним багатоядерним). Для ефективного застосування всіх наявних обчислювальних ресурсів необхідно розробляти паралельні (розподілені) версії алгоритмів скінченно-елементного аналізу певних класів крайових задач.

Таким чином, резюмуючи все вищесказане, можна зробити висновок про те, що на сьогодні актуальною є проблема створення таких "легковагих" систем скінченно-елементного аналізу, які б, з одного боку, дозволяли користувачеві формально описувати постановку широких класів задач математичної фізики і методів їх розв'язання, а з іншого, давали можливість враховувати особливості паралельної архітектури сучасних комп'ютерних систем.

Це вимагає розв'язання наступних задач:

1) проведення критичного аналізу відповідної предметної області, огляд наявних методів і підходів до процесу скінченно-елементного моделювання складних інженерно-технічних систем;

2) створення спеціалізованої предметно-орієнтованої мови (лінгвістичного забезпечення), за допомогою якої користувач міг би описувати:

– геометричну модель вихідного об'єкту, що досліджується, та параметри її дискретизації на СЕ заданої форми;

– математичну модель задачі та метод її розрахунку із застосуванням МСЕ;

– формули розрахунку додаткових даних на основі раніше отриманих;

3) розробки паралельних алгоритмів, що реалізують різні аспекти скінченно-елементного аналізу на основі програми, описаної користувачем на раніше розробленій мові;

4) розробки відповідного програмного забезпечення із застосуванням сучасних технологій і інструментів програмування, таких, наприклад, як шаблони (патерни) проєктування.

Висновки до розділу 1

Отже, на підставі виконаного аналізу можна зробити висновок про те, що задача розробки лінгвістичного та програмного забезпечення систем скінченно-елементного моделювання у паралельних обчислювальних системах є актуальною.

Для її ефективного розв’язання необхідно:

– розробити проблемно-орієнтовані мови, які дадуть змогу користувачеві формально описувати широкі класи задач математичної фізики: від геометричної моделі, до методів їх розрахунку й параметрів візуалізації результатів;

– реалізувати відповідні алгоритми та методи для роботи в паралельних обчислювальних системах із застосуванням придатних для цього шаблонів проєктування;

– виконати тестові розрахунки для верифікації отриманих результатів.

Основні наукові і практичні результати даного розділу опубліковано в роботах [49, 50].

2 КАРКАС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектурні шаблони програмного забезпечення скінченно-елементного аналізу

Розробка та супровід сучасного програмного забезпечення (ПЗ) дедалі стає все більш складним і тривалим процесом. Одним з варіантів підвищення його ефективності є застосування при реалізації складних програмних систем (САПР, АСП, систем скінченно-елементного аналізу тощо) універсальних узагальнених методів та типових програмних рішень, які можуть повторно використовуватися – патернів (шаблонів) проєктування.

Вперше шаблони проєктування програмного забезпечення були запропоновані наприкінці 80-х років минулого сторіччя і зараз вони є дуже вживаними в індустрії розробки ПЗ [51-53]. На сьогодні найбільш популярною й ефективною технологією розробки ПЗ є об’єктно-орієнтоване програмування (ООП) [54, 55]. Проте, оскільки значна частина наукового ПЗ була традиційно реалізована на мові програмування Fortran [56], який тривалий час не підтримував ООП, то розробники часто уникали застосування цієї технології. Крім того, використання ООП може інколи призводити до зменшення швидкодії розрахункових програм за рахунок збільшення накладних витрат при обробці перезавантажених операцій, а також до збільшення їх фізичного розміру (тобто й необхідної оперативної пам’яті) при застосуванні узагальненого програмування [57].

Але, не зважаючи на ці недоліки, за рахунок того, що наукове ПЗ стає все більш складним і великим, саме застосування ООП дозволяє підвищити гнучкість його розробки та супроводу [58]. Тому, при розробці такого ПЗ на сьогодні зазвичай застосовують певні архітектурні шаблони, під якими розуміються деякі

узагальнені рішення в архітектурі програмного забезпечення, які застосовуються у певному заданому контексті [59]. При цьому під архітектурним шаблоном розуміється типове рішення, яке застосовується в конкретному контексті.

Найбільш поширеними на практиці при розробці складного ПЗ є наступні архітектурні шаблони:

- багаторівнева архітектура;
- канали й фільтри;
- клієнт-серверна архітектура;
- MVC (Model-View-Controller) та інші [59].

Багаторівнева (або n-рівнева) архітектура є однією з найбільш поширених, оскільки її застосування добре узгоджується зі структурним аналізом та об'єктно-орієнтованою декомпозицією [55]. Очевидно, що наведена на рис. 1.1 типова структура системи скінченно-елементного аналізу повністю відповідає цьому шаблону архітектури ПЗ.

Архітектура "канали й фільтри" застосовується в тому випадку, коли процес роботи ПЗ можна розбити на декілька окремих кроків, кожен з яких буде виконуватися окремим оброблювачем. Головними елементами цієї архітектури є "фільтр" і "канал", а також додаткові "джерело даних" та "споживач даних". Кожен потік обробки даних розпочинається джерелом даних і закінчується їх споживачем. Він представляється серією фільтрів та каналів, що чергуються. При цьому канали забезпечують передачу даних і їх синхронізацію. Задачею фільтра є приймання на вході даних, їх обробка та трансформація в деяке інше представлення. Після чого дані при необхідності передаються далі. Прикладом застосування такої архітектури може бути препроцесор, якщо розглядати його архітектуру як послідовність фільтрів: обробка формального опису геометрії вихідного об'єкта, побудова первинної апроксимації границі області, її оптимізація, побудова дискретної моделі, її оптимізація тощо.

Клієнт-серверна архітектура також належить до числа однієї з найбільш вживаних на практиці. При застосуванні такої архітектури для розробки систем

скінченно-елементного аналізу обчислювальні завдання розподіляються між постачальниками послуг – серверами, і замовниками послуг – клієнтами. Їх внутрішня архітектура і програмна реалізація можуть бути абсолютно незалежними один від одного, а для обміну даними між ними застосовується спеціальне лінгвістичне забезпечення або протоколи. Так, наприклад, при реалізації FEA-систем архітектура “клієнт-сервер” може застосовуватися як для відокремлення реалізації інтерфейсної частини ПЗ від розрахункового ядра (що дозволяє створювати кросплатформні версії програм), так і для реалізації паралельних розрахунків в мультикомп’ютерах (системах з розподіленою пам’яттю).

Архітектура MVC (модель-представлення-контролер) розділяє функціонал ПЗ на компоненти трьох стандартних типів:

– модель, яка містить дані програми;

– представлення, що реалізує відображення певної частини даних і реалізує взаємодію з користувачем;

– контролер, що виконує роль посередника між моделлю та представленням, а також керує повідомленнями про зміну стану.

Цей тип архітектури є дуже поширеним при реалізації графічного інтерфейсу (GUI – Graphical User Interface), оскільки він дозволяє відокремити функціональність користувацького інтерфейсу від функціоналу ПЗ і забезпечити при цьому можливість внесення швидких змін як інтерфейс програми, так і в реалізацію її бізнес-логіки.

Як видно з цього огляду, сучасне складне ПЗ для скінченно-елементного аналізу може складатися з підсистем, при розробці кожної з яких використовувалися різні архітектурні шаблони або їх певні композиції. Проте, реалізація кожної архітектури вимагає застосування певних патернів програмування, що істотно спрощує та пришвидшує процес розробки програм.

Патерни проєктування ПЗ є ефективними засобами розв’язання широкого кола задач, що виникають при проєктування складних програмних систем. При

цьому слід зазначити, що об’єктно-орієнтований патерн (шаблон) є не закінченою реалізацією певного програмного коду, а лише зразком вирішення деякої проблеми в певному контексті. Зазвичай шаблон тільки визначає відношення між класами та об’єктами, а не вказує, як воно буде програмно реалізоване.

2.2 Патерни проєктування наукового програмного забезпечення

Під патерном або шаблоном проєктування розуміється типовий спосіб вирішення певної задачі, з якою часто стикаються розробники при проєктуванні архітектури ПЗ. На відміну від бібліотек функцій чи класів, шаблон проєктування не можна напряму застосувати, просто додавши його до програми. Він не є програмним кодом, а лише загальним принципом вирішення певної проблеми. Іншими словами шаблон проєктування потрібно адаптувати для конкретної програми. Таким чином, на відміну від алгоритму, який є набором чітких і однозначних команд, патерн є високорівневим описом певного рішення [51]. Він зазвичай складається з наступних елементів:

– опису проблеми, яку він вирішує;

– опису переваг, які надає патерн при вирішення проблеми способом, що він пропонує;

– структура класів, що реалізують вирішення проблеми;

– конкретний приклад (певною мовою програмування);

– опис особливостей його реалізації в різних ситуаціях;

– наявні зв’язки з іншими шаблонами проєктування [53].

Застосування патернів проєктування при розробці ПЗ надає ряд переваг, таких, наприклад, як:

– стандартизація програмного коду (використання типових уніфікованих рішень);

– використання вже верифікованих іншими розробниками рішень, що дозволяє зменшити час на розробку й налагодження програм;

– уніфікація термінології, що дає можливість покращити рівень комунікації з замовником та або іншими колегами та зменшити кількість помилок, що можуть виникати завдяки різному тлумаченню термінів.

Шаблони проєктування зазвичай можуть класифікуватися за складністю застосування, деталізацією або рівнем охоплення ПЗ, що проєктується. До найбільш низькорівневих шаблонів проєктування відносять так звані “ідіоми” [60]. Їх головний недолік – залежність від конкретної мови програмування.

До найбільш універсальних шаблонів відносяться “архітектурні патерни”, що можуть бути реалізовані із застосуванням будь-якої об’єктно-орієнтованої мови програмування. Зазвичай такі патерни застосовуються для проєктування всієї програмної системи, а не її окремих частин.

Але на практиці патерни найчастіше класифікуються за їх призначенням:

– твірні (породжуючі) шаблони (абстрагують процес створення нових об’єктів);

– структурні шаблони (описують способи утворення з класів та об’єктів більших за розмірами структур);

– поведінкові шаблони (реалізують способи комунікації між об’єктами).

Прикладами найбільш поширених твірних патернів, що реалізують безпечне та зручне створення нових об’єктів, є:

– “Фабричний метод” (Factory Method) – визначає загальний інтерфейс для створення об’єктів в базовому класі, дає можливість похідним класам змінювати типи об’єктів, що створюються;

– “Абстрактна фабрика” (Abstract Factory) – реалізує можливість створення сімейства класів без прив’язки до певних типів;

– “Будівельник” (Builder) – реалізує можливість створення складних об’єктів крок за кроком, використовуючи один і той же програмний код для отримання різних реалізацій об’єктів;

– “Одинак” (Singleton) – реалізує унікальність класу та глобальну точку доступу до нього;

– “Прототип” (Prototype) – визначає інтерфейс для створення копій поточного об’єкту без врахування подробиць його реалізації.

Структурні патерни проєктування забезпечують створення зручних в реалізації та подальшій підтримці ієрархій класів. Найбільш популярними серед них є:

– “Адаптер” (Adapter) – реалізує можливість взаємодії між об’єктами з різними інтерфейсами;

– “Міст” (Bridge) – дає можливість поділяти класи на дві окремі ієрархічні категорії – абстракцію та реалізацію, що дозволяє змінювати програмний код в певній гілці незалежно від іншої;

– “Компонувальник” (Composite) – реалізує можливість групування класів в одну структуру і працювати з нею, як з одним об’єктом;

– “Декоратор” (Decorator) – дає змогу динамічно додавати до класу нову функціональність шляхом застосування спеціальних класів-“обгорток” та ряд інших [51, 53].

Поведінкові патерни проєктування реалізують ефективну та безпечну взаємодію між об’єктами в ПЗ. Серед них можна виділити:

– “Ітератор” (Iterator) – реалізує можливість обходу певної колекції об’єктів без врахування їхньої внутрішньої структури;

– “Посередник” (Mediator) – зменшує кількість зв’язків між класами шляхом створення одного класу-посередника;

– “Спостерігач” (Observer) – дозволяє одним об’єктам стежити за станом та поведінкою інших та відповідно реагувати на певні події;

– “Стратегія” (Strategy) – дає можливість групувати схожі алгоритми в певному класі, що дозволяє при необхідності динамічно заміняти один алгоритм на інший;

– “Відвідувач” (Visitor) – реалізує додавання до ПЗ нових операцій без зміни класів, над об’єктами яких ці операції можуть виконуватися;

– “Стан” (State) – дає можливість об’єктам змінювати їх поведінку в залежності від їхнього поточного стану (без зміни класу).

2.3 Застосування патерну Prototype при реалізації паралельних програм

В наш час у зв’язку зі стрімким розвитком обчислювальної техніки та істотним ускладненням задач, що постають, наприклад, в машинобудуванні та будівництві, різко загострюється проблема розробки паралельних реалізацій чисельних методів розв’язання диференційних рівнянь в часткових похідних і, в першу чергу, МСЕ. Причому ця задача ускладнюється наявністю принципово різних типів паралельної обчислювальної техніки.

На сьогодні поділ паралельних обчислювальних систем на різновиди зазвичай виконується із застосуванням таксономії (класифікації) Флінна (M. Flynn) [61]. У відповідності до неї паралельні комп’ютери поділяються на типи згідно до способів організації в них взаємодії обчислювальних потоків (команд, що паралельно виконуються) і оброблюваних даних. Теоретично, можуть існувати такі типи паралельних комп’ютерів [62-64]:

– SISD (Single Instruction, Single Data) – стандартні послідовні обчислювальні системи, в яких єдиному потоку команд відповідає єдиний потік даних;

– SIMD (Single Instruction, Multiple Data) – обчислювальні системи з єдиним потоком команд і множинним потоком даних (багатопроцесорні комп’ютери, в яких одна й та ж команда в один момент часу може оброблювати декілька різних потоків даних);

– MISD (Multiple Instruction, Single Data) – комп’ютери, в яких множинному потоку команд відповідає єдиний потік даних (ідеалізована архітектура, яка на практиці не застосовується);

– MIMD (Multiple Instruction, Multiple Data) – обчислювальна систем з множинними потоками команд і даних (більшість сучасних паралельних багатопроцесорних систем належать саме до цього класу).

Слід зазначити, що на сьогодні найбільш поширеною категорією паралельних комп’ютерів є саме MIMD [63]. Тому для зручності така категорія паралельних комп’ютерів в свою чергу розподіляється на дві окремі групи:

1) мультипроцесори (multiprocessors) – обчислювальні системи зі спільною пам’яттю і декількома процесорами або одним процесором з багатьма ядрами (найбільш поширена на сьогодні група комп’ютерів) (рис. 2.1);

2) мультикомп’ютери (multicomputers) – обчислювальні системи, що фактично є об’єднаними в єдину мережу окремими комп’ютерами (кластери або масово-паралельні системи) (рис. 2.2).

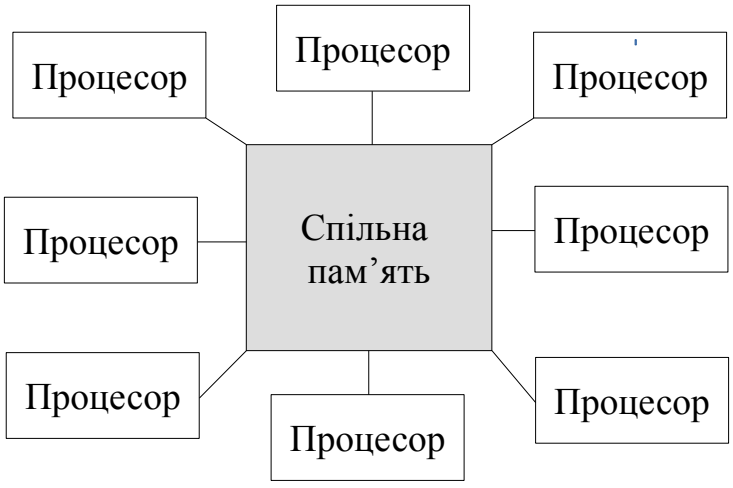

Рис. 2.1 – Архітектура мультипроцесора

В свою чергу кожна з цих груп поділяється на певні підгрупи. Так, наприклад, системи з розподіленою пам’яттю поділяються на дві окремі групи: масивно-паралельні системи (MPP – massively parallel processor) і кластери

(clusters), які на сьогодні є найбільш популярними [65]. Серед мультипроцесорів розрізняються, наприклад, обчислювальні системи з виключним використанням лише кеш-пам’яті процесорів; системи з когерентністю кешу процесорів та комп’ютери, в яких без апаратної підтримки когерентності кешу реалізовано спільний доступ до пам’яті різних процесорів.

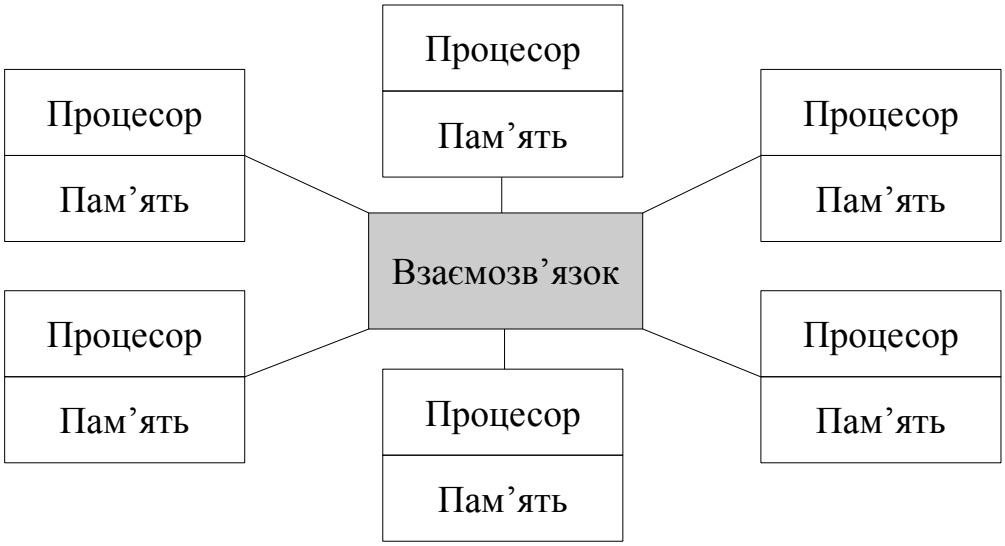

Рис. 2.2 – Архітектура мультикомп’ютера

Програмування паралельних розрахунків в цих системах істотно різниться. Так для мультипроцесорних систем завдяки наявності спільної пам’яті не потрібно реалізовувати інтерфейс синхронізації даних, оскільки всі вони зазвичай знаходяться в спільному віртуальному просторі. Проте в мультикомп’ютерах синхронізація обміну даних між окремими вузлами є досить складною процедурою і в загальному випадку вимагає істотних витрат ресурсів [64]. Для реалізації паралельних розрахунків в системах зі спільною пам’яттю частіше за все застосовується відкритий стандарт і однойменна бібліотека OpenMP [66]. Для розробки паралельних програм в мультикомп’ютерних системах частіше за все застосовують стандарт і відповідну бібліотеку MPI [67].

Таким чином, постає задача розробки таких систем скінченно-елементного аналізу, які б без істотної переробки можна було використовувати на різних типах

паралельних архітектур. Для розв’язання цієї задачі перспективним є застосування твірного патерну проєктування Prototype, який дає змогу створювати копії об’єктів, не вдаючись у подробиці їхньої реалізації [51]. Це дозволить з одного боку паралельно запускати програмні потоки на обчислювальних вузлах різної природи (ядрах процесора; окремих процесорах або на вузлах обчислювального кластеру), а з іншого – створювати стандартизований програмний код, а також мінімізувати кількість потенційних помилок в ньому за рахунок використання верифікованих рішень та зменшити час на налагодження програм.

Суть патерну. Як вже зазначалось, Prototype належить до породжувальних патернів проєктування.

Проблема. Його зазвичай застосовують, коли виникає проблема створення об’єктів, які є точними копіями один одного. Стандартні засоби копіювання об’єктів не завжди можна застосовувати, оскільки неможливо отримати доступ до приватних членів певного об’єкту. Крім того, програмний код, який виконує копіювання є специфічним для певного класу об’єкту, який він копіює.

Рішення. Для розв’язання проблеми копіювання об’єктів згідно з шаблоном Prototype необхідно доручити процес копіювання самим об’єктам, для яких необхідно створити клони. Це реалізується створенням загального інтерфейсу для всіх об’єктів, який зазвичай реалізується у вигляді спеціального методу *clone*(). Його реалізація в різних класах є практично однаковою – *clone*() створює новий об’єкт поточного класу, а також виконує копіювання в цей об’єкт значення всіх атрибутів вихідного об’єкта, який прийнято називати прототипом.

Слід зазначити, що при практичній реалізації цього патерну всі необхідні прототипи створюються та налаштовуються на етапі ініціалізації програми, а їх клонування відбувається при необхідності на етапі виконання програми.

Структура. Шаблон Prototype зазвичай має певну ієрархічну деревоподібну структуру, де коренем є базовий абстрактний клас, який у більшості випадків містить єдиний чистий віртуальний метод *clone*(). Класи-нащадки, що утворюють “гілки дерева”, містять певну реалізацію цього методу для копіювання всіх

атрибутів конкретного класу. При цьому, така реалізація зазвичай повністю прихована від клієнта (користувача), який застосовує клас.

Клієнт при необхідності виконує клонування потрібних йому об’єктів, використовуючи загальний для всієї ієрархії класів інтерфейс *clone*() [51]. На рис. 2.3 наведено відповідну UML-діаграму [68].

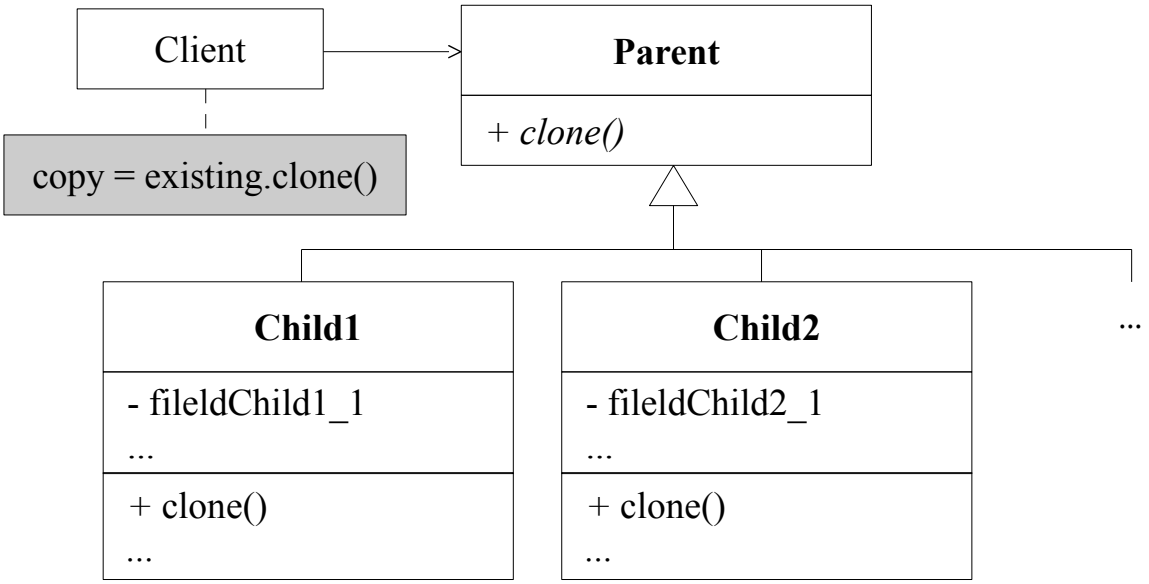

Рис. 2.3 – UML-діаграма реалізації патерну Prototype

Реалізація. Розглянемо приклад застосування шаблону Prototype при паралельній реалізації побудови локальних матриць жорсткості (ЛМЖ) [2] в різних обчислювальних архітектурах.

Із застосуванням універсального псевдокоду [69] відповідну ієрархію класів можна описати, наприклад, таким чином.

```
// Базовий прототип
abstract class FE is
    // Координати СЕ
    filed x: array of float
    filed y: array of float
    filed z: array of float
```

```
        // Пружні характеристики
        field e: float
        field m: float
        // Матриця коефіцієнтів ЛМЖ
        field data: array of array of float
        // …
        // Стандартний конструктор
        constructor FE() is
            // …
        // Конструктор прототипу
        constructor FE(x: array of float, y: array of float, z: array of float,
                    e: float, m: float) is
            this()
            this.x = x
            this.y = y
            this.z = z
            this.e = e
            this.m = m
        // …
        // Генерація ЛМЖ
        method generate() is
            // …
        // Абстрактний метод клонування
        abstract method clone(x: array of float, y: array of float, z: array of float,
                    e: float, m: float): FE
        // …
// Прототип для роботи в системах зі спільною пам’яттю
class FE_OpenMP extends FE is
        constructor FE_OpenMP(x: array of float, y: array of float,
```

```
                z: array of float, e: float, m: float) is
            super(x, y, z, e, m)
        // …
        // Метод клонування
        method clone(x: array of float, y: array of float, z: array of float,
                e: float, m: float): FE is
            return new FE_OpenMP(x, y, z, e, m)
        // …
    // Прототип для роботи в мультикомп'ютерах
    class FE_MPI extends FE is
        constructor FE_MPI(x: array of float, y: array of float, z: array of float,
                e: float, m: float) is
            super(x, y, z, e, m)
        // …
        // Метод клонування
        method clone(x: array of float, y: array of float, z: array of float,
            e: float, m: float): FE is
            return new FE_MPI(x, y, z, e, m)
        // …
    // …
```

Тоді застосування цих класів в клієнтській програмі може мати, наприклад, такий вигляд.

```
    // Клієнтський код
    class Application is
        field fe: FE
        field global_data: array of array of float
        constructor Application() is
            if mpi then
```

```
            fe = new FE_MPI()
        else
            fe = new FE_OpenMP()
        end if
        // …
    // …
    method generate_global_matrix() is
        // …
        for i ∈ [0, num_thread − 1] do
            add(global_data, fe.clone(x, y, z, e, m)
            // ...
        end for
        // …
    // …
```

Застосування. Зважаючи на вищезазначене, можна відзначити, що застосування патерну Prototype дозволяє виконувати копіювання об’єктів, внутрішня структура яких невідома користувачеві, і при цьому істотно спростити програму за рахунок зменшення кількості класів в ній. З наведеного прикладу видно, що застосування цього шаблону проєктування дозволяє легко створювати паралельні додатки для різних розподілених обчислювальних архітектур.

Висновки до розділу 2

Таким чином, можна зробити висновок, що на сьогодні для підвищення ефективності систем скінченно-елементного аналізу необхідно застосовувати всі наявні можливості сучасної обчислювальної техніки. Одним з ефективних

способів вирішення цієї проблеми є застосування різних технологій паралельних розрахунків.

Найбільш поширеними на сьогодні архітектурами розподілених обчислювальних систем є мультипроцесори та мультикомп'ютери. Розробка паралельних програм для них істотно розрізняється. Отже, виникає задача розробки такої програмної архітектури, яка б дозволяла користувачеві при необхідності без значних змін програмного коду виконувати паралельні розрахунки на різних типах розподілених обчислювальних систем.

Аналіз наявних на сьогодні шаблонів проєктування ПЗ показав, що для досягнення цієї мети ефективним є застосування патерну Prototype, який дає змогу спроєктувати програму таким чином, щоб її можна було запускати в різних типах паралельних комп'ютерів без істотної зміни програмного коду.

Основні наукові і практичні результати даного розділу опубліковано в роботах [49, 50, 70].

3 ПАРАЛЕЛІЗАЦІЯ ПОБУДОВИ СКІНЧЕННО-ЕЛЕМЕНТНИХ МОДЕЛЕЙ

3.1 Функціональний підхід до моделювання геометричних областей

Головною функцією препроцесору будь-якої системи скінченно-елементного аналізу є автоматизація побудови дискретної моделі геометричної області, яку займає об'єкт розрахунку. Ця задача може бути поділена на дві окремі частини:

1) створення формального опису геометрії вихідного об'єкту розрахунку у деякій формі, зручній для подальшої автоматичної обробки із застосуванням комп'ютерної техніки;

2) генерація скінченно-елементної моделі за раніше отриманим формальним описом вихідного об'єкту [20].

Задача створення формального опису геометрії об'єкта складної форми є досить складною та творчою. Застосування класичних підходів, таких, як каркасне моделювання, граничне подання та конструктивна блокова геометрія можуть бути достатньо складними й незручними при моделюванні геометричних областей нетипової форми. Тому найбільш універсальним та природним способом геометричного моделювання областей довільної форми є застосування функціонального подання [20, 30], яке базується на використанні теорії R-функцій академіка В. Л. Рвачова [29]. Згідно з ним довільна геометрична область Ω в $\mathbb{R}^3$ може бути описана у вигляді певної функції $F=F(x,y,z)$, для якої виконуються наступні співвідношення: $F(x,y,z)\geq 0$, якщо $(x,y,z)\in\Omega$ (причому $F(x,y,z)=0$, якщо точка $(x,y,z)\in\Gamma$, де Γ – границя вихідної області Ω), і $F(x,y,z)<0$, якщо $(x,y,z)\notin\Omega$.

В. Л. Рвачов довів [29], що для будь-якої геометричної області Ω можна сконструювати відповідну їй R-функцію $F(x,y,z)$ за допомогою елементарних

математичних функцій та логічних операцій кон'юнкції, диз'юнкції та інверсії над ними (тобто розв'язати обернену задачу аналітичної геометрії).

Функція $F_1 \wedge F_2$ називається R-кон'юнкцією та визначається співвідношенням:

$$F_1 \wedge F_2 = (F_1 + F_2 - \sqrt{F_1^2 + F_2^2})/2. \quad (3.1)$$

Аналогічним чином, функція $F_1 \vee F_2$ називається R-диз'юнкцією та визначається формулою:

$$F_1 \vee F_2 = (F_1 + F_2 + \sqrt{F_1^2 + F_2^2})/2. \quad (3.2)$$

Функція $\bar{F}$ називається R-інверсією та обчислюється наступним виразом:

$$\bar{F} = -F. \quad (3.3)$$

Так, наприклад, функціональну модель тривимірної геометричної області, утвореної об'єднанням (диз'юнкцією) семи сфер різного радіусу (рис. 3.1), можна описати наступним чином:

$$F = F_1 \vee F_2 \vee F_3 \vee F_4 \vee F_5 \vee F_6 \vee F_7,$$

де $F_1(x, y, z) = R^2 - x^2 - y^2 - z^2$;

$F_2(x, y, z) = r^2 - (x - R)^2 - y^2 - z^2$;

$F_3(x, y, z) = r^2 - (x + R)^2 - y^2 - z^2$;

$F_4(x, y, z) = r^2 - x^2 - (y - R)^2 - z^2$;

$F_5(x, y, z) = r^2 - x^2 - (y + R)^2 - z^2$;

$F_6(x, y, z) = r^2 - x^2 - y^2 - (z - R)^2$;

$F_7(x, y, z) = r^2 - x^2 - y^2 - (z + R)^2$;

R, r – радіуси більшої та меншої сфери (відповідно).

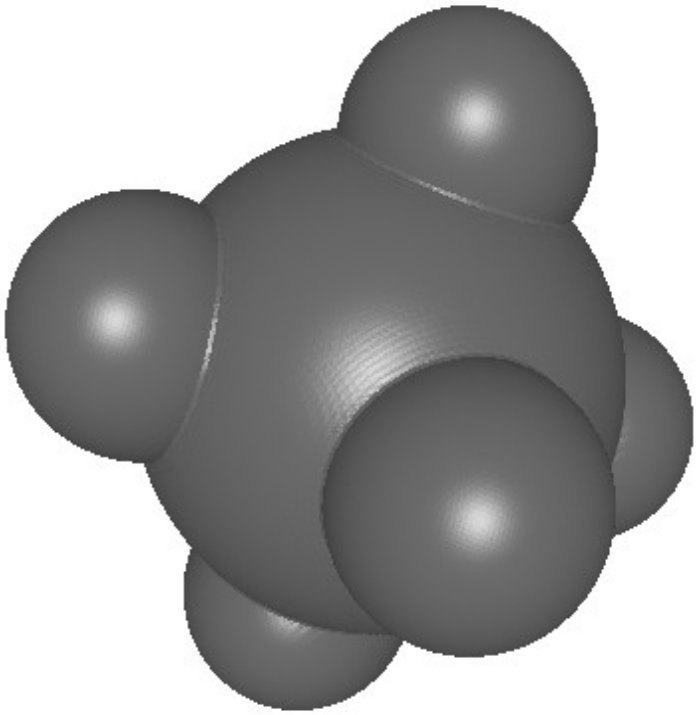

Рис. 3.1 – Фігура, що утворена об’єднанням семи
сфер різного радіусу

Очевидно, що, використовуючи математичний апарат аналітичної геометрії спільно із виразами (3.1)-(3.3), можна побудувати аналітичне співвідношення для будь-якої геометричної області. Проте, на жаль, практичне застосування функціонального підходу із застосуванням R-функцій до геометричного моделювання ускладнюється тим фактом, що функція $F(x, y, z)$ є неявною. Тобто пошук вузлів, які належать її границі, є достатньо нетривіальною задачею. Особливо вона ускладнюється при необхідності пошуку координат вузлів, які лежать на лініях перетину підобластей, що утворюють кінцеву геометричну область.

3.2 Проблемно-орієнтована мова опису геометричних моделей із застосуванням функціонального підходу FORL-G

Для однозначного опису геометричних моделей областей складної форми в першу чергу потрібно розробити відповідну формальну мову, зручну для автоматичної трансляції й обробки. Такі проблемно-орієнтовані мови для різних предметних областей мають загальну назву DSL (Domain-Specific Language).

Одним з найбільш відомих прикладів DSL для геометричного моделювання є GML (Generative Modelling Language) [71]. Ця мова призначена для опису складних тривимірних геометричних моделей із застосуванням парадигми генеративного моделювання [72]. Вона широко застосовується в комп'ютерній графіці для опису складних тривимірних моделей, але, на жаль, не підтримує задання об'єктів із застосуванням R-функцій. Таким чином, розробка DSL для функціонального моделювання тривимірних об'єктів складної форми є актуально задачею.

Для опису математичних моделей геометричних областей із застосуванням функцій В. Л. Рвачова необхідно, щоб відповідна предметно-орієнтована мова дозволяла описувати елементарні математичні співвідношення (арифметичні та логічні); мала підтримку стандартних математичних функцій та R-операцій, а також забезпечувала наявність достатнього набору операторів для задання допоміжних параметрів побудови дискретної моделі (застосування паралельних розрахунків, початкові параметри побудови моделі, її візуалізації тощо).

Для стандартизованого опису комп'ютерних мов зручним є застосування розширеної форми (нотації) Бекуса-Наура (РФБН) [73]. За допомогою РФБН можна послідовно описати одні синтаксичні конструкції формальної мови через інші, що є дуже зручним при реалізації транслятора відповідної мови.

Для функціонального опису геометричних моделей складних інженерно-технічних систем в паралельних обчислювальних системах пропонується нова

DSL-мова, яка отримала назву FORL-G (Formula Language – Geometric). Її формальний опис із застосуванням РФБН наведено нижче.

Слід зазначити, що на відміну від наявних на сьогодні аналогів [39, 71] мова FORL-G містить мовні конструкції для паралельної побудови дискретних моделей геометричних областей.

3.2.1 Основні символи мови FORL-G

За допомогою РФБН основні символи мови FORL-G можна представити наступним чином.

буква = “A”| “B” | “C” | “D” | “E” | “F” | “G” | “H” | “I” | “J” | “K” | “L” | “M” | “N” | “O” | “P” | “Q” | “R” | “S” | “T” | “U” | “V” | “W” | “X” | “Y” | “Z” | “a” | “b” | “c” | “d” | “e” | “f” | “g” | “h” | “i” | “j” | “k” | “l” | “m” | “n” | “o” | “p” | “q” | “r” | “s” | “t” | “u” | “v” | “w” | “x” | “y” | “z” | “_”

цифра = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

знак = “+” | “-”

спеціальний-символ = “+” | “-” | “*” | “/” | “^” | “(“ | “)” | “,” | “=” | “ {” | “}” | “@” | зарезервоване-слово

зарезервоване-слово = “abs” | “acos” | “and” | “asin” | “atan” | “atan2” | “cos” | “cosh” | “geometrical_model” | “object” | “exp” | “not” | “or” | “return” | “sin” | “sinh” | “tan” | “tanh”

ідентифікатор = буква { буква | цифра }

число-без-знака = ціле-без-знака | дійсне-без-знака

число-зі-знаком = [знак] число-без-знака

ціле-без-знака = послідовність-цифр

послідовність-цифр = цифра {цифра}

дійсне-без-знака = ціле-без-знака“.”дробова-частина [“E”|”e” порядок] |

ціле-без-знака [“E” | ”e” порядок]

дробова-частина = послідовність-цифр

порядок = ціле-зі-знаком

ціле-зі-знаком = [знак] ціле-без-знака

ASCII-послідовність = ASCII-символ | ASCII-послідовність

коментар = “//” [ASCII-послідовність]

Тут під терміном “ідентифікатор” розуміються певний лексичний примітив, який позначає сутність (змінні або ключові слова мови FORL-G). Слід зазначити, що, як і в більшості сучасних проблемно-орієнтованих формальних мов, семантика FORL-G забороняє вживати в якості ідентифікаторів зарезервовані слова.

Також слід підкреслити, що таке поняття, як “ASCII-символ” в РФБН формально не визначається. Під поняттям “ASCII-символ” розуміється будь-який з 256 стандартних символів таблиці кодування ASCII [74]).

3.2.2 Типи даних в FORL-G та правила їх утворення

Одним з найбільш фундаментальний понять любої предметно-орієнтованої формальної мови є “тип даних”, яке фактично описує формат збереження інформації (змінних та інших об’єктів, що застосовуються для опису алгоритму та/або певної сутності) і відповідний набір операцій над нею.

В мові FORL-G всі змінні належать до числових типів: цілих або дійсних (проте при виконанні розрахунків змінні цілого типу в FORL-G автоматично перетворюються в дійсні числа).

Довжина змінної залежить від поточної реалізації мови FORL-G.

У відповідності до РФБН тип даних можна визначити таким чином.

тип-даних = числовий-тип-даних

числовий-тип-даних = число-без-знака | число-зі-знаком

Явні перетворення типів даних в FORL-G не передбачені, неявні перетворення здійснюється за таким правилом числові константи різних типів завжди приводяться до старшого типу, яким є дійсне число.

Змінні в FORL-G задаються таким чином.

декларація-змінної = “variable” ідентифікатор [= арифметичний-вираз] {, ідентифікатор [= арифметичний-вираз]}

Слід зазначити, що люба змінна в FORL-G перед використанням повинна бути задекларована. При декларації їй може бути присвоєно початкове значення у вигляді числової константи або арифметичного виразу (яке формально буде описано нижче).

Всім змінним, при декларації без початкової ініціалізації автоматично присвоюється значення “0”. Крім того, змінній при необхідності може бути присвоєне нове значення. Робиться це таким чином.

опис-змінної = ідентифікатор “=” арифметичний-вираз

3.2.3 Арифметичні вирази в FORL-G

За допомогою РФБН поняття арифметичного виразу в мові FORL-G можна описати таким чином.

арифметичний-вираз = простий-арифметичний-вираз | логічний-вираз

логічний-вираз = простий-арифметичний-вираз логічна-операція простий-арифметичний-вираз | логічний-вираз логічна-операція логічний-вираз

простий-арифметичний-вираз = константа | змінна | елементарна-функція | простий-арифметичний-вираз арифметична-операція простий-арифметичний-вираз

константа = число-зі-знаком

змінна = ідентифікатор

елементарна-функція = ідентифікатор “(“ [список-параметрів] “)”

список-параметрів = параметр {“,” параметр}

параметр = арифметичний-вираз | ідентифікатор

арифметична-операція = “+” | “-” | “*” | “/” | “^”

логічна-операція = “not” | “and” | “or”

Таким чином, арифметичний вираз в мові FORL-G є або простим арифметичним виразом (наприклад, R^2 – (x – a)^2 – (y – b)^2 – (z - c)^2)), або логічним виразом (наприклад, (F1 and F2) or (F3 and not F4)), або їх певною комбінацією.

Арифметичні та логічні вирази в мові FORL-G призначені для опису формул, з яких конструюються R-функції. В FORL-G для зручності користувача допускається опис складної R-функції у вигляді комбінації певної кількості більш простих функцій. Крім того, при формальному описі геометричних об’єктів складної форми в мові FORL-G передбачено наявність набору стандартних вбудованих елементарних математичних функцій. Їх повний перелік наведено в табл. 3.1.

Таблиця 3.1 – Реалізовані в FORL-G елементарні математичні функції

№	Функція	Опис
1	**2**	**3**
1	abs(x)	Абсолютне значення
2	acos(x)	Арккосинус
3	asin(x)	Арксинус
4	atan(x)	Арктангенс
5	atan2(x, y)	Арктангенс y / x, що виражений в радіанах
6	cos(x)	Косинус
7	cosh(x)	Косинус гіперболічний
8	exp(x)	Експонента

Продовження таблиці 3.1

1	2	3
9	sin(x)	Синус
10	sinh(x)	Синус гіперболічний
11	sqr(x)	Квадрат
12	sqrt(x)	Квадратний корінь
13	tan(x)	Тангенс
14	tanh(x)	Тангенс гіперболічний

3.2.4 Структура функціонального опису геометричного об'єкта із застосуванням мови FORL-G

Опис R-функції із застосуванням мови FORL-G згідно з нотацією РФБН може бути представлений таким чином.

опис-геометричного-об'єкту = "@geometrical_model" "(" назва-моделі [, "thread =" ціле-без-знака] ")" початок-блоку функціональний-блок { функціональний-блок } кінець-блоку

назва-моделі = "ідентифікатор"

функціональний-блок = "function" ім'я-функції "(" список-аргументів ")" початок-блоку [декларація-змінної { декларація-змінної }] [опис-змінної {опис-змінної}] результат кінець-блоку

ім'я-функції = ідентифікатор

початок-блоку = "{"

кінець-блоку = "}"

список-аргументів = ідентифікатор "," ідентифікатор ["," ідентифікатор]

результат = "return" арифметичний-вираз

Таким чином, функціональний опис геометричного об’єкту на мові FORL-G складається з непустої множини функціональних блоків (секції “function”), які можуть бути призначеними для опису певної R-функції або її частини.

Функціональний-блок може складатися з чотирьох частин:

1) заголовка;
2) декларації змінних (з можливою ініціалізацією);
3) опису змінних;
4) результатного виразу, який визначає підсумковий геометричний об’єкт або його певну складову частину.

Заголовок функціонального блоку визначає його назву та розмірність (два координатних параметри при описі двовимірної області, три – тривимірної).

Блок декларації змінних задає список ідентифікаторів, які застосовуються в подальших арифметичних виразах. Як вже зазначалося, при необхідності всі змінні можуть бути проініціалізовані при їх декларації. Цей блок є необов’язковим в разі, якщо в формулах застосовуються лише константи та координатні параметри.

Блок опису змінних також є необов’язковим. Він містить певну сукупність операторів присвоювання, в яких допоміжним змінним задаються певні значення.

Обов’язковим в FORL-G є блок, який визначає підсумкову формулу. Він складається з ключового слова “return”, після якого повинен бути описаний арифметичний вираз, що задає R-функцію або її певну складову.

Отже, структура функціонального опису геометричного об’єкту із застосуванням мові FORL-G фактично має модульну структуру, де кожен модуль може описувати як окрему R-функцію, так і її складові. Це дозволяє з одного боку підвищити наочність опису геометричної моделі, а з іншого виконувати реалізацію транслятора мови FORL-G із застосуванням паралельних розрахунків [75-77], що істотно підвищує загальну швидкість обчислень в сучасних комп’ютерних системах.

Необов’язковий параметр “thread” в описі геометричного об’єкту призначений для задання саме кількості застосовуваних для розрахунку значення R-функції обчислювальних вузлів (або потоків). За замовченням значення цього параметру дорівнює одиниці.

Також слід зазначити, що поняття “назва-моделі” визначає ім’я функціонального блоку, що задає підсумкове значення функціонального виразу (своєрідна точка входу до програми за аналогією з функцією *main*() в мові програмування C [78]).

Таким чином, загальну структуру опису функціональної моделі геометричної області можна описати наступним чином.

```
// Опис функціональної моделі...
@geometric_model(Main_func, thread = 16)
{
	// Геометричний примітив...
	function R1(x, y, z)
	{
		variable R = 1

		return R^2 - x^2 - y^2 - z^2
	}
	// Геометричний примітив...
	function R2(x, y, z)
	{
		//...
	}
	//...
	// Підсумкова функція
	function Main_func(x, y, z)
```

```
        {
                //...
        }
}
```

3.2.5 Приклади опису геометричних областей із застосуванням FORL-G

Розглянемо приклади опису деяких геометричних об’єктів із застосуванням запропонованої мови FORL-G.

1. Геометрична область “пішак” (рис. 3.2).

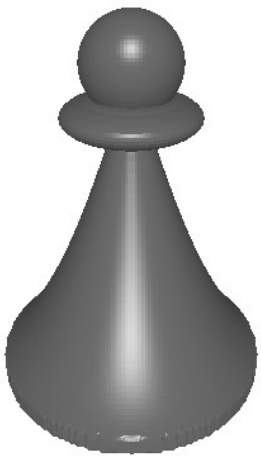

Рис. 3.2 – Геометрична область “пішак”

Цю фігуру можна описати із застосуванням R-функції такого виду:

$$W = W_1 \vee W_2, \tag{3.4}$$

де $W_1 = E_3 \wedge E_6$;

$$W_2 = E_4 \vee E_5;$$

$$E_3 = y(7-y);$$

$$E_4 = 1 - x^2 - z^2 - (7-y)^2;$$

$$E_5 = 2 - x^2 - z^2 - 9(6-y)^2;$$

$$E_6 = E_1 \wedge E_2;$$

$$E_1 = \frac{7}{16}\left(\sqrt{(x^2+z^2)} - 4\right)^2 - y;$$

$$E_2 = 9 - x^2 - z^2.$$

На мові FORL-G геометричний об’єкт, заданий із застосуванням співвідношення (3.4), можна описати таким чином.

```
// Геометрична область “Пішак”
@geometric_model(pawn, thread = 9)
{
	function e1(x, y, z)
	{
		return 7 / 16 * (sqr(sqrt(sqr(x) + sqr(z)) - 4.0)) - y
	}
	function e2(x, y, z)
	{
		return 9 - sqr(x) - sqr(z)
	}
	function e3(x, y, z)
	{
		return y * (7 - y)
	}
```

```
        function e4(x, y, z)
        {
                return 1.0 - sqr(x) - sqr(z) - sqr(7 - y)
        }
        function e5(x, y, z)
        {
                return 2.0 - sqr(x) - sqr(z) - 9 * sqr(6 - y)
        }
        function e6(x, y, z)
        {
                return e1(x, y, z) and e2(x, y, z)
        }
        function w1(x, y, z)
        {
                return e3(x, y, z) and e6(x, y, z)
        }
        function w2(x, y, z)
        {
                return e4(x, y, z) or e5(x, y, z)
        }
        function pawn(x, y, z)
        {
                return w1(x, y, z) or w2(x, y, z)
        }
}
```

Іншим більш компактним варіантом опису цієї моделі на мові FORL-G може бути, наприклад, такий скрипт.

```
@geometric_model(pawn, thread = 9)
{
    function pawn(x, y, z)
    {
        variable e1 = 7 / 16 * (sqr(sqrt(sqr(x) + sqr(z)) - 4)) - y,
            e2 = 9 - sqr(x) - sqr(z),
            e3 = y * (7 - y),
            e4 = 1 - sqr(x) - sqr(z) - sqr(7 - y),
            e5 = 2 - sqr(x) - sqr(z) - 9 * sqr(6 - y),
            e6 = e1 and e2,
            w1 = e6 and e3,
            w2 = e4 or e5;
            return w1 or w2;
    }
}
```

Проте застосування саме такого варіанту опису досить складно автоматично розпаралелити, оскільки реалізація запуску в окремих обчислювальних потоках чи вузлах частин громіздкого виразу є нетривіальною задачею.

2. Геометрична область “колінчастий вал” (рис. 3.3).

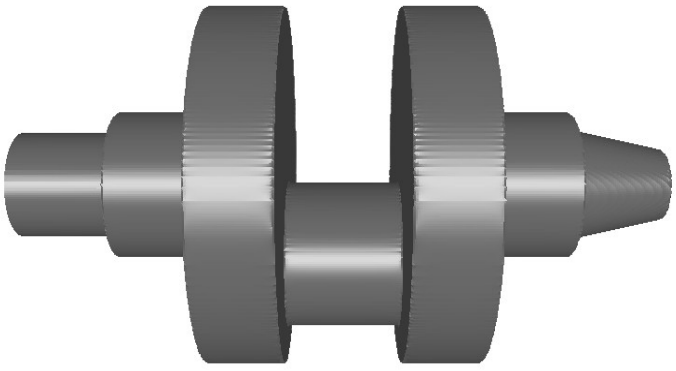

Рис. 3.3 – Геометрична область “колінчастий вал”

Такий досить складний геометричний об'єкт може бути описаний з використанням наступного співвідношення [79]:

$$F = F_1 \vee F_2 \vee F_3 \vee (F_4 \wedge \overline{F_5}) \vee F_7, \tag{3.5}$$

де $F_1 = \left(\frac{D_1}{2} - y^2 - z^2\right) \wedge \left(\frac{(l_5 - l_2)^2}{4} - x^2\right) \wedge \left(x^2 - \frac{(l_4 - l_3)^2}{4}\right)$ – права й ліва щоки;

$F_2 = \left(\frac{D_3}{2} - y^2 - z^2\right) \wedge \left(\frac{(l_6 - l_1)^2}{4} - x^2\right) \wedge \left(x^2 - \frac{(l_5 - l_3 + 0.5(l_3 - l_2))^2}{4}\right)$ – корінні шийки;

$F_3 = \left(\frac{D_2}{2} - y^2 - (z - cD)^2\right) \wedge \left(\frac{(l_6 - l_4 + 0.5(l_3 - l_2))^2}{4} - x^2\right)$ – шатунна шийка;

$F_3 = \left(\frac{D_2}{2} - y^2 - (z - cD)^2\right) \wedge \left(\frac{(l_6 - l_4 + 0.5(l_3 - l_2))^2}{4} - x^2\right)$ – праве кріплення для зняття навантаження;

$F_5 = \left(\frac{D_6}{2} - y^2 - z^2\right) \wedge \left(x - \frac{l_6 - l_1}{2} + l_7 - l_6 - l_8\right)$ – отвір у правому кріпленні;

$F_6 = \left(\frac{(x + dx)^2}{2} - \frac{4y^2 + 4z^2}{D_5^2}\right)$ – формула конічної поверхні лівого кріплення, в якому $h = \frac{l_1 \dot{D}_5}{D_4 - D_5}$, $dx = h + \frac{l_4 - l_3}{2} + l_4 - l_1$;

$F_7 = F_6 \wedge \left(x + l_3 + \frac{l_4 - l_3}{2}\right) \wedge \left(l_1 - x - \frac{l_2 + l_4}{2}\right)$ – ліве конічне кріплення для зняття навантаження;

D_1 – діаметр кожної щоки;

D_2 – діаметр шийки шатуна;

D_3 – діаметр корінних шийок;

cD – зміщення шатунної шийки відносно вісі корінних шийок.

На мові FORL-G модель цієї геометричної області можна, наприклад, описати таким чином.

```
@geometric_model(shaft)
{
    function shaft(x, y, z)
    {
        float R1 = 4.25, // Радіус кожної щоки валу
              R2 = 1.75, // Радіус шийки шатуну
              R3 = 1.75, // Радіус шийок колінчастого валу
              R4 = 1.25, // Великий радіус конічного кріплення
              R5 = 0.75, // Малий радіус конічного кріплення
              R6 = 1, // Радіус отвору правого кріплення
              R7 =1.25, // Радіус правого кріплення
              DC = 1.75,
              L1 = 2.5,
              L2 = 5,
              L3 = 7,
              L4 = 10,
              L5 = 12,
              L6 = 14.5,
              L7 = 17.25,
              L8 = 2,
        // Щоки колінчастого валу
        f1 = ((sqr(R1) - sqr(z) – sqr(y)) and (0.25 * sqr(L5 - L2) –
              sqr(x))) and (sqr(x) - 0.25 * sqr(L4 - L3)),
        // Шийки
```

```
            f2 = ((R3 * R3 - z * z - y * y) and (0.25 * sqr(L6 - L1) - x * x)) and
                (x * x - 0.25 * sqr(L5 - L3 + 0.5 * (L3 - L2))),
            f3 = (R2 * R2 - sqr(z - DC) - y * y) and (0.25 * sqr(L6 - L4 +
                0.5 * (L3 - L2))- x * x),
            // Праве кріплення
            f4 = ((R7 * R7 - z * z-y * y) and (0.25 * L7 * L7 - x * x)) and
                (x - (L5 - L4)),
            f5 = (R6 * R6 - z * z - y * y) and (x - (0.5 * (L6 - L1) +
                 (L7 - L6) - L8)),
            // Ліве кріплення
            h = L1 * R5 / (R4 - R5),
            dx = h + (L4 - L1) + 0.5 * (L4 - L3),
            f6 = sqr(x + dx) / (h * h) - z * z / (R5 * R5) - y * y / (R5 * R5),
            f7 = (f6 and (x + L3 + 0.5 * (L4 – L3))) and -
                (x + L3 - L1 - 0.5 * (L3 - L2) + 0.5 * (L4 - L3));
            return f1 or f2 or f3 or (f4 and not f5) or f7;
        }
    }
```

3.3 Алгоритмізація побудови дискретної моделі геометричної області, описаної із застосуванням мови FORL-G

Програмна реалізація функціонального підходу до геометричного моделювання ускладнюється тим фактом, що R-функція $F(x, y, z)$ є неявною. Для візуалізації або дискретизації на скінченні елементи функціонально заданого геометричного об’єкту в першу чергу необхідно побудувати певну апроксимацію

її поверхні, що вимагає розв'язання нетривіальної задачі пошуку координат деякої множини граничних вузлів (для яких виконується співвідношення $F(x, y, z)=0$).

Огляд послідовних алгоритмів пошуку точок на поверхні функціонально заданої області наведено в роботах [80, 81]. Для підвищення їх ефективності необхідно розробляти відповідні паралельні алгоритми. Найбільш простим для практичної реалізації є метод паралельної декомпозиції "Розділяй і володарюй" [82]. Згідно з ним вихідна задача пошуку множини точок, що належать границі Г неявно заданої геометричної області Ω, поділяється на певну сукупність підзадач, які розв'язуються паралельно, а отримані результати згодом об'єднуються.

Послідовний алгоритм пошуку границі області, заданої R-функцією, можна описати за допомогою псевдокоду [69] наступним чином:

Алгоритм *Процедура пошуку граничних вузлів геометричній області*
procedure *FindBnPts*(*BoundaryPoints*, *Box*, *N*)
BnPts – шуканий вектор координат точок на границі області
$Box=(X_{min}, Y_{min}, Z_{min}, X_{max}, Y_{max}, Z_{max})$ – координати кубоїду (зони пошуку)
$N=(N_x, N_y, N_z)$ – кількість кроків вздовж осей координат
begin
$H_x=(X_{max}-X_{min})/N_x$
$H_y=(Y_{max}-Y_{min})/N_y$
$H_z=(Z_{max}-Z_{min})/N_z$
while $i\in[0, N_x-1]$ **do**
$x_0=X_{min}+i\cdot H_x$
$x_1=X_{min}+(i+1)\cdot H_x$
while $j\in[0, N_y-1]$ **do**
$y_0=Y_{min}+j\cdot H_y$
$y_1=Y_{min}+(j+1)\cdot H_y$

while $k \in [0, N_k - 1]$ **do**

$z_0 = Z_{min} + k \cdot H_z$

$z_1 = Z_{min} + (k+1) \cdot H_z$

if $F(x_0, y_0, z_0) \le 0$ **and** $F(x_1, y_1, z_1) \ge 0$ **then**

$find(x_0, y_0, z_0, x_1, y_1, z_1) \rightarrow BnPts$

end if

end while

end while

end while

end procedure

Тут процедура *find*() реалізує пошук на відрізку $(x_0, y_0, z_0) - (x_1, y_1, z_1)$ координати точки, для якої R-функція, що описує вихідну геометричну область, приймає нульове значення.

Відповідну паралельну реалізацію вищенаведеного послідовного алгоритму можна отримати, якщо виконати розбиття вихідного кубоїду пошуку на певну кількість окремих підобластей, кількість яких вибирається в залежності від наявного числа процесорів в обчислювальній системі. Після чого результати роботи паралельно виконуваних алгоритмів об'єднати в один підсумковий масив граничних вузлів (рис. 3.4).

Після отримання множини вузлів, які належать границі вихідного геометричного об'єкту Ω, потрібно виконати первинну побудову апроксимації границі Γ. Ефективним способом вирішення цієї задачі є застосування алгоритму marching cubes [83]. Проте попередня апроксимація границі вихідної області, отримана із застосуванням цього алгоритму при невеликій кількості кроків, може бути достатньо неякісною (рис. 3.5).

Зменшення кроку призведе до покращення якості первинної апроксимації гранці області, проте, з одного боку, це може привести до небажаного збільшення

кількості граничних елементів (ГЕ), а, з іншого боку, він не гарантує коректну обробку особових точок (вершин кутів, границь отворів тощо) (рис. 3.6).

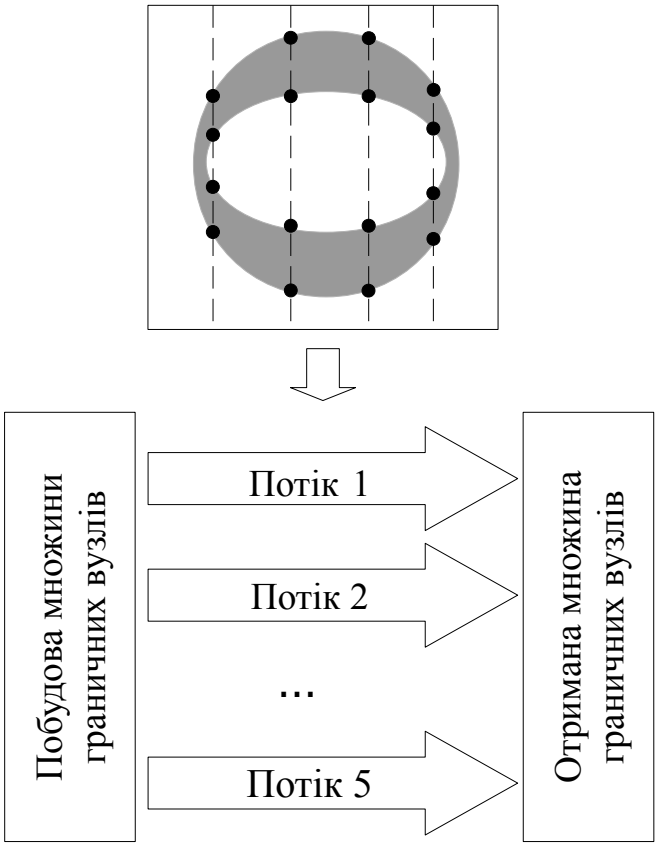

Рис. 3.4 – Застосування техніки “Розділяй і володарюй” при пошуку множини граничних вузлів

Таким чином, після побудови первинної апроксимації для створення якісної дискретної моделі вихідного геометричного об’єкту необхідно реалізувати певну процедуру оптимізації раніше отриманої апроксимації, яка фактично є лише першим наближенням до підсумкової скінченно-елементної моделі. Отримання якісної сітки для поверхні об’єкту дозволить потім відносно просто отримати й множину СЕ для внутрішньої частини вихідної геометричної області.

Огляд методів, що реалізують оптимізацію попереднього розбиття на СЕ або ГЕ наведено в [58]. В даній дисертаційній роботі для цього було реалізовано локальний алгоритм пошуку мінімуму функціоналу відстані-довжини. Результат

його застосування для оптимізації попередньої апроксимації границі області “колінчастий вал” (рис. 3.5) наведено на рис. 3.7.

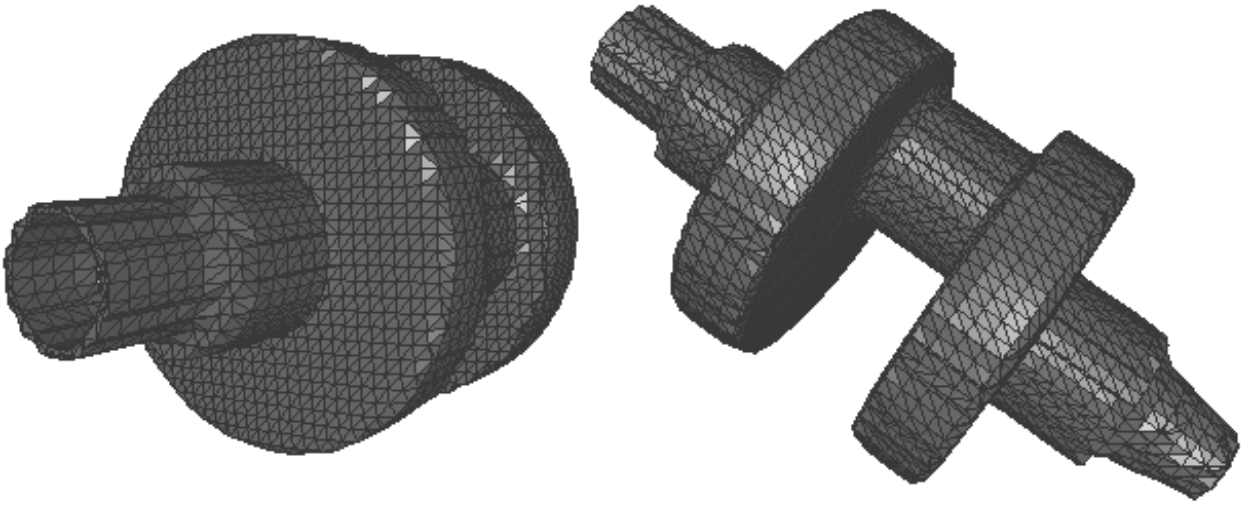

Рис. 3.5 – Попередня апроксимація границі області, отримана із застосуванням алгоритму marching cubes

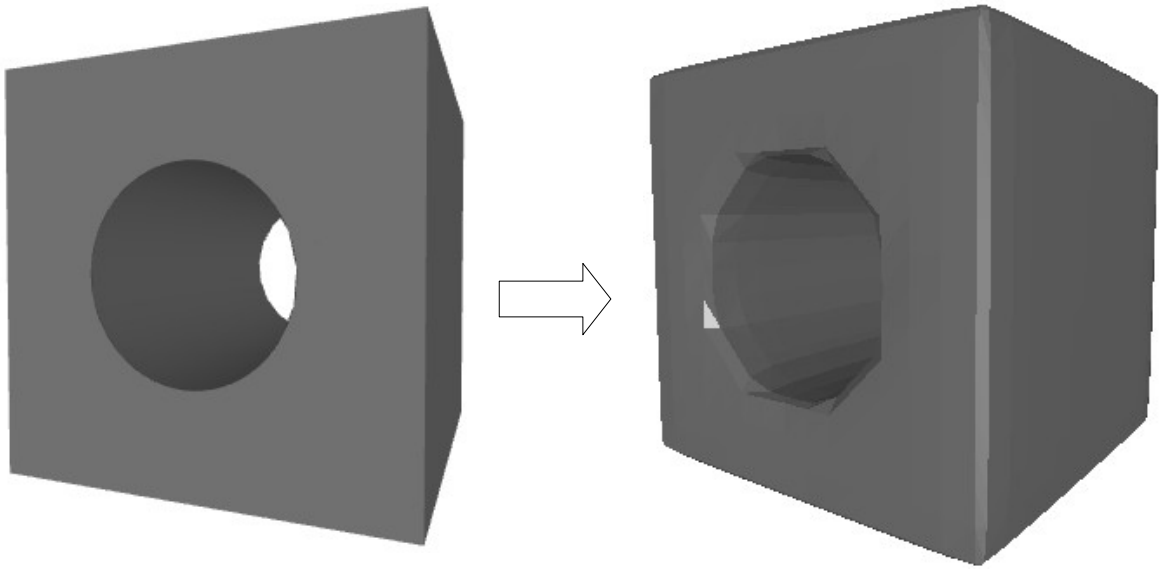

Рис. 3.6 – Приклад втрати особових точок границі вихідної області при невірно вибраному значенні кроку алгоритму marching cubes

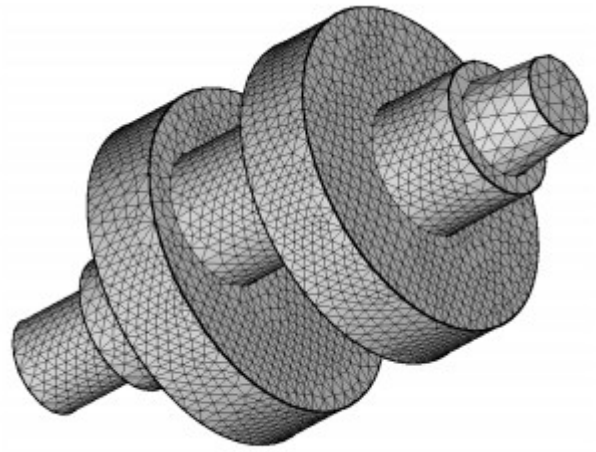

Рис. 3.7 – Оптимізована дискретна
модель границі геометричної області
“колінчастий вал”

Після отримання апроксимації границі вихідного геометричного об’єкта побудова його скінченно-елементної моделі може бути отримана, наприклад, із застосуванням фронтального алгоритму [84, 85]. Така стратегія на думку автора є найбільш ефективною при побудові дискретних моделей функціонально заданих об’єктів.

3.4 Застосуванням патерну Prototype для паралельної реалізації побудови дискретних моделей

Розглянемо застосування твірного патерну проєктування Prototype для програмної реалізації побудови дискретної скінченно-елементної моделі функціонально заданого вихідного геометричного об’єкту. Послідовність етапів побудови скінченно-елементної моделі можна представити таким чином (рис. 3.8).

Рис. 3.8 – Головні етапи побудови скінченно-елементної моделі функціонально-заданого геометричного об’єкту

Розглянемо приклад застосування шаблону Prototype для паралельної реалізації алгоритму побудови первинної апроксимації границі вихідного геометричного об’єкту. Ця операція є найбільш важливою, оскільки некоректно побудована границя області не дозволить створити якісну скінченно-елементну модель всієї вихідної геометричної області. При застосування шаблону Prototype відповідний програмний модуль потрібно спроєктувати таким чином, щоб при необхідності можна було виконати клонування об’єкту, що реалізує цей алгоритм для заданої підобласті на певному вузлі обчислювального кластера або процесорі (ядрі процесора), без прив’язки до його класу. UML-діаграму такої реалізації можна представити так, як це зображено на рис. 3.9.

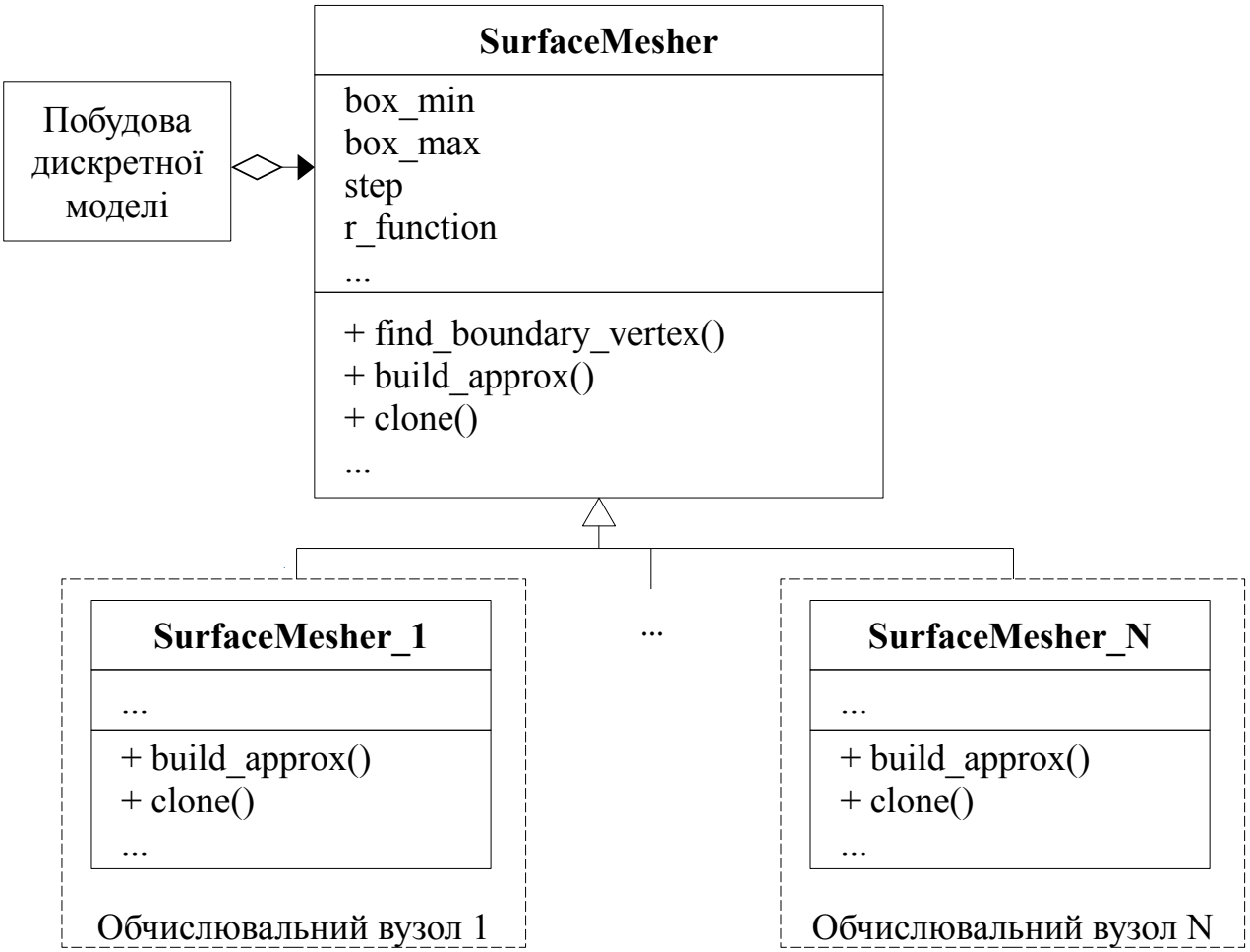

Рис. 3.9 – Реалізація клонування процедури побудови апроксимації границі із застосуванням патерну Prototype

Програмна реалізація на мові C++ [54] клонування об’єктів для запуску розрахунків в паралельних обчислювальних потоках на мультипроцесорній архітектурі може мати наступний типовий вигляд.

```
// ...
int nx{ui->le_nx->text().toInt()},
    ny{ui->le_ny->text().toInt()},
    nz{ui->le_nz->text().toInt()},
    num_threads{ui->le_threads->text().toInt()},
    step{nx / num_threads};
vector<std::thread> thr(num_threads);
```

```
array<float, 3> box_min{ui->le_minx->text().toFloat(),
  ui->le_miny->text().toFloat(),

  ui->le_minz->text().toFloat()},
            box_max{ui->le_maxx->text().toFloat(),
  ui->le_maxy->text().toFloat(),
  ui->le_maxz->text().toFloat()};
vector<Triangle> triangles;
MarchingCubes mc;

mc.set_box(box_min, box_max);
mc.set_n({float(nx), float(ny), float(nz)});
mc.set_function(get_r_function());

// ...
// Запуск процедури побудови апроксимації поверхні
// в паралельних обчислювальних потоках
for (int i = 0; i < num_threads; i++)
   thr[i] = std::thread(i == 0 ? mc : *mc.clone(),
         i * step, (i == num_threads - 1) ? nx :
         (i + 1) * step, ref(triangles));

for_each (thr.begin(), thr.end(), [](auto& tr) { tr.join(); });
// ...
```

Опис класу, що реалізує алгоритм marching cubes для побудови первинної апроксимації границі геометричної області, спроєктованого із застосуванням патерну проєктування Prototype, наведено в Додатку Б.

Аналогічним чином реалізуються й інші етапи побудови скінченно-елементної моделі вихідного функціонально заданого геометричного об’єкту.

3.5 Обчислювальний експеримент

Для верифікації запропонованого підходу було виконано наступний обчислювальний експеримент. В якості прикладу було розглянуто геометричну область “Чашка” [86], яка задається наступним співвідношенням:

$$F(x,y,z)=F_1(x,y,z)\vee F_2(x,y,z),$$

де

$F_1(x,y,z)=F_3(x,y,z)\wedge F_4(x,y,z);$

$F_2(x,y,z)=F_5(x,y,z)\vee F_6(x,y,z);$

$F_3(x,y,z)=a\sqrt{(x+a)^2+(y+b)^2}-(x+a)^2-(y+b)^2-z^2-b;$

$F_4(x,y,z)=-x-b;$

$F_5(x,y,z)=F_7(x,y,z)\wedge F_8(x,y,z);$

$F_6(x,y,z)=2b\sqrt{x^2+z^2}-x^2-y^2-z^2-2a;$

$F_7(x,y,z)=F_9(x,y,z)\wedge F_{10}(x,y,z);$

$F_8(x,y,z)=-y;$

$F_9(x,y,z)=1-\frac{\left(x^2+z^2\right)^2}{c}-\frac{y^4}{16c};$

$F_{10}(x,y,z)=\frac{\left(x^2+z^2\right)^2}{4c}+\frac{y^4}{5c}-1;$

$a=4;\ b=3;\ c=25.$

На рис. 3.10 представлено попереднє граничне подання цієї геометричної області, яке було отримане із застосуванням вищенаведеного паралельного алгоритму.

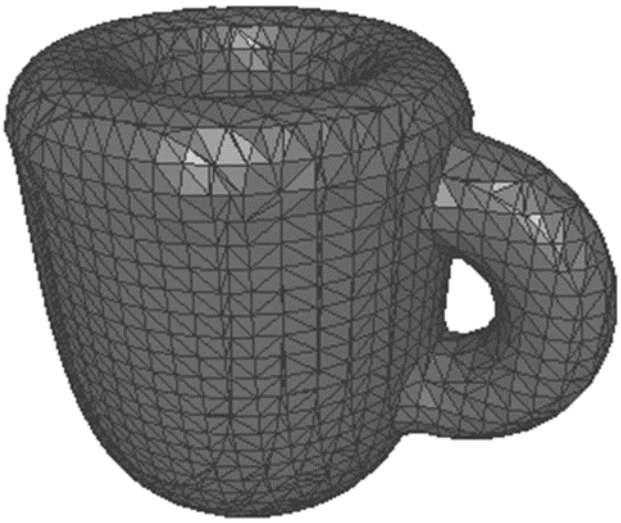

Рис. 3.10 – Первинне граничне подання
геометричної області “Чашка”

Час роботи алгоритму в залежності від кількості вживаних обчислювальних потоків наведено на рис. 3.11. Обчислювальний експеримент проводився на комп’ютері з процесором AMD Ryzen 7 2700X Eight-Core Processor з тактовою частотою 3.70 Гц і об’ємом оперативної пам’яті 32 Гбайт під управлінням операційної системи Windows 10. Для пошуку границі області будувалася сітка, що складалася з 100×100×100 вузлів.

З наведеного графіку видно, що час роботи паралельного алгоритму логарифмічно зменшується зі зростанням числа залучених до обчислень потоків до того моменту, коли їх кількість не стане дорівнювати кількості фізичних ядер процесора (AMD Ryzen 7 облаштовано вісьмома фізичними ядрами). Після чого швидкість роботи практично не змінюється, що пояснюється зростанням накладних витрати планувальника операційної системи на перемикання управління між потоками.

Після оптимізації отриманого граничного подання (тріангульованої поверхні вихідної геометричної області) і застосування фронтального алгоритму дискретизації, отримаємо підсумкову дискретну скінченно-елементну модель геометричного об’єкту “Чашка” (рис. 3.12).

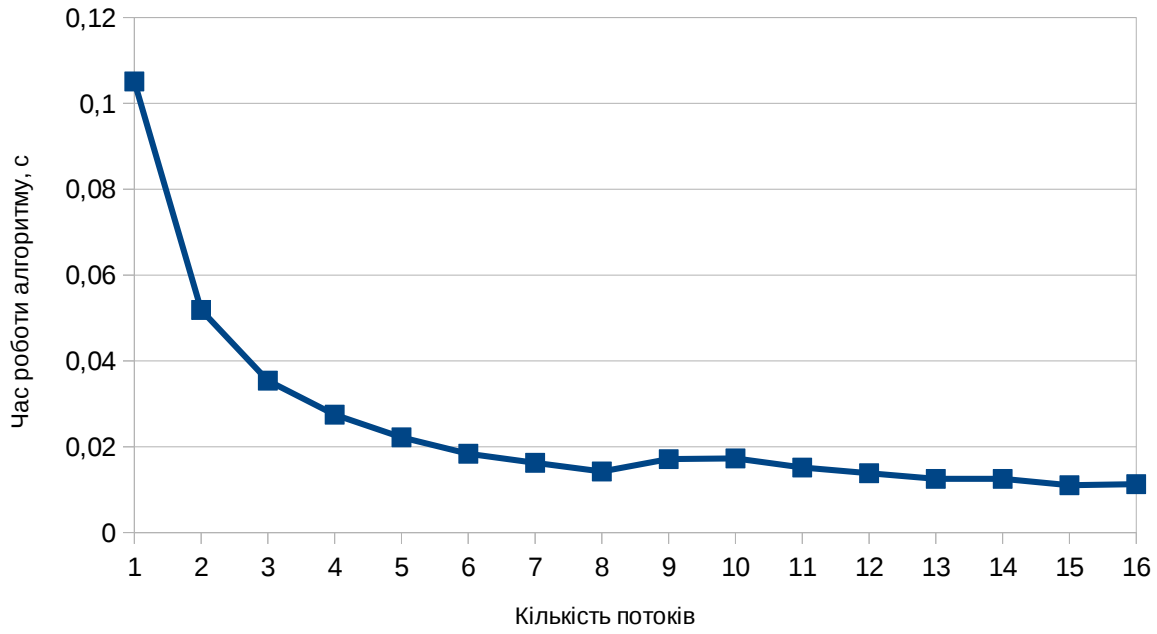

Рис. 3.11 – Час виконання паралельного алгоритму побудови геометричної області “Чашка” в залежності від кількості потоків

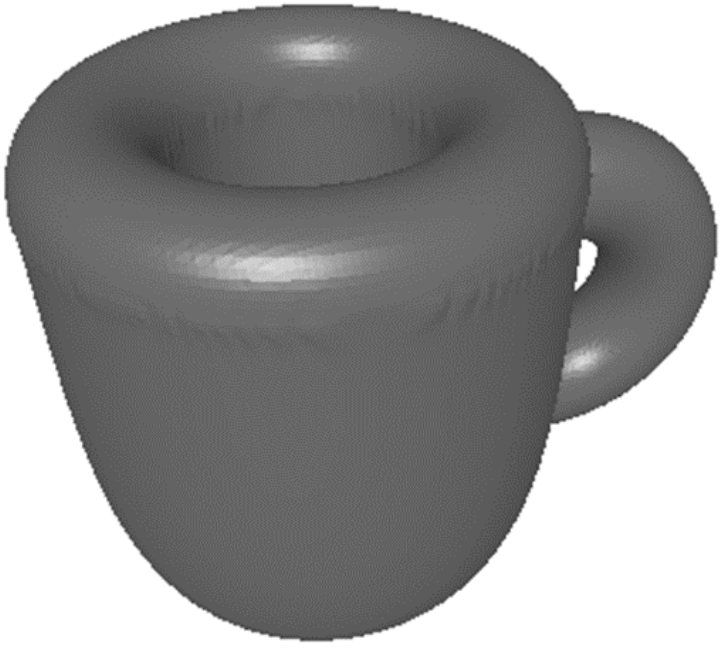

Рис. 3.12 – Дискретна модель R-функції, що описує геометричну область “Чашка”

Вона складається з 52333 вузлів і 179906 скінченних елементів у формі лінійного чотирьох-вузлового тетраедра.

Для перевірки можливості застосування запропонованого підходу на мультикомп'ютерах алгоритм запускався в цій же обчислювальній системі із застосуванням технології MPI. Графік залежності часу розрахунку від кількості вживаних обчислювальних вузлів (віртуальних процесорів) наведено на рис. 3.13.

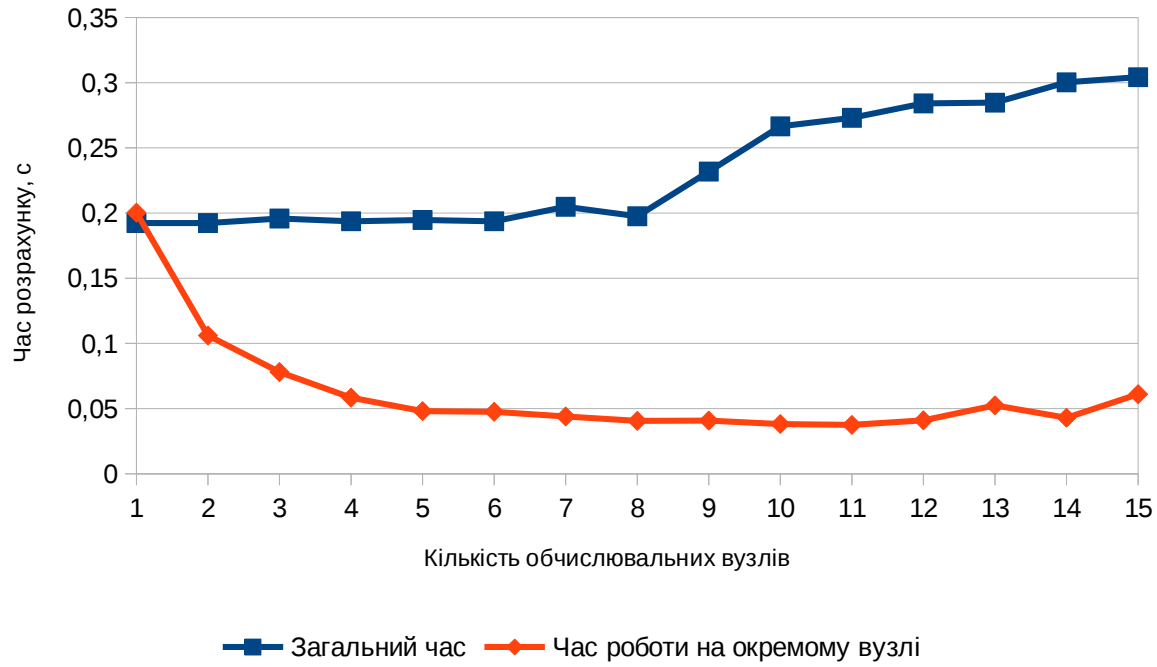

Рис. 3.13 – Час побудови дискретної моделі геометричної області "Чашка" із застосуванням технології MPI в залежності від кількості обчислювальних вузлів

Як і у попередньому випадку загальний час роботи алгоритму залежить від кількості наявних фізичних обчислювальних вузлів. При збільшенні їх числа час роботи програми також збільшується, що пояснюється накладними витратами на обмін даними по мережі та їх синхронізацію. Проте, час роботи алгоритму на кожному окремому вузлі логарифмічно зменшується.

Висновки до розділу 3

У третьому розділі наведено опис запропонованої проблемно-орієнтованої мови геометричного моделювання FORL-G, яка дозволяє із застосуванням функціонального підходу описувати дво- та тривимірні геометричні області будь-якої форми. В FORL-G наявні спеціалізовані засоби, для використання можливостей сучасних паралельних систем при геометричному моделюванні й дискретизації областей складної форми.

В третьому розділі також розглянуто застосування патерну проєктування Prototype для паралельної реалізації препроцесору системи скінченно-елементного аналізу, що дозволяє уніфікувати програмний код для застосування в паралельних обчислювальних системах різних типів: мультипроцесорах та мультикомп’ютерах.

Запропонований підхід було реалізовано програмною Для його верифікації було виконано обчислювальний експеримент, який підтвердив його ефективність.

Основні наукові і практичні результати третього розділу опубліковано в роботах [49, 50, 87-93].

4 РЕАЛІЗАЦІЯ СКІНЧЕННО-ЕЛЕМЕНТНОГО РОЗРАХУНКУ В ПАРАЛЕЛЬНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМАХ

4.1 Автоматизація виведення співвідношень для скінченно-елементних розрахунків

Основні рівняння математичної фізики, що використовуються при розв'язанні задач в статиці, можуть бути безпосередньо виведені з варіаційного принципу д'Аламбера-Лагранжа [94]:

$$\delta(\Pi - A) = 0\,, \tag{4.1}$$

де $\Pi = \int\limits_{\Omega} \sum\limits_{i,j} \sigma_{ij}\varepsilon_{ij}\, d\Omega$ – кінетична енергія механічної системи;

$A = \int\limits_{\Omega} \sum\limits_{i} X_i u_i\, d\Omega + \int\limits_{\Gamma} \sum\limits_{i} \overline{X}_i u_i\, d\Gamma$ – робота зовнішніх та внутрішніх сил;

σ_{ij} – компоненти тензору напружень;

ε_{ij} – компоненти тензору деформацій;

u_i – компоненти вектору переміщень;

X_i, $\overline{X}_i$ – компоненти векторів зовнішніх та внутрішніх навантажень (відповідно);

Ω – область, яку займає механічна система;

Γ – границя Ω.

Таким чином, співвідношення (4.1) для цього енергетичного функціоналу можна переписати у наступній формі:

$$\delta\Big(\int\limits_{\Omega} \sum\limits_{i,j} \sigma_{ij}\varepsilon_{ij}\, d\Omega - \int\limits_{\Omega} \sum\limits_{i} X_i u_i\, d\Omega - \int\limits_{\Gamma} \sum\limits_{i} \overline{X}_i u_i\, d\Gamma\Big) = 0\,. \tag{4.2}$$

Для подальшого виведення необхідних для розв'язання задачі співвідношень необхідно мати формули, які визначають наступні залежності:

$$\sigma_{ij}=\sigma_{ij}(\varepsilon_{ij}), \tag{4.3}$$

$$\varepsilon_{ij}=\varepsilon_{ij}(u_i), \tag{4.4}$$

$$u_i=u_i(x_1,x_2,x_3). \tag{4.5}$$

Тут формула (4.3) визначається законом Гуку, а (4.4) – співвідношенням Коші [95]. Таким чином, для здійснення подальшого виведення залежностей, необхідних для розв'язання певної задачі математичної фізики, необхідно мати формулу для співвідношення (4.5), що визначає переміщення і залежить від обраної для розв'язання конкретної задачі гіпотези.

В МСЕ в якості (4.5) застосовується наступна апроксимація полів переміщень у вузлах СЕ:

$$u_i(x_{1,}x_{2,}x_3)=\sum_{k=1}^{n}u_{i,k}N_k(x_{1,}x_{2,}x_3), \tag{4.6}$$

де $u_{i,k}$ – невідомі значення переміщень функції $u_i(x_{1,}x_{2,}x_3)$ у k-му вузлі СЕ, що підлягають визначенню;

$N_k(x_1,x_2,x_3)$ – функції форми, що залежать від типу СЕ [2, 96],

n – кількість вузлів СЕ.

Підставляючи (4.6) у (4.4) та (4.3), отримаємо наступні вирази для компонент тензорів деформацій та напружень:

$$\varepsilon_{ij}=\sum_{k=1}^{n}u_{i,k}\frac{\partial N_k}{\partial x_i}+(1-\delta_{ij})u_{j,k}\frac{\partial N_k}{\partial x_j}, \tag{4.7}$$

$$\sigma_{ij}=K\left(\sum_{k=1}^{n}u_{i,k}\frac{\partial N_k}{\partial x_i}+(1-\delta_{ij})u_{j,k}\frac{\partial N_k}{\partial x_j}\right)+\lambda\left(\sum_{m=1}^{3}\sum_{k=1}^{n}u_{m,k}\frac{\partial N_k}{\partial x_m}\right),\ i=j\,, \tag{4.8}$$

$$\sigma_{ij}=G\left(\sum_{k=1}^{n}u_{i,k}\frac{\partial N_k}{\partial x_i}+u_{j,k}\frac{\partial N_k}{\partial x_j}\right),\ i\neq j\,, \tag{4.9}$$

де δ_{ij} – символ Кронекера;

K, G, λ – числові параметри, що залежать від пружних характеристик механічної системи.

Підставляючи співвідношення (4.7)-(4.9) в (4.2) та виконуючи інтегрування, отримаємо наступний вираз:

$$\delta\left(\alpha_1 u_1^2+\alpha_2 u_1 u_2+\alpha_3 u_1 u_3+\alpha_4 u_2^2+\ldots+\alpha_q u_2 u_3\right)=0\,, \tag{4.10}$$

де α_l – числові коефіцієнти, які в загальному випадку залежать від координат СЕ та пружних параметрів механічної системи.

Для розв’язання цієї варіаційної задачі можна, наприклад, скористатися методом варіації довільної сталої [97]. Тоді, враховуючи граничні умови, розв’язання вихідної задача (4.1) зведеться до пошуку коренів СЛАР наступного виду:

$$[\boldsymbol{K}]\{\boldsymbol{u}\}=\{\boldsymbol{f}\}\,, \tag{4.11}$$

де $[\boldsymbol{K}]$ – симетрична розріджена матриця коефіцієнтів СЛАР, яку прийнято називати матрицею жорсткості;

$\{\boldsymbol{u}\}$ – вектор шуканих вузлових переміщень;

$\{\boldsymbol{f}\}$ – вектор правої частини СЛАР, який зазвичай визначає вузлові навантаження, що діють на механічну систему.

Аналогічним чином, із застосуванням всього трьох операцій: підстановки, диференціювання та інтегрування можна автоматично виводити розрахункові співвідношення для різних функціоналів, наприклад, Гамільтона-Остроградського в задачах динаміки [98].

Таким чином, наведений вище спосіб виведення розрахункових співвідношень для МСЕ не залежить від розмірності задачі та її типу (статика або динаміка), що дозволяє одноманітно виводити необхідні співвідношення для різних задач. Отже, поклавши його в основу методу формалізації опису постановок задач математичної фізики і чисельних схем їх розрахунку, можна отримати достатньо універсальний підхід до розв'язання задач різних типів.

Очевидно, що цей підхід потребує:

1) розробки формального способу опису варіаційних принципів і інших необхідних співвідношень у зручній для обробки із застосуванням комп'ютера формі;

2) створення паралельних алгоритмів, які реалізують виведення розрахункових формул на основі заданих користувачем співвідношень.

4.2 Проблемно-орієнтована мова опису математичних моделей FORL-F

Для запису варіаційних формул і правил виведення з них всіх необхідних розрахункових співвідношень, що застосовуються в МСЕ, було розроблено спеціалізовану проблемно-орієнтовану мову FORL-F (Formula Language – Functional), яка, як і FORL-G, також підтримує засоби паралельної обробки.

В наступних пунктах цього параграфу за допомогою нотації РФБН буде формально описано синтаксис та семантику мови FORL-F, а також приклади її застосування для опису схем розрахунку різних задач.

4.2.1 Основні символи мови FORL-F

Основні символи мови FORL-F за допомогою РФБН визначаються аналогічно тому, як це робилося для мови FORL-G. Іншим є опис спеціальних символів та зарезервованих слів, які мають такий вигляд.

Спеціальний-символ = “+” | “-” | “*” | “/” | “^” | “(“ | “)” | “,” | “=” | “{” | “}” | “<” | “>” | “!” | “@” | зарезервоване-слово

зарезервоване-слово = “abs” | “acos” | “and” | “asin” | “atan” | “atan2” | “constant” | “cos” | “cosh” | “diff” | “function” | “functional” | “functional_model” | “load” | “object” | “exp” | “not” | “or” | “result” | “return” | “sin” | “sinh” | “surface_integral” | “tan” | “tanh” | “var” | “volume_integral”

Як і в мові FORL-G, семантика FORL-F не дозволяє застосовувати в якості ідентифікаторів зарезервовані слова.

4.2.2 Визначення типів даних в FORL-F та допустимих операцій над ними

Типи даних у мові FORL-F визначаються таким чином.

тип-даних = числовий-тип-даних | векторний-тип-даних | матричний-тип-даних

“Числовий-тип-даних” у мові FORL-F визначаються так само, як і в мові FORL-G. Це певна підмножина множини дійсних чисел, яка залежить від конкретної реалізації або комп’ютерної платформи. Так, наприклад, в платформі Windows 10 x64 діапазон можливих значень типу double мови програмування C++ становить 1.7E +/- 308 (15 чисел).

Під поняттям “векторний-тип-даних” розуміється одновимірний масив (вектор) дійсних чисел, які є коефіцієнтами функції форми або результатом певних операцій над ними.

Тип “матричний-тип-даних” визначає двовимірний масив (матрицю) дійсних чисел, яка утворюється в результаті виконання операції варіювання (var) над двома векторами. Як правило, цей тип даних визначає локальну матрицю жорсткості або її певну частину.

Змінні різних типів в FORL-F задаються наступним чином.

декларація-змінної = декларація-результатів | декларація-констант | декларація-функцій | декларація-навантажень | декларація-функціоналів

декларація-результатів = “result” ідентифікатор {, ідентифікатор}

декларація-констант = “constant” ідентифікатор [= ініціалізуючий-вираз] {, ідентифікатор [= ініціалізуючий-вираз]}

декларація-функцій = “function” ідентифікатор [= ініціалізуючий-вираз] {, ідентифікатор [= ініціалізуючий-вираз]}

декларація-навантажень = “load” ідентифікатор [= ініціалізуючий-вираз] {, ідентифікатор [= ініціалізуючий-вираз]}

декларація-функціоналів = “functional” ідентифікатор [= ініціалізуючий-вираз] {, ідентифікатор [= ініціалізуючий-вираз]}

ініціалізуючий-вираз = вираз

Поняття виразу буде формально введено нижче. Таким чином, при декларації змінної їй можна присвоювати певне початкове значення. Виключенням тут є декларація змінних, які належать до типу даних “result”. Вони визначають шукані функції (наприклад, компоненти вектору переміщень при реалізації МСЕ в формі методу переміщень).

В мові FORL-F реалізована наступна ієрархія типів даних (від молодшого до старшого):

1) числовий-тип-даних (constant, load);
2) векторний-тип-даних (result, function);

3) матричний-тип-даних (functional).

Перетворення типів даних в FORL-F можливе лише від молодшого типу даних до старшого. У відповідності до цієї ієрархії реалізовані наступні правила роботи з типами:

– всі арифметичні операції над змінними, що належать до типу даних “числовий-тип-даних”, в якості результату утворюють величину, яка також належить до цього ж типу даних;

– добуток або ділення змінної, що належить до типу даних “векторний-тип-даних” або “матричний-тип-даних” на змінну, яка належить до типу “числовий-тип-даних”, в результаті дає “векторний-тип-даних” або “матричний-тип-даних” (відповідно);

– додавання або віднімання однотипних змінних в якості результату дає цей же самий тип даних

– операндами операції варіювання можуть бути лише змінні, що належать до типу даних “векторний-тип-даних”; результатом є змінна, що належить до векторний-тип-даних;

– операція диференціювання (diff) допускається тільки над змінними, що належать до типу “векторний-тип-даних”; її результатом є також цей тип даних.

Інші варіанти операцій над типами даних мови FORL-F не допускаються.

4.2.3 Вирази в FORL-F

Поняття виразу в мові FORL-F вводиться наступним чином.

вираз = [знак] елемент-виразу {оператор [знак] елемент-виразу}

елемент-виразу = константа | змінна | вбудована-функція | вираз

оператор = арифметичний-оператор | варіаційний-оператор

арифметичний-оператор = “+” | “-” | “*” | “/” | “^”

варіаційний-оператор = “var”

Константа в FORL-F визначається так само, як це робилося в мові геометричного моделювання FORL-G.

змінна = ідентифікатор

вбудована-функція = елементарна-функція | часткова-похідна | поверхневий-інтеграл | об’ємний-інтеграл

Перелік наявних в FORL-F вбудованих елементарних функцій такий самий, як і в мові FORL-G (табл. 3.1).

Функція, що реалізує часткову похідну, визначається наступним чином.

часткова-похідна = [знак]“diff”(вираз, ідентифікатор)

Перший параметр цієї функції визначає вираз, який підлягає диференціюванню (від обов’язково повинен належати до векторного типу), а другий – ідентифікатор координатної або часової змінної, за якою відбувається взяття похідної.

Функції, що виконують інтегрування по об’єму або поверхні СЕ, визначаються наступним чином.

поверхневий-інтеграл = [знак]“surface_integral”(вираз)

обёэмний-інтеграл = [знак]“volume_integral”(вираз)

Вирази, які є параметрами цих функцій, обов’язково повинні належати до матричного типу.

Окремим типом виразів є логічні вирази. Вони застосовуються для визначення множини вузлів, які задовольняють крайовим умовам. Логічні вирази визначаються так.

логічний-оператор = вираз логічний-оператор вираз {логічна-операція логічний вираз}

логічний-оператор = “>” | “>=” | “<” | “<=” | “==” | “!=”

логічна-операція = “and” | “not” | “or”

Тоді граничні умови можна визначити в мові FORL-F наступним чином.

гранична-умова = ідентифікатор ”(“ логічний-оператор ”)“ “=” арифметичний-вираз

В якості ідентифікатора тут може застосовуватися лише змінна, яка задекларована в секції “result”.

Аналогічним чином вводиться поняття зосередженого навантаження.

зосереджене-навантаження = ідентифікатор ”(“ логічний-оператор ”)“ “=” арифметичний-вираз

Тут в якості ідентифікатора за семантикою мови FORL-F може застосовуватися лише змінна, яка задекларована в секції “load”.

Приклади запису формул на мові FORL-F.

```
// …
constant E = 1.0E+6, m = 0.3, K, G, R
result u, v
load X = 1.0E+5, Y = 0
function Exx, Eyy, Exy, Sxx, Syy, Sxy
functional W, A

// Арифметичні формули
R = 10
K = E / (1 – m * m)
G = E / (2 + 2 * m)

Exx = diff(u, x)
Eyy = diff(v, y)
Exy = diff(u, y) + diff(v, x)
Sxx = K * (Exx + m * Eyy)
Syy = K * (m * Exx + Eyy)
Sxy = G * Exy
```

```
// Варіаційні формули
W = 0.5 * volume_integral(Sxx var Exx + Syy var Eyy + Sxy var Exy)
A = volume_integral(X var u + Y var v)
// ...
// Граничні умови та зосереджені навантаження
u(x ^ 2 + y ^ 2 == R ^ 2) = 0
Y(x >= 10) = -1000
// …
```

4.2.4 Структура опису схеми розрахунку задачі на мові FORL-F

Опис математичної моделі і схеми розрахунку задачі математичної фізики із застосуванням мови FORL-F у відповідності до РФБН виглядає так.

опис-математичної-моделі = “@functional_model” “(“ назва-моделі [, “thread =” ціле-без-знака] “)” початок-блоку варіаційний-блок {, варіаційний-блок} кінець-блоку

Обов’язковий параметр “назва-моделі” визначається таким же самим чином, як і у мові FORL-G.

варіаційний-блок = “object” ім’я-об’єкту “(“ дискретна-модель список-аргументів “)” початок-блоку декларація-результатів [декларація-констант] [декларація-навантажень] [декларація-функцій] [декларація-функціоналів] варіаційний-результат кінець-блоку

ім’я-об’єкту = ідентифікатор

дискретна-модель = mesh-файл

Поняття “mesh-файл” в РФБН не визначене, оскільки воно задає назву файлу, який містить інформацію про скінченно-елементу модель об’єкту, що

розраховується, і залежить від прийнятих в поточній файловій системі правил напису назв файлів.

Початок та кінець блоку, а також список аргументів визначаються так само, як і у FORL-G.

варіаційний-результат = “return” варіаційний-вираз

варіаційний-вираз = вираз

Згідно з семантикою мови FORL-F тип виразу, який повертається оператором “return”, обов’язково повинен бути матричним.

Варіаційний-блок може складатися з наступних частин:

1) заголовка;
2) декларації шуканих функцій (обов’язковий блок);
3) декларації констант;
4) декларації допоміжних функцій;
5) декларації компонент вектору навантажень;
6) декларації варіаційних співвідношень (функціоналів);
7) опису констант;
8) опису допоміжних функцій;
9) опису компонент вектору навантажень;
10) опису варіаційних формул (функціоналів);
11) опису крайових умов;
12) результатного виразу, який визначає підсумкову варіаційну формулу, мінімізація якої приведе до розв’язання вихідної задачі.

Заголовок варіаційного блоку визначає його назву, розмірність та скінченно-елементу модель, із застосуванням якої виконуються розрахунки.

Блок декларації шуканих функцій задає список ідентифікаторів, які визначають назву функцій, значення яких буде розраховуватися як результат мінімізації функціоналу. Ці змінні не можуть бути проініціалізовані і є обов’язковими.

Блоки опису констант, навантажень, допоміжних функцій та функціоналів є необов’язковим. Вони містить певну сукупність операторів присвоювання, в яких допоміжним змінним задаються певні значення.

Обов’язковим в FORL-F є блок, який визначає підсумкову варіаційну формулу. Він утворюється ключовим словом “return”, після якого йде варіаційна формула, мінімізація якої повинна привести до розв’язання вихідної задачі.

Таким чином, по аналогії з FORL-G структура опису схеми розрахунку задачі із застосуванням мові FORL- F має модульну структуру, де кожен модуль описує певний об’єкт. Сукупність об’єктів утворюють механічну систему.

Як і в FORL-G необов’язковий параметр “thread” в описі задачі дозволяє вказати кількість вживаних обчислювальних вузлів чи потоків. За замовченням значення цього параметру дорівнює одиниці.

Загальну структуру опису математичної моделі вихідної задачі можна представити так.

```
// Опис схеми розрахунку задачі
@functional_model(actual_problem, thread = 16)
{
        // Одновимірний об’єкт для розрахунку
        object rod(rod.mesh, x)
        {
                result u
                // ...
                return volume_integral(1.0E+6 * diff(u, x) var diff(u, x))
        }
        // ...
}
```

4.2.5 Приклади опису задач на мові FORL-F

Розглянемо декілька прикладів опису математичних моделей і схем розрахунку задач із застосуванням мови FORL-F.

1. Розтягнення стрижня.

Розглянемо одновимірну задачу про пошук параметрів напружено-деформованого стану стрижня, один кінець якого жорстко затиснений, а до іншого прикладена сила, що діє вздовж його вісі (рис. 4.1).

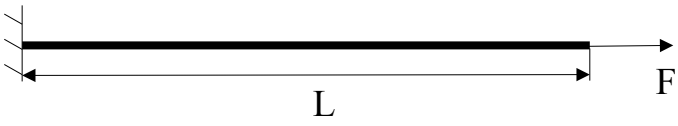

Рис. 4.1 – Задача про розтягнення стрижня

Її можна описати за допомогою мови FORL-F наступним чином.

```
// Задача про розтягнення стрижня
@functional_model(problem_1d)
{
    // Опис моделі стрижня і схеми його розрахунку
    object rod(rod.mesh, x)
    {
        result u
        constant E = 203200, L = 10, F = 1
        function Exx, Sxx
        load X
        functional W

        // Співвідношення Коші
```

```
        Exx = diff(u, x)
        // Закон Гуку
        Sxx = E * Exx
        // Варіаційний принцип Лагранжу
        W = 0.5 * volume_integral(Sxx var Exx)
        // Граничні умови
        u(x == 0) = 0
        // Зосереджене навантаження
        X(x == L) = F
        return W
    }
}
```

2. Прогин балки під дією поверхневого навантаження.

Розглянемо задачу про прогин балки з жорстко затисненими краями, на верхню грань якої діє рівномірно розподілене поверхневе навантаження (рис. 4.2).

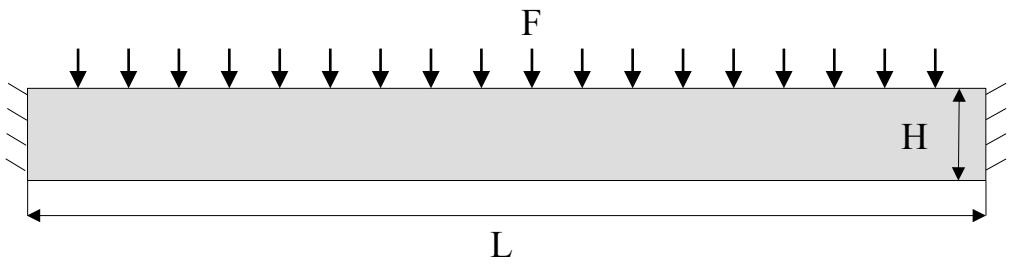

Рис. 4.2 – Задача про згин балки під дією рівномірно розподіленого поверхневого навантаження

На мові FORL-F цю двовимірну задачу можна описати наступним чином.

```
// Задача про згин балки
@functional_model(problem_2d, thread = 8)
```

```
{
    // Опис моделі балки і схеми її розрахунку
    object beam(beam.mesh, x, y)
    {
        result u, v
        constant E = 203200, m = 0.27, K = E / (1 - m * m),
        constant G = E / (2 + 2 * m), L = 10, H = 0.5, F = 100
        function Exx, Eyy, Exy, Sxx, Syy, Sxy
        load X, Y
        functional W, A

        Exx = diff(u, x)
        Eyy = diff(v, y)
        Exy = diff(u, y) + diff(v, x)

        Sxx = K * (Exx + m * Eyy)
        Syy = K * (m * Exx + Eyy)
        Sxy = G * Exy

        X = 0
        Y(y == H / 2) = -F
        W = 0.5 * volume_integral(Sxx var Exx + Syy var Eyy +
            Sxy var Exy)
        A = surface_integral(X var u + Y var v)

        u(x == 0) = 0
        v(x == 0) = 0
        u(x == L) = 0
        v(x == L) = 0
```

```
        return W - A
    }
}
```

3. Деформація об’єкту під дією власної ваги.

Розглянемо задачу про пошук параметрів напружено-деформованого стану об’єкту “Чашка” (рис. 3.12), що знаходиться під дією власної ваги. Цю задачу можна повністю описати із застосуванням мов FORL-G (геометрична модель) і FORL-F (математична модель) наступним чином.

```
// Функціональна модель геометричного об’єкту “Чашка”
@geometric_model(cup)
{
    function cup(x, y, z)
    {
        variable r10 = (x ^ 2 + z ^ 2) ^2 / 16 + y ^ 4 / 1280 - 1,
            r9 = 1 – (x ^ 2 + z ^ 2) ^ 2 / 256 – y ^ 4 / 4096,
            r8 = -y,
            r7 = r9 and r10,
            r6 = 6 * sqrt(x ^ 2 + z ^ 2) – (x ^ 2 + y ^ 2 + z ^ 2) - 8,
            r5 = r7 and r8,
            r4 = -x - 3,
            r3 = 4 * sqrt((y + 3) ^ 2 + (x+4) ^ 2) – (x+4) ^ 2 – z ^ 2 –
                    (y+3) ^ 2 - 3,
            r2 = r5 or r6,
            r1 = r3 and r4;
        return r1 or r2;
    }
}
```

```
// Обчислювальна схема розрахунку об’єкту “Чашка”
@functional_model(problem_3d)
{
	// Опис моделі задачі
	object cup(cup.mesh, x, y, z)
	{
		constant E = 203200, m = 0.27, G = E / (2 + 2 * m),
		constant L = 2 * m * G / (1 - 2 * m)
		function Exx, Eyy, Ezz, Exy, Exz, Eyz, Sxx, Syy, Szz, Sxy, Sxz, Syz
		load X = 0, Y = -100, Z = 0
		functional W, A

		Exx = diff(u, x)
		Eyy = diff(v, y)
		Ezz = diff(w, z)
		Exy = diff(u, y) + diff(v, x)
		Exz = diff(u, z) + diff(w, x)
		Eyz = diff(v, z) + diff(w, y)

		Sxx = 2 * G * Exx + L * (Exx + Eyy + Ezz)
		Syy = 2 * G * Eyy + L * (Exx + Eyy + Ezz)
		Szz = 2 * G * Ezz + L * (Exx + Eyy + Ezz)
		Sxy = G * Exy
		Sxz = G * Exz
		Syz = G * Eyz

		W = 0.5 * volume_integral(Sxx var Exx + Syy var Eyy +
			Szz var Ezz + Sxy var Exy + Sxz var Exz + Syz var Eyz)
		A = volume_integral(X var u + Y var v + Z var w)
```

```
            u(z < -7.9) = 0
            v(z < -7.9) = 0
            w(z < -7.9) = 0
            return W - A
        }
}
```

4.3 Паралельні алгоритми розрахунку задач, описаних на мові FORL-F

Чисельний аналіз задач математичної фізики у статичній постановці із застосуванням МСЕ зазвичай складається з наступних етапів [2, 96]:

1) генерація ЛМЖ для кожного СЕ та ансамблювання їх до глобальної матриці жорсткості;

2) розрахунок правої частини СЛАР (вектора-стовпця навантажень);

3) врахування граничних умов;

4) розв’язання отриманої СЛАР;

5) розрахунок додаткових вузлових результатів (наприклад, деформацій та напружень на основі раніше отриманих переміщень).

Найбільш важливим та складним в реалізації є перший етап, оскільки він залежить від типу СЕ, що застосовується, варіаційного принципу, прийнятих гіпотез тощо. В більшості сучасних систем скінченно-елементного аналізу побудова ЛМЖ здійснюється із застосуванням спеціальних шаблонів, які розташовані у бібліотеках типових СЕ (рис. 4.3). Чим більше типів СЕ (оболонки, пластини, контактна взаємодія тощо) є в бібліотеці, тим більше класів задач може розв’язувати певна система. Такий підхід має ряд переваг (наприклад, високу швидкість розрахунків), але головним його недоліком, очевидно, є обмеженість

можливостей розв’язання задач наявними типами СЕ в бібліотеці. Поповнення ж її новими типами зазвичай є досить складною задачею.

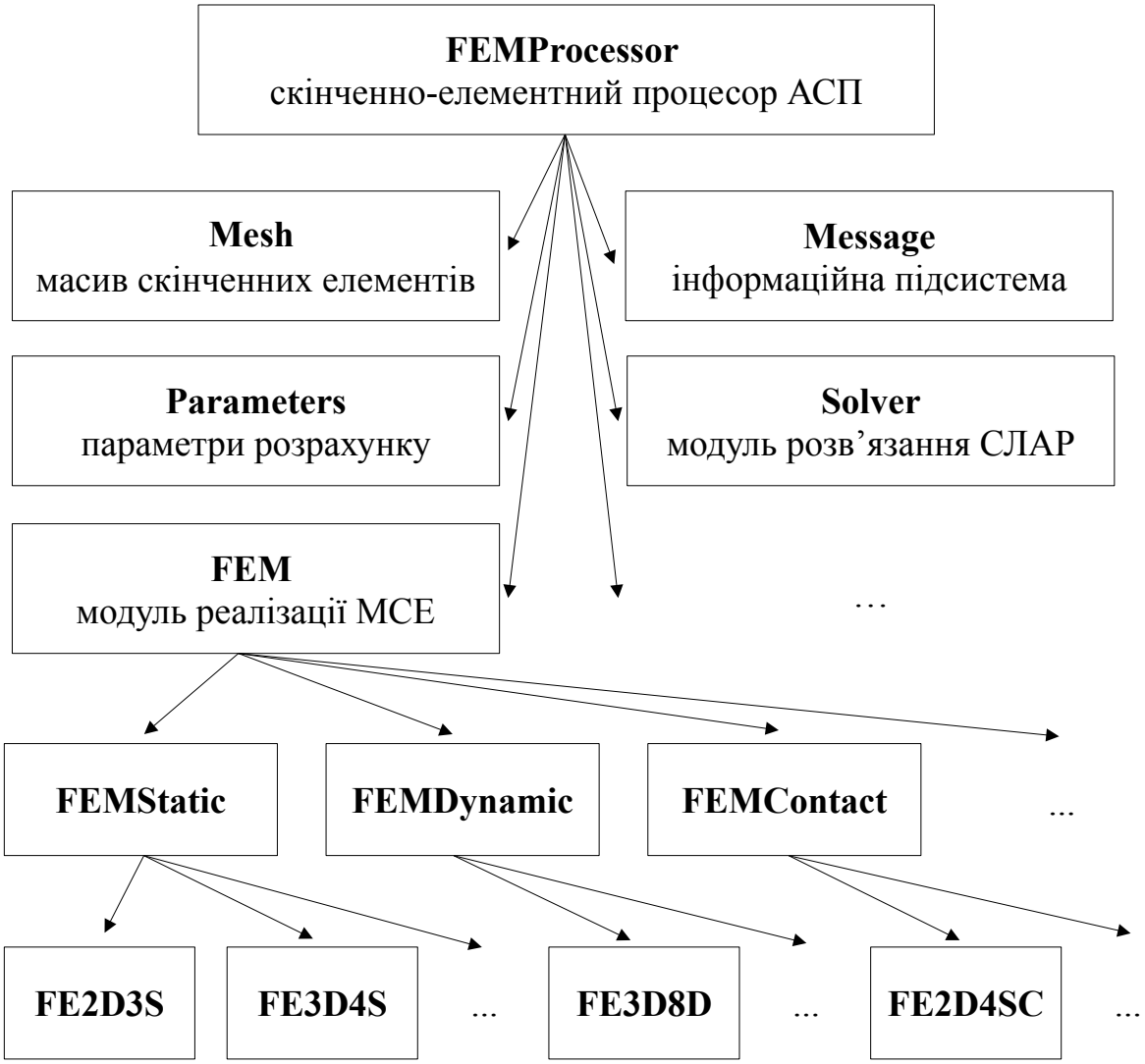

Рис. 4.3 – Типова структура скінченно-елементного процесору

Тому для усунення цього недоліку в даній роботі застосовується підхід, який базується на безпосередньому розрахунку коефіцієнтів ЛМЖ на основі співвідношень, описаних мовою FORL-F. Розглянемо алгоритм побудови ЛМЖ із застосуванням співвідношень, що описані на цій предметно-орієнтованій мові.

Нехай функція форми СЕ описується наступним співвідношенням:

$$N_i(x,y,z)=c_{i,1}+c_{i,2}x+c_{i,3}y+c_{i,4}z+\dots, \qquad (4.12)$$

де $c_{i,j} \in \mathbb{R}$, $i = \overline{1,n}$, $j = 1,2,\ldots$ – коефіцієнти, що залежать від координат елементу.

Тоді апроксимацію функцій – компонентів вектору переміщень на СЕ можна представити наступною структурою даних (рис. 4.4).

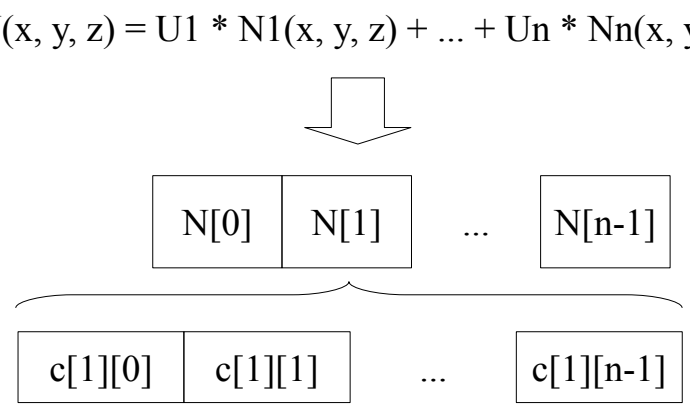

Рис. 4.4 – Структура даних для збереження інформації про апроксимацію функції

Таким чином, інформація про будь-яку функцію в трансляторі мови FORL-F зберігається у вигляді вектору певних числових коефіцієнтів (наприклад, коефіцієнтів відповідних функцій форм для шуканих функцій). Очевидно, що реалізація операцій додавання (віднімання) не викликає труднощів. Проте, пряме виконання операцій диференціювання або множення вже повністю залежить від виду функції форми (тобто типу СЕ). В даній роботі для розрахунку довільного виразу (4.1) пропонується наступний алгоритм.

При обчисленні виразу (4.2) застосовуються операнди – функціональні об’єкти, які можуть примати наступні значення:

1) дійсне число;
2) вектор коефіцієнтів функцій форм (рис. 4.4);
3) вектор дійсних чисел;
4) матриця дійсних чисел.

Ясно, що всі константи, які беруть участь в розрахунках (наприклад, пружні параметри), описуються дійсними числами.

Шукані функції (ідентифікатори, що задекларовані за допомогою оператора “result” мови FORL-F) описуються із застосуванням векторів коефіцієнтів функцій форм. Люба операція над такими об’єктами виконується за такою схемою: якщо це операція диференціювання, то для кожного об’єкта N[i], i = 0, 1, ..., n - 1 спочатку викликається вбудований метод *diff*(), який обчислює відповідну часткову похідну, а потім розраховується значення отриманої похідної функції (яке поки що зберігається у вигляді, представленому на рис. 4.4) у поточному вузлі інтегрування, після чого результат зберігається вже у вигляді вектору дійсних чисел.

Таким чином, векторами дійсних чисел на кожному етапі розрахунку виразу (4.2) представлені всі змінні, які описані за допомогою директиви “function” мови FORL-F, а також “result” (після виконання будь-яких операцій над ними).

Результатом виконання операції варіювання завжди є матриця дійсних чисел. В залежності від типів аргументів ця операція може бути бінарною, наприклад, $\sigma_{ij}\,\delta\varepsilon_{ij}$ (коли обидва операнди є векторами), або унарною – $X_i\,\delta u_i$ (коли операнд зліва – дійсне число, а справа – вектор). Формально, при виконанні унарного варіювання результатом є вектор-стовпець, але з точки зору програмної реалізації його також можна вважати матрицею.

Розглянемо наступний приклад. Нехай потрібно обчислити значення виразу

$$\int\limits_{\Omega_k} \sigma_{xy}\,\varepsilon_{xy}\,d\,\Omega_k\,, \tag{4.13}$$

де Ω_k k-й СЕ.

Згідно із співвідношеннями Коші та законом Гуку компоненти тензору деформацій ε_{xy} та напружень σ_{xy} можуть бути виражені через переміщення наступним чином:

$$\varepsilon_{xy}=\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}=u_y+v_x\,, \tag{4.14}$$

$$\sigma_{xy}=G\,\varepsilon_{xy}=G\left(u_y+v_x\right). \tag{4.15}$$

Підставляючи (4.14) і (4.15) у (4.13), отримаємо таке співвідношення:

$$G\int\limits_{\Omega_k}\left(u_y^2+2u_y v_x+v_x^2\right)d\Omega_k\,. \tag{4.16}$$

Розрахунок формули (4.16) в загальному вигляді цілком залежить від способу обчислення похідних u_y та v_x, тобто від типу СЕ, обраного способу апроксимації переміщень і таке інше. Проте, якщо розрахунок інтегралу здійснювати чисельно, то для обчислення (4.16) достатньо програмно реалізувати лише процедуру розрахунку значення підінтегральної функції в кожному вузлі інтегрування. Обчислення похідної від функції форми (4.12), яка зберігається за допомогою структури даних, наведеної на рис. 4.4, є нескладною процедурою і легко реалізується у відповідному класі, що інкапсулює функції форми певного типу СЕ. Таким способом можна уніфіковано реалізувати обчислення виразу (4.2) будь-якої складності.

Із застосуванням сучасного стандарту C++17 [99] мови програмування C++ структуру даних (клас), яка одночасно може описувати числове значення, два типи векторів і матрицю можна реалізувати таким чином.

```
#include <variant>
// ...
using Scalar = double;
using Vector = variant<vector<T>, vector<double>>;
using Matrix = matrix<double>;
variant<Scalar, Vector, Matrix> val;
// ...
```

Тут під узагальненим типом “T” розуміється клас, що описує функцію форми певного СЕ (і відповідно реалізує обчислення часткових похідних).

Тоді, обчислення суми функціональних об’єктів за допомогою псевдокоду можна описати наступним чином.

Алгоритм *Сума функціональних об’єктів*
procedure *Sum*(*lhs*, *rhs*)
lhs – функціональний об’єкт зліва
rhs – функціональний об’єкт справа
begin
res – результуючий функціональний об’єкт
if is_scalar(*lhs*) **and** is_scalar(*rhs*) **then**
res = *lhs* + *rhs*
else
if is_vector(*lhs*) and is_vector(*rhs*) **then**
while $i \in [0, n-1]$ **do**
$res_i = lhs_i + rhs_i$
end do
end if
else
if is_matrix(*lhs*) and is_matrix(*rhs*) **then**
while $i \in [0, n-1]$ **do**
while $j \in [0, n-1]$ **do**
$res_{ij} = lhs_{ij} + rhs_{ij}$
end do
end do
end if
else

type_error()

end if

Sum = *res*

end procedure

Таким чином, допускається сумування тільки однотипних функціональних об’єктів. Аналогічним чином описується і процедура їх віднімання.

Алгоритм обчислення добутку функціональних об’єктів більш складний. Його можна описати так.

Алгоритм *Добуток функціональних об’єктів*

procedure *Mul*(*lhs*, *rhs*)

lhs – функціональний об’єкт зліва

rhs – функціональний об’єкт справа

begin

res – результуючий функціональний об’єкт

if is_double(*lhs*) **then**

if is_double(*rhs*) **then**

$ret = lhs \cdot rhs$

else

if is_vector(*rhs*) **then**

while $i \in [0, n-1]$ **do**

$ret_i = lhs \cdot rhs_i$

end do

end if

else

if is_matrix(*rhs*) **then**

while $i \in [0, n-1]$ **do**

while $j \in [0, n-1]$ **do**

$ret_{ij} = lhs \cdot rhs_{ij}$

end do

end do

end if

end if

else

if is_scalar(*rhs*) **then**

if is_scalar(*lhs*) **then**

$ret = lhs \cdot rhs$

endif

else

if is_vector(*lhs*) **then**

while $i \in [0, n-1]$ **do**

$ret_i = lhs_i \cdot rhs$

end do

end if

else

while $i \in [0, n-1]$ **do**

while $j \in [0, n-1]$ **do**

$ret_{ij} = lhs_{ij} \cdot rhs$

end do

end do

end if

else

type_error()

end if

Mul = *ret*

end procedure

Аналогічним чином реалізується процедура ділення функціональних об’єктів.

Процедура пошуку варіації функціональних об’єктів може бути описана таким чином.

Алгоритм *Варіювання функціональних об’єктів*

procedure *Var*(*lhs*, *rhs*)

lhs – функціональний об’єкт зліва

rhs – функціональний об’єкт справа

begin

res – результуючий функціональний об’єкт

if is_vector(*lhs*) **and** is_vector(*rhs*) **then**

while $i \in [0, n-1]$ **do**

while $j \in [0, n-1]$ **do**

$ret_{ij} = lhs_i \cdot rhs_j + lhs_j \cdot rhs_i$

end do

end do

else

if is_scalar(*lhs*) **and** is_vector(*rhs*) **then**

while $i \in [0, n-1]$ **do**

$ret_{i\,n-1} = lhs \cdot rhs_i$

end do

end if

else

type_error()

end if

Var = ret

end procedure

Повний опис класу, що реалізує збереження функціональних об’єктів і операції над ними, приведено в Додатку В. Приклад опису функції форми для СЕ у формі лінійного тетраедра приведено в Додатку Г.

Такий підхід до обчислення виразу (4.2) на відміну від способу, запропонованого в [39], дозволяє розраховувати варіаційні співвідношення будь якої складності, оскільки значення функціонального об’єкту перед безпосереднім застосуванням обчислюється у поточному вузлі інтегрування, що дозволяє уникнути втрати членів апроксимаційних поліномів (переповнення), яке виникає при, наприклад, декількох операціях множення функцій.

Загальна реалізація розрахунку задачі, описаної на мові FORL-F, із застосуванням МСЕ складається з наступних етапів:

1) трансляція вихідного тексту схеми розрахунку на FORL-F і формування відповідного байт-коду;

2) розрахунок ЛМЖ і вектору-стовпцю навантажень для кожного СЕ і формування глобальних матриць жорсткості і вектора-стовпця правої частини СЛАР;

3) врахування граничних умов і зосереджених навантажень;

4) розв’язання СЛАР;

5) розрахунок допоміжних результатів.

На мові C++ цей алгоритм можна описати наступним чином.

```
// ...
// Узагальнений клас, що реалізує транслятор FORL-F
TParser<T> parser;
// Вектор результатів
vector<double> res;

// Трансляція схеми розрахунку у внутрішній байт-код
parser.set_program(program);
// Налаштування
```

```
// ...
// Формування СЛАР
create_global_matrix(parser);
// Врахування граничних умов
use_boundary_condition(parser);
// Розв’язання СЛАР
if (solve_equations(res))
{
  calc_results(parser, res);
  save_result(prog_name.substr(0, prog_name.find_last_of("."))
      + ".res");
  print_result_summary();
}
// ...
```

Розглянемо застосування патерну проєктування Prototype для реалізації паралельної побудови СЛАР. Наприклад, процедура формування глобальної матриці жорсткості в обчислювальній системі зі спільною пам’яттю може бути реалізована таким чином.

```
// Формування глобальної матриці жорсткості
template <typename T> void create_global_matrix(TParser<T> &parser)
{
    int step = (int)mesh.get_fe().size1() / numThread;
    vector<thread> thr(numThread);

    progress.set_process(Message::GeneratingMatrix, 1, (int)mesh.get_fe().size1());

    // Створення заданої кількості обчислювальних потоків
    for (int i = 0; i < numThread; i++)
      thr[i] = thread(&TFEM<S>::create_global_matrix_thread<T>, this, ref(parser),
            i * step, (i == numThread - 1) ? (int)mesh.get_fe().size1() : (i + 1) * step);
```

```
    for_each (thr.begin(), thr.end(), [](auto &tr) { tr.join(); });
    progress.stop_process();
}


// Розрахунок частини глобальной матрицы жесткости на певному
// обчислювальному вузлі
template <typename T> void create_global_matrix_thread(TParser<T> &p,
                                int begin, int end)
{
    TParser<T> *parser = p.clone(); // Клонування транслятора формул

    // Розрахунок ЛМЖ для заданого діапазону СЕ
    for (auto i = begin; i < end; i++)
    {
       progress.add_progress();
       parser->set_data(mesh.get_shape<T>(i));
       ansamble_local_matrix(parser->run(mesh.get_coord_fe(i)).asMatrix(), i);
    }
}
```

Аналогічним чином реалізуються паралельні версії алгоритмів, що виконують врахування граничних умов, розв’язання СЛАР тощо.

4.4 Обчислювальний експеримент

Для ілюстрації ефективності запропонованого підходу розглянемо задачу про пошук параметрів напружено-деформованого стану (переміщень, деформацій та напружень) об’єкту “Чашка” (рис. 3.12), який знаходиться під дією власної ваги.

Задача розв’язувалася при наступних безрозмірних пружних параметрах: $E=203200$, $\nu=0.27$. Об’ємне навантаження, яке діє вздовж осі ординат – $Y=100$. Граничні умови: $u|_{y<-7.9}=0$, $v|_{y<-7.9}=0$, $w|_{y<-7.9}=0$. Повна схема чисельного розрахунку цієї задачі на мові FORL-F наведена в пункті 4.2.5.

Для обчислювального експерименту застосовувалася комп’ютерна система зі спільною пам’яттю (мультипроцесор) з наступними характеристиками: багатоядерний процесор (4 фізичних двоконвеєрних ядра) – Intel(R) Core(TM) i7 CPU 930 @ 2.80GHz з тактовою частотою 2.80 Гц; об’єм оперативної пам’яті – 8 Гбайт; операційна система Windows 10 x64.

З метою дослідження залежності часу роботи запропонованого паралельного алгоритму від кількості застосованих обчислювальних потоків була виконана серія розрахунків.

На рис. 4.5 наведено скриншот результату роботи програми при застосуванні одного обчислювального потоку. Ця програмна реалізація була здійснена у вигляді консольного додатку (без графічного інтерфейсу користувача) із застосуванням стандартного компілятора мови програмування C++ фірми Microsoft.

Залежність часу розв’язання задачі від кількості застосованих обчислювальних вузлів наведено на рис. 4.6.

Аналіз наведеного графіку показує, що час роботи паралельного алгоритму розрахунку задачі із застосуванням запропонованого підходу, як і у випадку побудови дискретної моделі, логарифмічно зменшується до того моменту, коли кількість застосованих потоків не буде дорівнювати кількості фізичних ядер процесора (враховуючи, що один потік застосовується для виконання головної процедури програми). Далі швидкість роботи алгоритму фактично не змінюється, що також пояснюється зростанням накладних витрати планувальника операційної системи на обслуговування потоків.

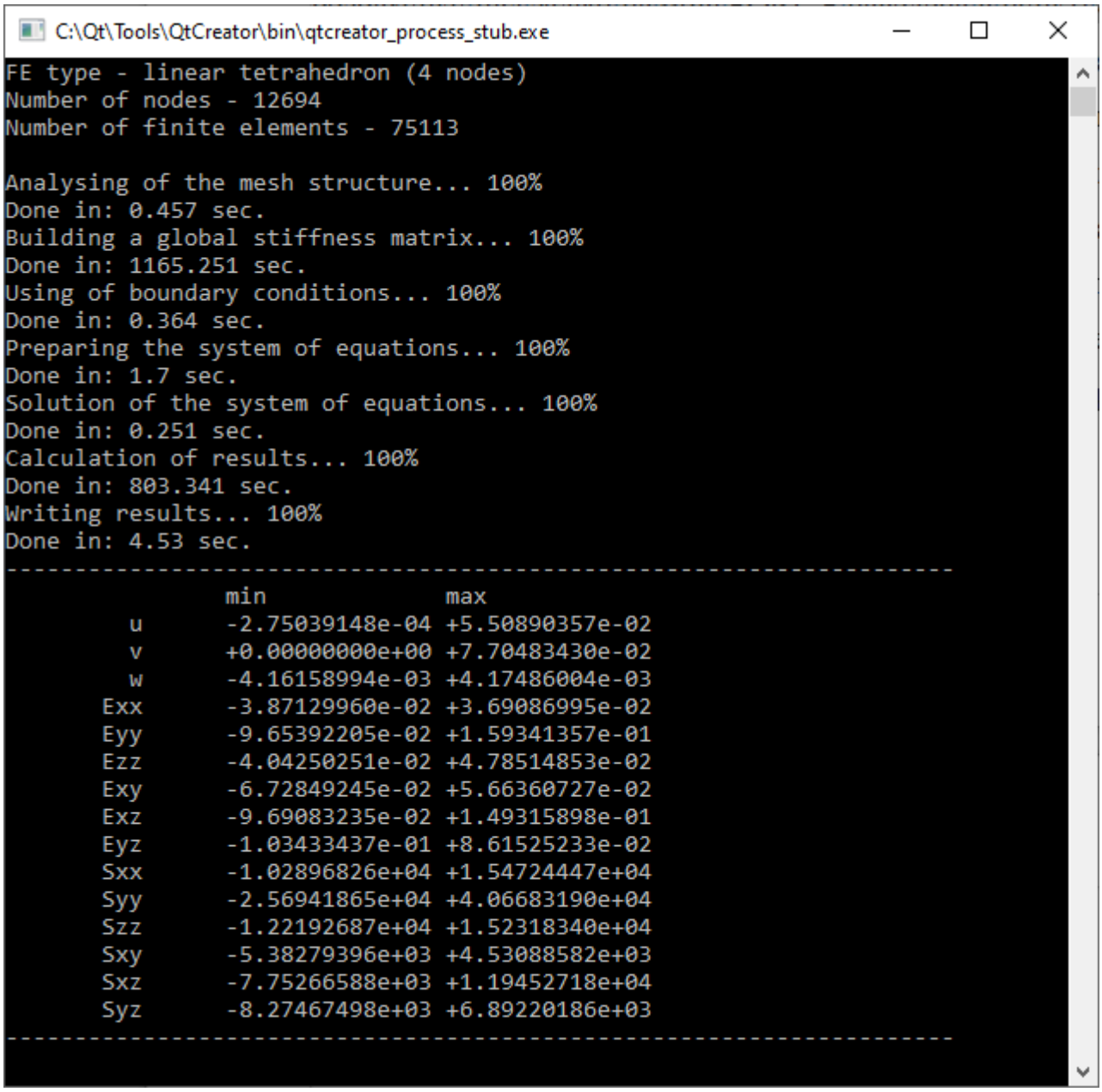

Рис. 4.5 – Результати розрахунку об’єкту “Чашка” при застосування одного обчислювального потоку

Виконання цих же розрахунків на шістнадцятиядерному процесорі AMD Ryzen 7 2700X Eight-Core Processor з тактовою частотою 3.70 Гц приводить до наступних результатів (рис. 4.7).

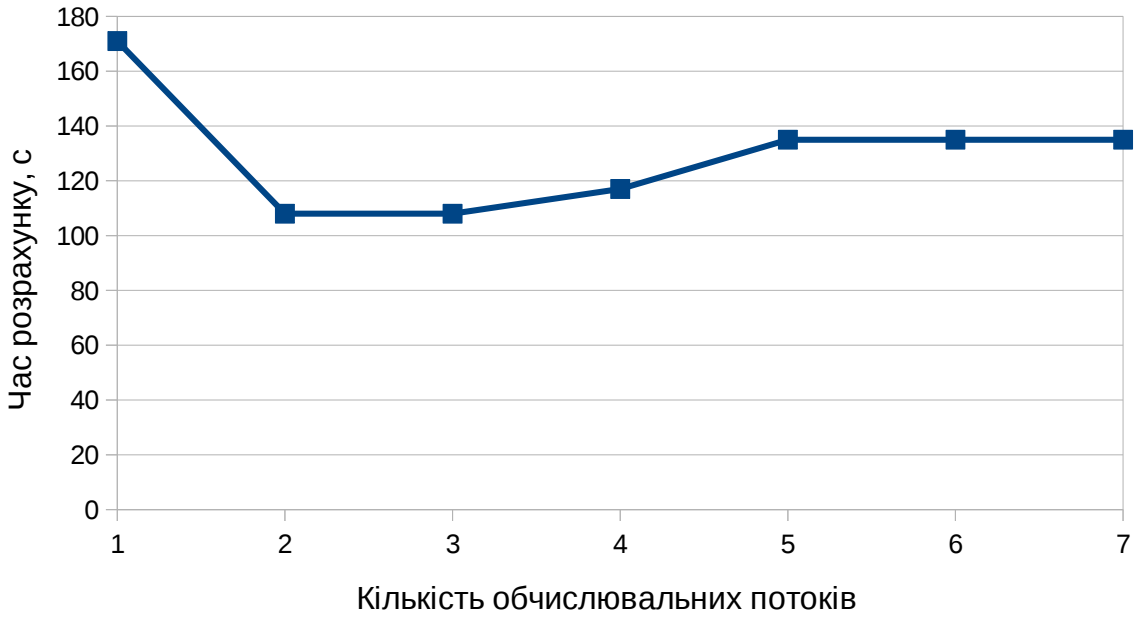

Рис. 4.6 – Залежність часу розрахунку об’єкту “Чашка” від кількості вживаних обчислювальних вузлів на 8-ядерному процесорі Intel Core i7

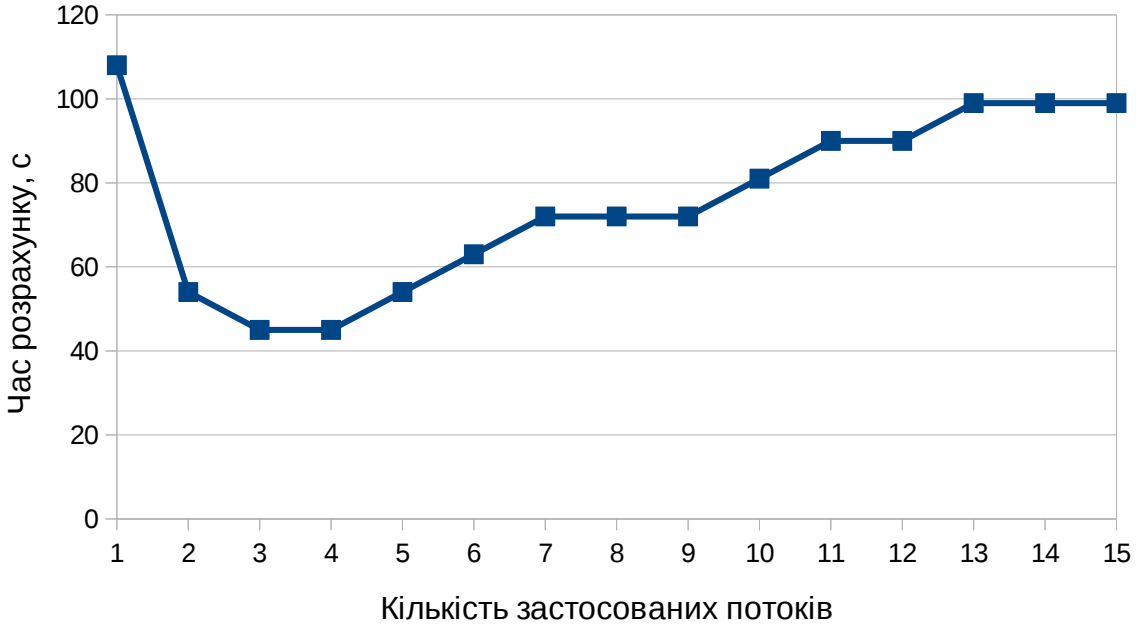

Рис. 4.7 – Залежність часу розрахунку об’єкту “Чашка” від кількості обчислювальних потоків на 16-ти ядерному процесорі AMD Ryzen 7

Порівняння цих графіків показує, що збільшення кількості потоків (обчислювальних вузлів), що використовуються при розрахунках, не завжди приводить до отримання істотного приросту у швидкості обчислень. Тому, очевидно, що в кожному конкретному випадку при розв’язанні складної задачі потрібно проводити окремі дослідження щодо необхідної кількості залучених обчислювальних міцностей.

Висновки до розділу 4

У четвертому розділі наведено опис розробленої мови опису схем чисельного розрахунку задач математичної фізики FORL-F, яка дозволяє користувачеві описувати варіаційні принципи механіки і розрахункові співвідношення для чисельного розв’язання одно-, двох- та тривимірних задач в статичній постановці. Мова FORL-F, як і FORL-G, підтримує спеціалізовані засоби, які дозволяють виконувати розрахунки в паралельних обчислювальних системах.

В цьому розділі також наведені вперше запропоновані алгоритми виведення розрахункових співвідношень безпосередньо з енергетичних функціоналів, які на відміну від наявних аналогів, дозволяють виконувати розрахунки будь-якої складності.

В четвертому розділі також реалізовано застосування патерну проєктування Prototype для паралельної реалізації процесору FEM-систем.

Запропоновані алгоритми було реалізовано програмно Для їх верифікації було виконано ряд обчислювальних експериментів.

Основні наукові і практичні результати четвертого розділу опубліковано в роботах [49, 50, 70, 88, 100].

5 АВТОМАТИЗАЦІЯ АНАЛІЗУ РЕЗУЛЬТАТІВ СКІНЧЕННО-ЕЛЕМЕНТНОГО АНАЛІЗУ

5.1 Паралельний алгоритм візуалізації чисельних результатів

Аналіз чисельних результатів, отриманих при застосуванні МСЕ, зазвичай стикається з двома основними проблемами:

– великим обсягом чисельної інформації, що підлягає обробці;

– необхідністю синтезу додаткової інформації (наприклад, деформацій за відомими переміщеннями).

Вирішення цих проблем призводить до необхідності застосування спеціалізованих програмних підсистем, які отримали назву постпроцесорів. В них для наочного представлення великих масивів даних частіше за все застосовується різні способи візуалізації: графіки, діаграми, лінії рівня тощо [101, 102]. Найбільш ефективним засобом наочного представлення отриманих в результаті розрахунку результатів є їх візуалізація, коли розподіл досліджуваної величини по області розрахунку кодується певним кольором (рис. 5.1).

Для програмної реалізації побудови розподілу досліджуваної числової величини по тривимірній області в першу чергу необхідно розробити алгоритм, який реалізує цю візуалізацію для окремого ГЕ (наприклад, трикутника).

В даній роботі пропонується алгоритм розбиття граничного елемента трикутної форми на деяку підмножину “однокольорових” трикутників. Тобто таких, у вузлах яких досліджувана величина має однакове значення (рис. 5.2). Послідовна (однопотокова) версія цього алгоритму із застосуванням псевдокоду може бути описана наступним чином.

Алгоритм *Обробка граничного елемента (трикутника)*

procedure *ProcessBoundarySegment*(*Coord*, *U*)

$U = \left(U_{min}, U_{mid}, U_{max}\right)$ – вузлові значення шуканої функції

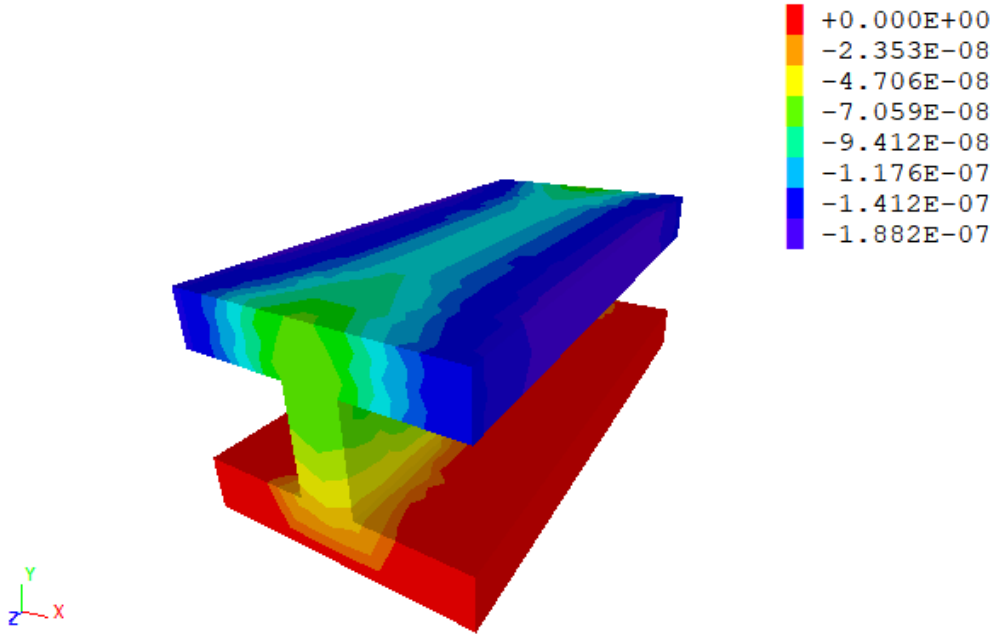

Рис. 5.1 – Приклад візуалізації розподілу вертикальної компоненти вектору переміщень по двотавровій балці

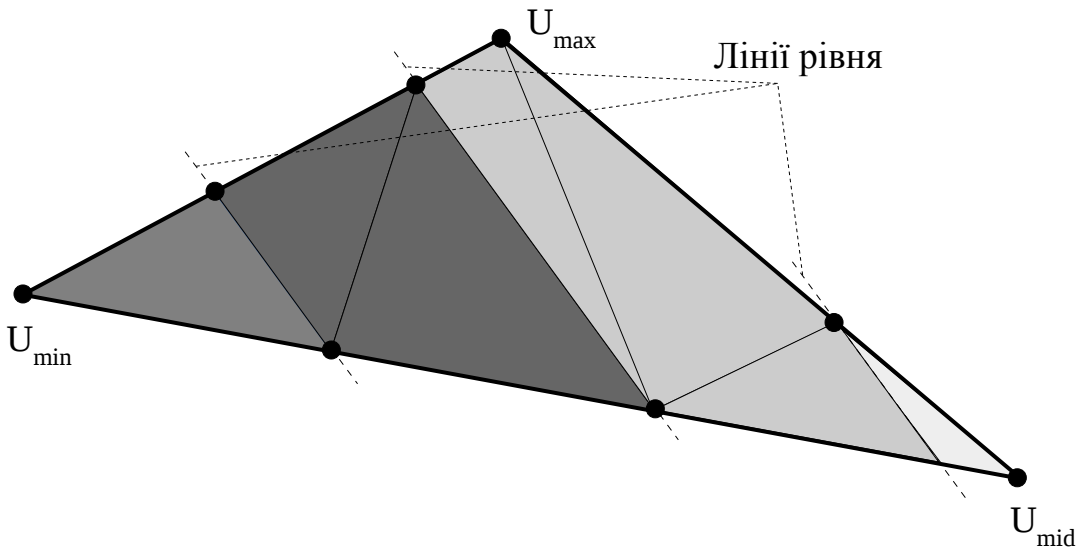

Рис. 5.2 – Схема розбиття граничного сегмента на трикутники, яким відповідає однакове значення досліджуваної величини

$Coord = (X_0, Y_0, Z_0, X_1, Y_1, Z_1, X_2, Y_2, Z_2)$ – координати вузлів ГЕ

P^{02}, P^{012} – допоміжні вектори координат

begin

$C_{min} = ColorIndex(U_{min})$

$C_{mid} = ColorIndex(U_{mid})$

$C_{max} = ColorIndex(U_{max})$

if $C_{min} = C_{mid}$ **and** $C_{mid} = C_{max}$ **then**

$ProcessTriangle(Coord, C_{min})$

return

end if

$Step = C_{max} - C_{min} + 1$

$H_x = (X_{max} - X_{min}) / Step$

$H_y = (Y_{max} - Y_{min}) / Step$

$H_z = (Z_{max} - Z_{min}) / Step$

$H_c = (C_{max} - C_{min}) / Step$

while $i \in [0, Step - 1]$ **do**

$P^{02} \leftarrow (X_0 + i \cdot H_x, Y_0 + i \cdot H_y, Z_0 + i \cdot H_z, C_{min} + i \cdot H_c)$

end while

$P^{02} \leftarrow (X_2, Y_2, Z_2, C_{max})$

$Step = C_{mid} - C_{min} + 1$

$H_x = (X_{mid} - X_{min}) / Step$

$H_y = (y_{mid} - y_{min}) / Step$

$H_z = (z_{mid} - z_{min}) / Step$

$H_c = (C_{mid} - C_{min}) / Step$

while $i \in [1, Step-1]$ **do**

$P^{012} \leftarrow (X_0 + i \cdot H_x, Y_0 + i \cdot H_y, Z_0 + i \cdot H_z, C_{min} + i \cdot H_c)$

end while

$P^{012} \leftarrow (x_1, y_1, z_1, C_{mid})$

$step = C_{max} - C_{mid} + 1$

$h_x = (x_{max} - x_{mid}) / step$

$h_y = (y_{max} - y_{mid}) / step$

$h_z = (z_{max} - z_{mid}) / step$

$h_c = (C_{max} - C_{min}) / step$

while $i \in [1, step-1]$ **do**

$P^{012} \leftarrow (x_1 + i \cdot h_x, y_1 + i \cdot h_y, z_1 + i \cdot h_z, C_{mid} + i \cdot h_c))$

end while

while $i \in [0, len(P^{02}) - 2]$ **do**

if $i < len(P^{012})$ **then**

$CurCoord = (P^{02}_{i+1,0}, P^{02}_{i+1,1}, P^{02}_{i+1,2}, P^{02}_{i,0}, P^{02}_{i,1}, P^{02}_{i,2}, P^{012}_{i,0}, P^{012}_{i,1}, P^{02}_{i,2})$

$ProcessTriangle(CurCoord, P^{02}_{i,3})$

end if

if $i+1 < len(P^{012})$ **then**

$CurCoord = (P^{02}_{i+1,0}, P^{02}_{i+1,1}, P^{02}_{i+1,2}, P^{012}_{i,0}, P^{012}_{i,1}, P^{012}_{i,2}, P^{012}_{i+1,0}, P^{012}_{i+1,1}, P^{02}_{i+1,2})$

$ProcessTriangle(CurCoord, P^{012}_{i+1,3})$

end if

end while

end procedure

Тут процедура *ProcessTriangle*() безпосередньо реалізує обробку (наприклад, збереження або зображення) трикутника, у вершинах якого досліджувана функція приймає однакові значення, а *ColorIndex*() – повертає індекс кольору в обраній кольоровій шкалі, який відповідає вузловим значенням функції.

Приклад роботи цього послідовного алгоритму наведено на рис. 5.1. Для його розпаралелювання пропонується такий самий підхід, як і у випадку побудови дискретної моделі або чисельних розрахунків: вся множина ГЕ поділяється на певну кількість підмножин, для елементів з яких вищенаведений алгоритм виконується паралельно.

На мові C++ фрагмент коду, який реалізує паралельну процедуру побудови зображення розподілу функції, що аналізується, можна описати таким чином.

```
// ...
unsigned step = be->size1() / num_thread;
vector<std::thread> thr(num_thread);

// ...

// Цикл по кількості обчислювальних вузлів
for (auto i = 0; i < num_thread; i++)
{
  // Клонування процесу...
  thr[i] = std::thread(&GLFunWidget::clone, this, i * step,
        (i == num_thread - 1) ? be->size1() : (i + 1) * step);
}

for_each(thr.begin(), thr.end(), [](auto& tr) { tr.join(); });
// ...
```

Тут слід підкреслити, що із застосуванням патерну проєктування Prototype процес розбиття загальної множини граничних сегментів на підмножини, для яких

обробка виконується паралельно, є значно простішим, ніж при застосуванні стандартних способів паралельного програмування.

5.2 Приклади візуалізації результатів розрахунку

Запропонований алгоритм візуалізації результатів розрахунку задач математичної фізики із застосуванням МСЕ було реалізовано програмно. На рис. 5.3-5.5 наведені приклади зображення розподілу компонент вектору переміщень по об’єкту “Чашка” (рис. 3.12), що знаходиться під дією власної ваги.

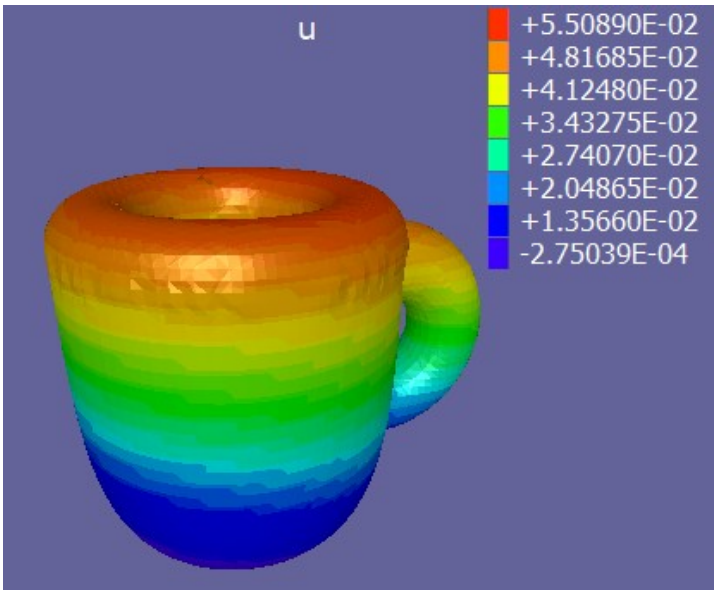

Рис. 5.3 – Розподіл компоненти вектору переміщень *u* по області розрахунку

Розподіл значень компонент вектору переміщень кодується із застосуванням кольорів, які змінюються від фіолетового (найбільші за модулем від’ємні значення) до червоного (найбільші додатні значення).

5.3 Обчислювальний експеримент

Тестування запропонованого паралельного алгоритму було виконано при побудові зображення компоненти вектору переміщень вздовж вісі абсцис (рис. 5.4).

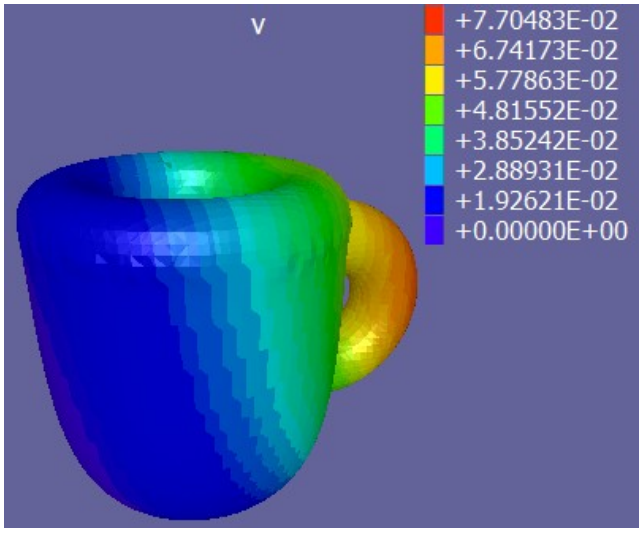

Рис. 5.4 – Розподіл компоненти вектору переміщень *v* по області розрахунку

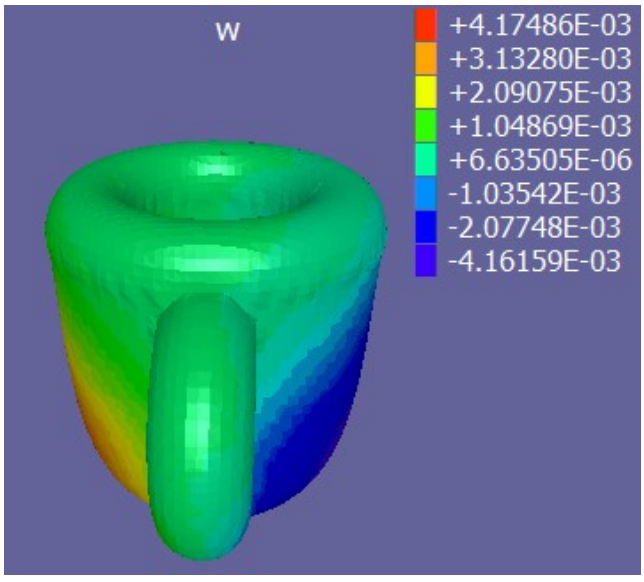

Рис. 5.5 – Розподіл компоненти вектору переміщень *w* по області розрахунку

Експеримент виконувався на комп’ютері з наступними характеристиками: процесор – AMD Ryzen 7 2700X Eight-Core Processor з тактовою частотою 3.70 Гц; об’єм оперативної пам’яті – 32 Гбайта; операційна система Windows 10 x64. Залежність часу роботи алгоритму від кількості використаних обчислювальних потоків наведена на рис. 5.6.

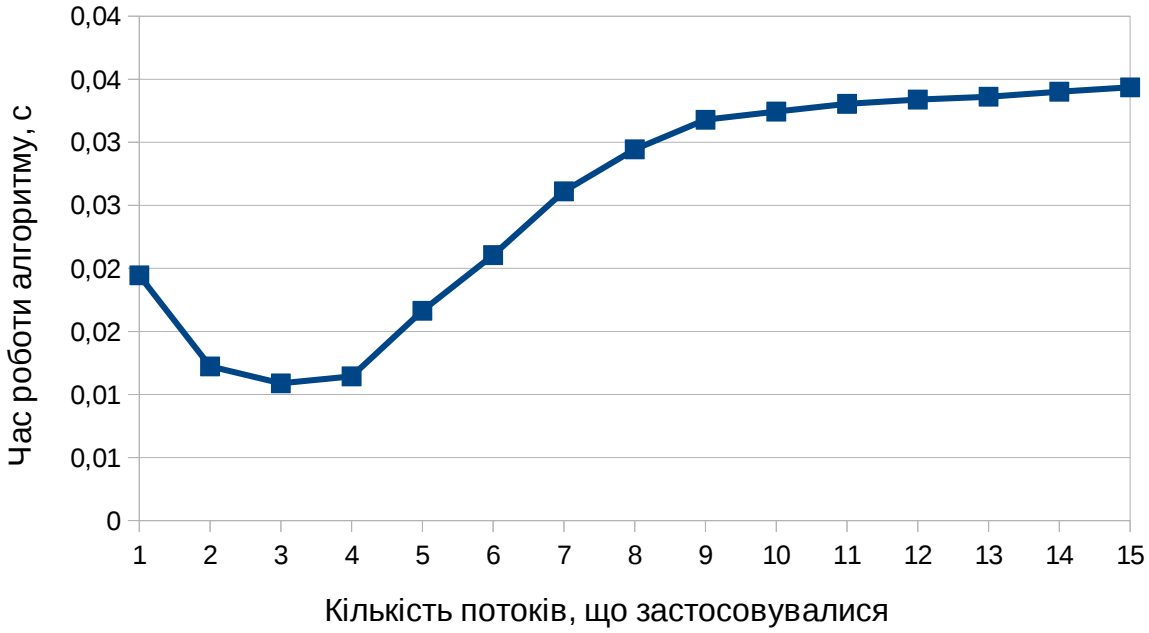

Рис. 5.6 – Залежність часу роботи паралельного алгоритму візуалізації результатів розрахунку від кількості обчислювальних потоків

Аналіз цього графіка показує, що при збільшенні кількості залучених потоків швидкість роботи алгоритму замість прискорення знижується, що пояснюється тим фактом, що накладні витрати на обслуговування багатьох потоків операційною системою стають надто великими. Таким чином, оптимальну кількість обчислювальних вузлів в кожному конкретному випадку треба визначати окремо.

Висновки до розділу 5

У п’ятому розділі наведено опис запропонованого алгоритму візуалізації чисельних результатів, отриманих при застосуванні МСЕ. На відміну від наявних аналогів, цей алгоритм підтримує паралельну реалізацію, що дозволяє істотно підвищити продуктивність його застосування при аналізі великих масивів даних. В розділі також описано застосування патерну проєктування Prototype для реалізацію цього алгоритму.

Запропоновані підходи були програмно реалізовані. Для доведення їх ефективності було проведено відповідний обчислювальний експеримент, який підтвердив доцільність його використання. Проте оптимальну кількість залучених для використання обчислювальних вузлів (або потоків) слід визначати для кожної операційної системи та комп’ютеру окремо.

Основні наукові і практичні результати п’ятого розділу опубліковано в роботах [49, 50, 87, 91].

ВИСНОВКИ

У дисертаційній роботі вирішена актуальна науково-технічна проблема підвищення ефективності розробки систем скінченно-елементного аналізу задач математичної фізики з використанням паралельних обчислень.

Отримано наступні наукові результати:

– проведено критичний огляд наявних систем скінченно-елементного аналізу складних інженерно-технічних систем, який показав необхідність створення нових підходів до розробки відповідного програмного забезпечення з підтримкою паралельних розрахунків;

– розроблено спеціалізовану предметно-орієнтовану мову для формального опису геометричних моделей об’єктів складної форми із застосуванням функціонального підходу;

– розроблено спеціалізовану предметно-орієнтовану мову опису математичних моделей і схем чисельного розрахунку задач математичної фізики із застосуванням МСЕ;

– із застосуванням патерну проєктування Prototype розроблено програмне забезпечення, яке реалізує всі аспекти скінченно-елементного аналізу: від побудови скінченно-елементної моделі, до проведення всіх розрахунків і візуалізації отриманих чисельних результатів;

– виконано ряд обчислювальних експериментів, який підтвердив ефективність запропонованих підходів при скінченно-елементному моделюванні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Норенков И. П. Основы автоматизированного проєктирования: Учеб. для вузов. 2-е изд., перераб. и доп. Москва: Из-во МГТУ им. Н. Э. Баумана, 2002. 336 с.

2. Zienkiewicz O. C., Taylor R. L., Zhu J. Z. The Finite Element Method: Its Basis and Fundamentals. Sixth edition. Butterworth-Heinemann, 2016. 753 p.

3. Зенкевич О., Морган К. Конечные элементы и аппроксимация. Москва: Мир, 1986. 318 с.

4. Зенкевич О. Метод конечных элементов в технике. Москва: Мир, 1975. 543 с.

5. Киричевский В. В., Толок В. А. Метод конечных элементов и суперэлементов в приложении к трехмерным задачам механики. Киев: Наук. думка, 2001. 364 с.

6. Метод конечных элементов: теория, алгоритмы, реализация / В. А. Толок. Киев : Наук. думка, 2003. 316 с.

7. Метод конечных элементов в вычислительном комплексе ”МІРЕЛА+” / В. В. Киричевский. Киев: Наук. думка, 2005. 403 с.

8. Design and Engineering Simulation | SIMULIA – Dassault Systemes. URL: https://www.3ds.com/products-services/simulia/ (дата звернення: 12.04.2021)

9. Engineering Simulation & 3D Design Software | Ansys. URL: https://www.ansys.com/ (дата звернення: 12.04.2021)

10. COMSOL Multiphysics ® Modelling Software. URL: https://www.comsol.com/ (дата звернення: 12.04.2021)

11. Home | Livermore Software Technology Corp. URL: http://www.lstc.com/

12. MSC Nastran – Multidisciplinary Structural Analysis. URL: https://www.mscsoftware.com/product/msc-nastran (дата звернення: 19.04.2021)

13. SOLIDWORKS. URL: https://www.solidworks.com/ (дата звернення: 12.04.2021)

14. Top Finite Element Analysis (FEA) Software List, Reviews, Comparison & Price | TEC. URL: https://www3.technologyevaluation.com/sd/category/finite-element-analysis-fea (дата звернення: 12.04.2021)

15. The dial.II Finite Element Library. URL: https://www.dealii.org/ (дата звернення: 12.04.2021)

16. FreeFEM – An open-source PDE Solver using The Finite Element Method: URL: https://freefem.org/ (дата звернення: 12.04.2021)

17. OpenCAD The Programmers Solid 3D CAD Modeller. URL: https://www.openscad.org/

18. qzCAD. URL: https://github.com/qzcad (дата звернення: 12.04.2021)

19. QFEM – The Simple FEM Solver. URL: https://github.com/SeregaGomen/QFEM (дата звернення: 12.04.2021)

20. Функциональный подход к геометрическому моделированию технических систем / С. В. Чопоров и др. Запорожье: Запорожский национальный университет, 2016. 176 с.

21. Сазонов А. А. Трехмерное моделирование в AutoCAD 2011. Москва: ДМК Пресс, 2011. 376 с.

22. AutoCAD for Mac и Windows | ПО для 2D/3D-проєктирования | Autodesk. URL: https://www.autodesk.ru/products/autocad/overview?term=1-YEAR (дата звернення: 14.04.2021)

23. Stroud I. Boundary Representation Modelling Techniques. London: Springer-Verlag, 2006. 788 p.

24. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. URL: https://gmsh.info/ (дата звернення: 14.04.2021)

25. GRUMMP Home Page. URL: http://tetra.mech.ubc.ca/GRUMMP/ (дата звернення: 15.04.2021)

26. Foley J. D. Computer Graphics: Principles and Practice. Massachusetts: Addison-Wisley, 1996. 1251 p.

27. Netgen/NGSolve. URL: https://ngsolve.org/ (дата звернення: 15.04.2021)

28. Patran – интегрирующая среда для систем анализа, моделирования и проєктирования на основе универсального графического пользовательского интерфейса. URL: http://www.mscsoftware.ru/products/patran (дата звернення: 15.04.2021)

29. Рвачев В. Л. Теория R-функций и некоторые ее приложения. Киев: Наук. думка, 1982. 106 с.

30. Pasko A., Adzhiev V., Sourin A. Savchenko V. Function representation in geometric modeling: concepts, implementation and applications. *The visual computer*. 1995. Vol. 11. P. 429-446.

31. Чопоров С. В. Построение неравномерных дискретных сеток для функциональных математических моделей на базе теории R-функций. *Радиоэлектроника, информатика, управление*. 2011. № 2. С. 70-75.

32. OFELI Library – An Object Oriented Finite Element Library. URL: http://ofeli.org/ (дата звернення: 15.04.2021)

33. GetFEM – An open-source finite element library. URL: http://getfem.org/ (дата звернення: 15.04.2021)

34. Hermes – hp-FEM Group. URL: https://www.hpfem.org/hermes/ (дата звернення: 15.04.2021)

35. MFEM – Finite Element Discretization Library. URL: https://mfem.org/ (дата звернення: 15.04.2021)

36. MATLAB – MathWorks – MATLAB & Simulink. URL: https://www.mathworks.com/products/matlab.html (дата звернення: 15.04.2021)

37. Метод конечных элементов в вычислительном комплексе “МІРЕЛА+” / В. В. Киричевский и др. Киев: Наук. думка, 2005. 403 с.

38. Home | Moose. URL: https://mooseframework.inl.gov/ (дата звернення: 15.04.2021)

39. Гоменюк С. И. Объектно-ориентированные модели и методы анализа механических процессов. Никополь: Никопольская коммунальная типография, 2004. 311 с.

40. Welcome to Python.org. URL: https://www.python.org/ (дата звернення: 17.04.2021)

41. Лутц М. Программирование на Python, том I. Санкт-Петербург: Символ-Плюс, 2011. 992 с.

42. Лутц М. Программирование на Python, том II. Санкт-Петербург: Символ-Плюс, 2011. 992 с

43. Саммерфилд М. Программирование на Python 3. Подробное руководство. Санкт-Петербург: Символ-Плюс, 2009. 608 с.

44. Plot3d. URL: github.com/SeregaGomen/pyfem/blob/master/plot/plot3d.py (дата звернення: 17.04.2021)

45. PyFEM. URL: https://github.com/SeregaGomen/pyfem (дата звернення: 17.04.2021)

46. Post-processor for FEM solver "MIRELA". URL: https://github.com/SeregaGomen/MIRELA (дата звернення: 17.04.2021)

47. FEM post-processor for fatigue analysis – SINTEF. URL: https://www.sintef.no/en/software/fem-post-processor-for-fatigue-analysis/ (дата звернення: 17.04.2021)

48. GNS ANIMATOR 4. URL: https://www.cdh-ag.com/en/software/gns-software/gns-animator-4.html (дата звернення: 19.04.2021)

49. Математичне забезпечення інженерного аналізу об’єктів аерокосмічної техніки на базі хмарних технологій: монографія / С. В. Чопоров, О. В. Кудін, Є. В. Панасенко, Д. Д. Грищак, М. С. Ігнатченко [за наук. ред. С. В. Чопорова]. Херсон: Видавничий дім “Гельветика”, 2020. 300 с.

50. Mathematical and computer modelling of engineering systems : Collective monograph / In edition by Corresponding Member of the National Academy of Sciences of Ukraine V. S. Hudramovich. Riga, Latvia : “Baltija Publishing”, 2020. 164 p.

51. Рефакторинг і патерни проєктування. URL: https://refactoring.guru/uk (дата звернення: 30.04.2021)

52. Александреску А. Современное проєктирование на C++. Обобщенное програмиирование и прикладные шаблоны проєктироования. Москва: Диалектика-Вильямс, 2019. 336 с.

53. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проєктирования. Паттерны проєктирования. Санкт-Петербург: Питер, 2001. 368 с.

54. Страуструп Б. Язык программирования C++. Москва: Бином, 2017. 1136 с.

55. Буч Г. Объектно-ориентированный анализ и проєктирование с примерами приложений на C++. Москва: Бином, 1998. 560 с.

56. Curcic M. Modern Fortran: Building efficient parallel applications. USA: Manning Publications, 2020. 416 p.

57. Готтшлинг П. C++ для инженерных и научных расчетов. Санкт-Петербург: ООО “Диалектика”, 2020. 512 с.

58. Чопоров С. В. Дискретные модели форм технических объектов: монография. Херсон: Видавничий дім “Гельветика”, 2018. 324 с.

59. Найважливіші архітектурні шаблони, які необхідно знати | DevZone. URL: https://devzone.org.ua/post/nayvazhlivishi-arkhitekturni-shabloni-yaki-neobkhidno-znati (дата звернення: 01.05.2021)

60. Coplien J. O. Advanced C++ Programming Styles and Idioms. Programming Styles and Idioms. USA: Addison-Wesley, 1992. 520 p.

61. Flynn M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*. 1972. Vol 21 (9). P. 948-960.

62. Hockney R. W., Jesshope C. R. Parallel Computers 2. Architecture, Programming and Algorithms. USA: Philadelphia, 1988. 625 p.

63. Таненбаум Э., Остин Т. Архитектура компьютера. Санкт-Петербург: Питер, 2013. 816 с.

64. Таненбаум Э., Ван Стеен М. Распределенные системы. Принципы и парадигмы. Санкт-Петербург : Питер, 2003. 877 с.

65. TOP500. URL : https://www.top500.org/ (дата звернення: 03.05.2021)

66. Антонов А. С. Параллельное программирование с использованием технологии OpenMP. Москва: Из-во МГУ, 2009. 77 с.

67. Антонов А. С. Параллельное программирование с использованием технологии MPI. Москва: Национальный Открытый Университет ”Интуит”, 2016. 84 с.

68. Рамбо Дж., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка. Санкт-Петербург: Питер, 2007. 544 с.

69. Pseudocode standard. URL: http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html (дата звернення: 27.04.2021)

70. Игнатченко М. С., Гнездовский А. В. Объектно-ориентированное моделирование задач механики. Сучасні проблеми машинобудування. Конференція молодих вчених та спеціалістів : зб. тез доп. – Харків : Інститут проблем машинобудування НАН України, 2016. С. 23-24.

71. Generative Modeling. URL: https://web.archive.org/web/20060721140550/http://www.generative-modeling.org/ (дата звернення: 19.04.2021)

72. Czarnecki K., Eisenecker U. Generative Programming. Principles and Techniques of Software Engineering Basedon Automated Configuration and Fragment-Based Component Models. Boston: Addison-Wesley Professional, 2000. 429 p.

73. Feynman R. EBNF: A Notation to Describe Syntax. URL: https://www.ics.uci.edu/~pattis/misc/ebnf2.pdf (дата звернення: 19.04.2021)

74. American Standard Code for Information Interchange. URL : https://ostermiller.org/calc/ascii.html (дата звернення: 21.04.2021).

75. Антонов А.С. Параллельное программирование с использованием технологии OpenMP: Учебное пособие. Москва: Изд-во МГУ, 2009. 77 с.

76. Хьюз К., Хьюз Т. Параллельное и распределенное программирование на C++. Москва: Издательский дом “Вильямс”, 2004. 672 с.

77. Уильямс Э. Параллельное программирование на C++ в действии. Практика разработки многопоточных программ. Москва: ДМК Пресс, 2012. 672 с.

78. Прата С. Язык программирования С. Лекции и упражнения. Москва: ООО “И.Д. Вильямс”, 2015. 928 с.

79. Лісняк А. О. Трикутні скінченні елементи у математичному моделюванні геометричних об’єктів на базі теорії R-функцій : дис. на здобуття наук. ступеня канд. фіз.-мат. наук: 01.05.02 / Запорізький національний ун-т. Запоріжжя, 2012. 138 с.

80. Толок А. В. Функционально-воксельный метод в компьютерном моделировании. Москва: ФИЗМАТЛИТ, 2016. 112 с.

81. Гоменюк С. І., Чопоров С. В., Аль-Атамнех Б. Г. М. Математичне моделювання геометричних об’єктів у паралельних комп’ютерних системах: монографія. Херсон: Видавничий дім “Гельветика”, 2018. 112 с.

82. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. Москва: Вильямс, 2005. 1296 с.

83. Lorensen W. E., Cline H. E. Marching Cubes: A high resolution 3D surface construction algorithm. URL: https://dl.acm.org/doi/10.1145/37402.37422 (дата звернення: 28.04.2021)

84. Lo S. H. D. Finite Element Mesh Generation. Boca Ration: CRC Press, 2015. 618 p.

85. Frey P. J., George P.-L. Mesh Generation. Application to finite elements. London: Wiley-ISTE, 2008. 848 p.

86. Мильцев О. М. Математичне моделювання форм багатовимірних геометричних об’єктів з використанням засобів когнітивної комп’ютерної графіки: дис. на здобуття наук. ступеня канд. фіз.-мат. наук: 01.05.02 / запорізький національний ун-т. Запоріжжя, 2020. 179 с.

87. Ігнатченко М. С., Кудін О. В. Візуалізація геометричних областей складної форми в паралельних обчислювальних системах зі спільною пам’яттю. *Вісник Запорізького національного університету.* 2019. № 2. С. 48-54.

88. А.с. № 100690. Комп’ютерна програма “Об’єктно-орієнтована бібліотека класів, що реалізують інтерпретацію складних арифметичних виразів “PARSER” / С. В. Чопоров, С. І. Гоменюк, М. С. Ігнатченко, М. І. Клименко, О. А. Головань. – опубл. 18.11.2020.

89. Ігнатченко М. С. Паралельний алгоритм пошуку границі двовимірної геометричної області, заданої неявною функцією. *Інформаційні технології та взаємодії (IT&I – 2019)*. VI міжнародна науково-практична конференція. Матеріали доповідей. Київ : Київський національний університет імені Тараса Шевченка, 2019. С. 24-25.

90. Ігнатченко М. С., Кудін О. В. Моделювання складних геометричних областей із застосуванням функціонального підходу. *Актуальні проблеми математики та інформатики* : тези доповідей Десятої Всеукраїнської, сімнадцятої регіональної наукової конференції молодих дослідників. Запоріжжя : Запорізький національний університет, 2019. С. 97-98.

91. Ігнатченко М. С., Кудін О. В. Візуалізація R-функцій в паралельних обчислювальних системах зі спільною пам’яттю. *Інформаційні технології в металургії та машинобудуванні. ІТММ’2021.* : тези доповідей міжнародної науково-практичної конференції імені професора Михальова О. І. Дніпро : НМетАУ, 2020. С. 161-165.

92. Ігнатченко М. С., Кудін О. В. Застосування паралельних обчислювальних систем зі спільною пам’яттю для візуалізації R-функцій. *Актуальні проблеми математики та інформатики* : Збірка тез доповідей Одинадцятої Всеукраїнської, вісімнадцятої регіональної наукової конференції молодих дослідників (м. Запоріжжя, 23-24 квітня 2020 р.). – Херсон : Видавничий дім “Гельветика”, 2020. С. 26-27.

93. Игнатченко М. С. Параллельная реализация алгоритма marching cubes с использованием паттерна проєктирования Prototype. *Colloquium-journal.* 2021. №10 (97). P. 49-52.

94. Федорченко А. М. Теоретична механіка. Київ: Вища школа, 1975. 516 с.

95. Тимошенко С. П., Гудьер Д. Теория упругости. Москва: Наука, 1979. 560 с.

96. Сегерлинд Л. Применение метода конечных элементов. Москва: Мир, 1979. 392 с.

97. Басов О. В. Лекции по математическому анализу. Москва: Из-во ”Физматлит”, 2014. 480 с.

98. Чабан А. Принцип Гамільтона-Остроградського в електромеханічних системах. Львів: Вид-во Т. Сороки, 2015. 463 с.

99. ISO/IEC 14882:2017 - Programming languages – C++. URL: https://www.iso.org/standard/68564.html (дата звернення: 18.05.2021)

100. Игнатченко М. С., Кудин А. В., Гнездовский А. В. Объектно-ориентированная реализация библиотеки конечно-элементного анализа на языке программирования Python. *Вісник Запорізького національного університету. Фізико-математичні науки*. 2020. № 1. С. 138-147.

101. Healy K. Data Visualization : A Practical Introduction. New Jersey: Princeton University Press, 2018. 296 p.

102. Зиновьев А. Ю. Визуализация многомерных данных. Красноярск: Изд-во КГТУ, 2000. 168 с.

ДОДАТКИ

Додаток А

Список опублікованих праць за темою дисертації

1) Математичне забезпечення інженерного аналізу об’єктів аерокосмічної техніки на базі хмарних технологій: монографія / С. В. Чопоров, О. В. Кудін, Є. В. Панасенко, Д. Д. Грищак, М. С. Ігнатченко [за наук. ред. С. В. Чопорова]. Херсон: Видавничий дім “Гельветика”, 2020. 300 с.

2) Mathematical and computer modelling of engineering systems : Collective monograph / In edition by Corresponding Member of the National Academy of Sciences of Ukraine V. S. Hudramovich. Riga, Latvia: “Baltija Publishing”, 2020. 164 p.

3) Ігнатченко М. С., Кудін О. В. Візуалізація геометричних областей складної форми в паралельних обчислювальних системах зі спільною пам’яттю. *Вісник Запорізького національного університету.* 2019. № 2. С. 48-54.

4) Игнатченко М. С., Кудин А. В., Гнездовский А. В. Объектно-ориентированная реализация библиотеки конечно-элементного анализа на языке программирования Python. *Вісник Запорізького національного університету. Фізико-математичні науки*. 2020. № 1. С. 138-147.

5) Игнатченко М. С. Параллельная реализация алгоритма marching cubes с использованием паттерна проєктирования Prototype. *Colloquium-journal.* 2021. №10 (97). P. 49-52.

6) А.с. № 100690. Комп’ютерна програма “Об’єктно-орієнтована бібліотека класів, що реалізують інтерпретацію складних арифметичних виразів

“PARSER” / С. В. Чопоров, С. І. Гоменюк, М. С. Ігнатченко, М. І. Клименко, О. А. Головань. – опубл. 18.11.2020.

7) Игнатченко М. С., Гнездовский А. В. Объектно-ориентированное моделирование задач механики. *Сучасні проблеми машинобудування*. Конференція молодих вчених та спеціалістів: зб. тез доп. Харків: Інститут проблем машинобудування НАН України, 2016. С. 23-24.

8) Ігнатченко М. С. Паралельний алгоритм пошуку границі двовимірної геометричної області, заданої неявною функцією. *Інформаційні технології та взаємодії (IT&I – 2019)*. VI міжнародна науково-практична конференція. Матеріали доповідей. Київ: Київський національний університет імені Тараса Шевченка, 2019. С. 24-25.

9) Ігнатченко М. С., Кудін О. В. Моделювання складних геометричних областей із застосуванням функціонального підходу. *Актуальні проблеми математики та інформатики*: тези доповідей Десятої Всеукраїнської, сімнадцятої регіональної наукової конференції молодих дослідників. Запоріжжя: Запорізький національний університет, 2019. С. 97-98.

10) Ігнатченко М. С., Кудін О. В. Візуалізація R-функцій в паралельних обчислювальних системах зі спільною пам’яттю. *Інформаційні технології в металургії та машинобудуванні. ITMM’2021*: тези доповідей міжнародної науково-практичної конференції імені професора Михальова О. І. Дніпро: НМетАУ, 2020. С. 161-165.

11) Ігнатченко М. С., Кудін О. В. Застосування паралельних обчислювальних систем зі спільною пам’яттю для візуалізації R-функцій. *Актуальні проблеми математики та інформатики*: Збірка тез доповідей Одинадцятої Всеукраїнської, вісімнадцятої регіональної наукової конференції молодих дослідників (м. Запоріжжя, 23-24 квітня 2020 р.). Херсон: Видавничий дім “Гельветика”, 2020. С. 26-27.

Додаток Б

Паралельна реалізація алгоритму marching cubes із застосуванням патерну проєктування Prototype

```
#ifndef TMARCHINGCUBES_H
#define TMARCHINGCUBES_H


#include <mutex>
#include <array>
#include <functional>

extern int edge_table[];
extern int triangle_table[][16];

using namespace std;

using vector3 = array<float, 3>;
using vector4 = array<float, 4>;

extern mutex mtx;

extern vector3 cross_product(const vector3&, const vector3&);
extern vector3 operator - (const vector3&, const vector3&);
extern vector4 operator + (const vector4&, const vector4&);
extern vector4 operator - (const vector4&, const vector4&);
extern vector4 operator * (const vector4&, float);
extern vector4 operator / (const vector4&, float);

struct Triangle
{
    vector3 p[3]{{0, 0, 0}, {0, 0, 0}, {0, 0, 0}};
    vector3 norm{0, 0, 0};
};

class TMarchingCubes
{
private:
    float min_value{0};
    vector3 n{100, 100, 100};
    vector3 box_min{0, 0, 0};
    vector3 box_max{0, 0, 0};
```

```
    function<float (float, float, float)> f;

    void get_cube(vector4*, int, int, int);
    void marching_cubes(int, int, vector<Triangle>&);

public:
    TMarchingCubes() {}
    TMarchingCubes(const TMarchingCubes &lhs)
    {
        min_value = lhs.min_value;
        n = lhs.n;
        box_min = lhs.box_min;
        box_max = lhs.box_max;
        f = lhs.f;
    }
    ~TMarchingCubes() {}
    void set_function(function<float (float, float, float)> f)
    {
        this->f = f;
    }
    void set_n(const vector3 &n)
    {
        this->n = n;
    }
    void set_box(const vector3 &box_min, const vector3 &box_max)
    {
        this->box_min = box_min;
        this->box_max = box_max;
    }
    void set_min_value(float minValue)
    {
        this->min_value = minValue;
    }
    void run(int begin, int end, vector<Triangle> &triangles)
    {
        marching_cubes(begin, end, triangles);
    }
    void operator () (int begin, int end, vector<Triangle> &tri)
    {
        run(begin, end, tri);
    }
    TMarchingCubes *clone(void)
    {
        return new TMarchingCubes(*this);
    }
};

#endif // TMARCHINGCUBES_H
```

Додаток В

Реалізація функціонального об’єкту

```
#ifndef VALUE_H
#define VALUE_H

#include <memory>
#include <cmath>
#include <algorithm>
#include <variant>
#include <exception>
#include "shape/shape.h"

using namespace std;



template <typename T> class TValue
{
using Scalar = double;
using Vector = variant<vector<T>, vector<double>>;
using Matrix = matrix<double>;

private:
    variant<Scalar, Vector, Matrix> val;
public:
    static array<double, 3> x;
    TValue(double d = 0) noexcept : val{d}  {}
    TValue(const TValue &rhs) noexcept: val{rhs.val} {}
    TValue(const vector<double> &s) noexcept: val{s} {}
    TValue(const vector<T> &s) noexcept: val{s} {}
    TValue(const matrix<double> &s) noexcept: val{s} {}
    ~TValue(void) noexcept = default ;
    TValue value(void) const noexcept
    {
        vector<double> res;

        if (get_if<Vector>(&val) &&
            get_if<vector<T>>(&get<Vector>(val)))
        {
            for (auto i: get<vector<T>>(get<Vector>(val)))
                res.push_back(i.value(x));
```

```
        return TValue(res);
    }
    return *this;
}
TValue &operator = (const TValue &rhs) noexcept
{
    val = rhs.val;
    return *this;
}
bool isScalar(void)
{
    return (get_if<Scalar>(&val)) ? true : false;
}
bool isVector(void)
{
    return (get_if<Vector>(&val)) ? true : false;
}
bool isMatrix(void)
{
    return (get_if<Matrix>(&val)) ? true : false;
}
double asScalar(void)
{
    if (not get_if<Scalar>(&val))
        throw TError(Message::AsScalar);
    return get<Scalar>(val);
}
vector<double> asVector(void)
{
    vector<double> res;

    if (not get_if<Vector>(&val))
        throw TError(Message::AsVector);
    if (get_if<vector<T>>(&get<Vector>(val)))
        for (auto i: get<vector<T>>(get<Vector>(val)))
            res.push_back(i.value(x));
    else
        res = get<vector<double>>(get<Vector>(val));
    return res;
}
matrix<double> asMatrix(void)
{
    if (not get_if<Matrix>(&val))
        throw TError(Message::AsMatrix);
    return get<Matrix>(val);
}
friend TValue operator - (const TValue &rhs) noexcept
{
    TValue ret(rhs);

    if (get_if<Scalar>(&ret.val))
```

```
        get<Scalar>(ret.val) = -get<Scalar>(ret.val);
    else if (get_if<Vector>(&ret.val))
    {
        if (get_if<vector<T>>(&get<Vector>(ret.val)))
            inverse(get<vector<T>>(get<Vector>(ret.val)));
        else
            inverse(get<vector<double>>(get<Vector>(ret.val)));
    }
    else
        inverse(get<Matrix>(ret.val).asVector());
    return ret;
}
friend TValue diff(const TValue &lhs, const TValue &rhs)
{
    TValue ret(lhs);
    Direct dir;

    if (not get_if<vector<T>>(&get<Vector>(ret.val)) or
          not get_if<Scalar>(&rhs.val))
        throw TError(Message::InvalidOperation);
    dir = static_cast<Direct>(get<Scalar>(rhs.val));
    for (auto &i: get<vector<T>>(get<Vector>(ret.val)))
        i = i.diff(dir);
    return ret;
}
friend TValue var(const TValue &lhs, const TValue &rhs)
{
    TValue l{lhs.value()},
           r{rhs.value()};
    matrix<double> res;

    if (get_if<Vector>(&lhs.val) and get_if<Vector>(&rhs.val))
    {
        res.resize(get<vector<double>>(get<Vector>(l.val))
          .size(), get<vector<double>>(get<Vector>(l.val)).
          size() + 1);
        for (auto i = 0u; i < res.size1(); i++)
            for (auto j = 0u; j < res.size1(); j++)
                res[i][j] =
                  get<vector<double>>(get<Vector>(l.val))[i] *
                  get<vector<double>>(get<Vector>(r.val))[j] +
                  get<vector<double>>(get<Vector>(l.val))[j] *
                  get<vector<double>>(get<Vector>(r.val))[i];
    }
    else if (get_if<Scalar>(&lhs.val) and
        get_if<Vector>(&rhs.val))
    {
      res.resize(get<vector<double>>(get<Vector>(r.val)).size(),
          get<vector<double>>(get<Vector>(r.val)).size() + 1);
        for (auto i = 0u; i < res.size1(); i++)
            res[i][res.size1()] = get<double>(l.val) *
```

```
                get<vector<double>>(get<Vector>(r.val))[i];
        }
        else
            throw TError(Message::InvalidOperation);
        return TValue(res);
    }
    friend ostream &operator << (ostream &out, const TValue &rhs)
    {
        if (get_if<Scalar>(&rhs.val))
            out << "Scalar: " << get<Scalar>(rhs.val);
        else if (get_if<Vector>(&rhs.val))
        {
            out << "Vector: { ";
            if (get_if<vector<T>>(&get<Vector>(rhs.val)))
                for (auto &i: get<vector<T>>(get<Vector>(rhs.val)))
                    out << i << ' ';
            else
                for (auto &i:
                   get<vector<double>>(get<Vector>(rhs.val)))
                    out << i << ' ';
            out << "}";
        }
        else
        {
            out << "Matrix: [ ";
            for (auto i = 0u; i < get<Matrix>(rhs.val).size1(); i++)
            {
                for (auto j = 0u; j < get<Matrix>(rhs.val).size2();
                   j++)
                    out << get<Matrix>(rhs.val)[i][j] << ' ';
                out << endl;
            }
            out << "]";
        }
        return out;
    }
    friend TValue operator + (const TValue &lhs, const TValue &rhs)
    {
        TValue res;

#ifdef DEBUG
        watch_val(lhs);
        watch_val(rhs);
#endif
        if (get_if<Scalar>(&lhs.val) and get_if<Scalar>(&rhs.val))
            res.val = get<Scalar>(lhs.val) + get<Scalar>(rhs.val);
        else if (get_if<Vector>(&lhs.val) and
             get_if<Vector>(&rhs.val))
            res.val =
               get<vector<double>>(get<Vector>(lhs.value().val)) +
               get<vector<double>>(get<Vector>(rhs.value().val));
```

```
        else if (get_if<Matrix>(&lhs.val) and
           get_if<Matrix>(&rhs.val))
            res.val = get<Matrix>(lhs.val) + get<Matrix>(rhs.val);
        else
            throw TError(Message::InvalidOperation);
#ifdef DEBUG
        watch_val(res);
#endif
        return res;
    }
    friend TValue operator - (const TValue &lhs, const TValue &rhs)
    {
        TValue res;

        if (get_if<double>(&lhs.val) and get_if<double>(&rhs.val))
            res.val = get<Scalar>(lhs.val) - get<Scalar>(rhs.val);
        else if (get_if<Vector>(&lhs.val) and
                   get_if<Vector>(&rhs.val))
            res.val =
              get<vector<double>>(get<Vector>(lhs.value().val)) -
              get<vector<double>>(get<Vector>(rhs.value().val));
        else if (get_if<Matrix>(&lhs.val) and
              get_if<Matrix>(&rhs.val))
            res.val = get<Matrix>(lhs.val) - get<Matrix>(rhs.val);
        else
            throw TError(Message::InvalidOperation);
        return res;
    }
    friend TValue operator * (const TValue &lhs, const TValue &rhs)
    {
        TValue res;

        if (get_if<Scalar>(&lhs.val))
        {
            if (get_if<Scalar>(&rhs.val))
                res.val = get<Scalar>(lhs.val) *
                   get<Scalar>(rhs.val);
            else if (get_if<Vector>(&rhs.val))
                res.val = get<Scalar>(lhs.val) *
                  get<vector<double>>(get<Vector>(rhs.value().val));
            else
                res.val = get<Scalar>(lhs.val) *
                    get<Matrix>(rhs.val);
        }
        else if (get_if<Scalar>(&rhs.val))
        {
            if (get_if<Scalar>(&lhs.val))
                res.val = get<Scalar>(lhs.val) *
                    get<Scalar>(rhs.val);
            else if (get_if<Vector>(&lhs.val))
                res.val =
```

```
                    get<vector<double>>(get<Vector>(lhs.value().val)) *
                    get<Scalar>(rhs.val);
            else
                res.val = get<Matrix>(lhs.val) *
                    get<Scalar>(rhs.val);
        }
        else
            throw TError(Message::InvalidOperation);
        return res;
    }
    friend TValue operator / (const TValue &lhs, const TValue &rhs)
    {
        TValue res;

        if (not get_if<Scalar>(&rhs.val))
            throw TError(Message::InvalidOperation);
        if (get_if<Scalar>(&lhs.val))
            res.val = get<Scalar>(lhs.val) / get<Scalar>(rhs.val);
        else if (get_if<Vector>(&lhs.val))
            res.val =
              get<vector<double>>(get<Vector>(lhs.value().val)) /
              get<Scalar>(rhs.val);
        else
            res.val = get<Matrix>(lhs.val) / get<Scalar>(rhs.val);
        return res;
    }
};

#endif // VALUE_H
```

Додаток Г

Реалізація функції форми для лінійного тетраедру

```
// Параметры функции формы линейного тетраэдра
// N(x, y) = c0 + c1 * x + c2 * y
struct TShape3d4
{
    // Дифференцирование
    inline static vector<double> diff(const vector<double> &c,
         Direct direct)
    {
        vector<double> ret;

        if (direct == Direct::X)
            ret = {c[1], 0, 0, 0};
        else if (direct == Direct::Y)
            ret = {c[2], 0, 0, 0};
        else if (direct == Direct::Z)
            ret = {c[3], 0, 0, 0};
        else
            throw TError(Message::InternalError);
        return ret;
    }
    inline static double value(const vector<double> &c,
         const array<double, 3> &x) noexcept
    {
        return (c.size() == 1) ? c[0] : c[0] + c[1] * x[0] + c[2] *
           x[1] + c[3] * x[2];
    }
    inline static matrix<double> jacobi(int,
           const matrix<double> &x)
    {
        matrix<double> jacobi(3, 3);
        vector<double> d_xi{ -1.0, 1.0, 0.0, 0.0 },
                       d_eta{ -1.0, 0.0, 1.0, 0.0 },
                       d_psi{ -1.0, 0.0, 0.0, 1.0 };

        for (auto j = 0; j < 3; j++)
            for (auto k = 0; k < size(); k++)
            {
                jacobi(0, j) += d_xi[k] * x(k, j);
                jacobi(1, j) += d_eta[k] * x(k, j);
                jacobi(2, j) += d_psi[k] * x(k, j);
```

```
            }
        return jacobi;
    }
    inline static constexpr int size(void) noexcept
    {
        return 4;
    }
    inline static constexpr int freedom(void) noexcept
    {
        return 3;
    }
    inline static constexpr int quadrature_degree(void)
    {
        return 5;
    }
    inline static double w(int i)
    {
        return array<double, 5>{ -0.13333333333, 0.075,
                                  0.075, 0.075, 0.075 }[i];
    }
    inline static double xi(int i)
    {
        return array<double, 5>{ 0.25, 0.5, 0.16666666667,
                                 0.16666666667, 0.16666666667 }[i];
    }
    inline static double eta(int i)
    {
        return array<double, 5>{ 0.25, 0.16666666667, 0.5,
                                0.16666666667, 0.16666666667 }[i];
    }
    inline static double psi(int i)
    {
        return array<double, 5>{ 0.25, 0.16666666667, 0.16666666667,
                        0.5, 0.16666666667 }[i];
    }
    inline static double coeff(matrix<double> &x, int i, int j)
    {
        return vector<double>{ 1.0, x(i, 0), x(i, 1), x(i, 2) }[j];
    }
    inline static array<double, 3> x(int i, const matrix<double> &x)
    {
        return { x(0, 0) * (1.0 - xi(i) - eta(i) - psi(i)) +
            x(1, 0) * xi(i) + x(2, 0) * eta(i) + x(3, 0) * psi(i),
            x(0, 1) * (1.0 - xi(i) - eta(i) - psi(i)) + x(1, 1) *
            xi(i) + x(2, 1) * eta(i) + x(3, 1) * psi(i),
            x(0, 2) * (1.0 - xi(i) - eta(i) - psi(i)) +
            x(1, 2) * xi(i) + x(2, 2) * eta(i) + x(3, 2) * psi(i) };
    }
};
```