НАЦІОНАЛЬНА АКАДЕМІЯ НАУК УКРАЇНИ
ІНСТИТУТ ПРОБЛЕМ МОДЕЛЮВАННЯ В ЕНЕРГЕТИЦІ
ІМ. Г.Є. ПУХОВА

Кваліфікаційна наукова праця
на правах рукопису

СУШКО СЕРГІЙ ВОЛОДИМИРОВИЧ

УДК 004.89:004.4

ДИСЕРТАЦІЯ

МЕТОДИ ОПТИМАЛЬНОГО РОЗПАРАЛЕЛЮВАННЯ ПРОГРАМ МІКРОПРОЦЕСОРНИХ СИСТЕМ ДЛЯ ПІДВИЩЕННЯ ЇХ ЕФЕКТИВНОСТІ

Спеціальність 05.13.05 – комп’ютерні системи та компоненти

Галузь знань – інформаційні технології

Подається на здобуття наукового ступеня кандидата технічних наук

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

С.В. Сушко

Науковий керівник – Чемерис Олександр Анатолійович
доктор технічних наук, старший науковий співробітник

Київ – 2021

АНОТАЦІЯ

Сушко С.В. Методи оптимального розпаралелювання програм мікропроцесорних систем для підвищення їх ефективності. – На правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.05 – комп'ютерні системи та компоненти. Інститут проблем моделювання в енергетиці ім. Г.Є. Пухова НАН України, Київ, 2021.

У дисертаційній роботі отримано нові науково-прикладні результати щодо підвищення ефективності виконання комп’ютерних програм за рахунок використання методу розбиття обчислювальних циклів на блоки з підбором кращих розмірів блоків розбиття. Отримані експериментальні дані свідчать про прискорення швидкодії та енергоефективності обчислень для більшості тестових програм при використанні запропонованого методу, що отримав назву метод інтелектуального блочного розбиття.

Використання методу розбиття обчислювальних циклів на блоки покращує локальність даних і, тим самим, надає передумови для прискорення виконання комп’ютерних програм. В той же час, при використанні цього методу не завжди приділяється увага безпосередньо значенню самих розмірів блоків. В цій роботі автором досліджено вплив розмірів блоків розбиття на час виконання тестових комп’ютерних програм. Автор наводить дані та графіки, що підтверджують складну залежність часу виконання тестових програм від розмірів блоків розбиття.

Для пошуку кращих розмірів блоків розбиття автором використано дискретний метод рою часток, що ітеративно підбирає різні варіанти розмірів блоків і, тим самим виконує пошук найкращих розмірів блоків, що забезпечить мінімальний час виконання програм. Отже, час витрачений на пошук найкращих параметрів оптимізаційного методу дозволить знайти саме такі параметри, що прискорюють швидкодію комп’ютерної програми максимально можливим способом. Знайдені параметри оптимізації в подальшому можуть бути використані в оптимізованій програмі на будь-якій

кількості пристроїв. Таким чином, досягається доцільність тривалого пошуку кращих параметрів оптимізації.

Наукова новизна одержаних результатів полягає в наступному:

Вперше:

– Запропоновано ітераційний процес розпаралелювання операторів циклів мікропроцесорних програм, який відрізняється від відомих методів визначенням оптимального рішення щодо розбиття ітераційного простору оператора циклу на окремі блоки.

– Запропоновано оцінку доцільності оптимізації коду програм і отримано експериментальні дані, що засвідчують покращення ефективності обчислень (за часом або енергоспоживанням), в більшості випадків при застосуванні методу розбиття на блоки.

– Розроблено метод інтелектуального блочного розбиття, що виконує пошук оптимальних розмірів прямокутних блоків розбиття ітераційного простору операторів циклу мікропроцесорних програм на основі дискретного методу рою часток, який може бути застосовано до довільних обчислювальних циклів написаних мовами програмування С або С++.

Удосконалено:

– Метод оцінки часу виконання тестових програм при використанні різноманітних методів розбиття на блоки, який базується на автоматичному послідовному багаторазовому запуску тестових програм і фіксації отриманих результатів швидкодії, що можуть бути в подальшому проаналізовані.

– Дискретний метод рою часток шляхом визначення коефіцієнтів методу, а саме – початкового коефіцієнту інерції, індивідуального та соціального коефіцієнтів, при яких пошук розмірів блоків розбиття в задачі прискорення швидкодії виконується швидше класичного методу.

– Використання дискретного методу рою часток шляхом визначення кращих початкових даних для розташування часток рою, що зменшує число ітерацій для знаходження оптимального рішення.

В роботі наводяться дані вимірювань, що свідчать не тільки про прискорення виконання більшості тестових програм при використанні методів розбиття на блоки, але й про підвищення енергоефективності обчислень. Цей факт свідчить про доцільність використання методу в «зелених» обчисленнях.

Також в роботі наводиться порівняння енергоефективності обчислень на тих самих тестових програмах і розмірності даних для двох різних апаратних платформ: для плати Raspberry Pi 3 та для ПК на базі Intel® Core™ i5-4670K. Отримані дані свідчать, що в 92.8% випадків більш енергоефективною є плата Raspberry Pi 3.

Розробка знайде використання у комп'ютерних програмах, що написані мовами програмування C та C++. Перевагою запропонованого методу інтелектуального блочного розбиття є його універсальність; він може використовуватись для комп'ютерних програм будь-якого призначення, апаратної платформи та використаного компілятора. Недоліком методу є його ітеративна природа – необхідність вимірювання часу виконання програм для кожної запропонованої пари розмірів блоків розбиття. При великій кількості часток пошуку та ітерацій, а також якщо час виконання комп'ютерних програм достатньо довгий, підбір кращих розмірів блоків розбиття може зайняти тривалий час.

Ключові слова: оптимізація комп'ютерних програм, енергоефективні обчислення, прискорення швидкодії комп'ютерних програм, метод розбиття на блоки, дискретний метод рою часток, метод інтелектуального блочного розбиття.

ABSTRACT

Sushko S. V. Methods of optimal parallelization of programs of microprocessor systems to increase its efficiency. – As the manuscript.

Thesis for candidate's degree in technical science by speciality 05.13.05 – computer systems and components. – Pukhov Institute for Modeling in Energy Engineering, National Academy of Sciences of Ukraine, Kyiv, 2021.

New scientific and applied results for accelerating of execution of computer programs by using of tiling method with choice of best tiles' sizes were obtained in the thesis. Obtained experimental data show an improvement in processing speed and energy efficiency of calculations for most test programs with using of proposed Smart Tiling Method.

Usage of tiling method improves locality of data and, thus, provides the prerequisites for accelerating of execution of computer programs. At the same time, when using this method, attention is not always focused directly to values of tile sizes themselves. In this work the author investigates an influence of tile sizes on execution time of test computer programs. The author provides data and graphs that confirm a complicated dependence between tile sizes and execution time of test programs.

To find best tile sizes the author used Discrete Particle Swarm Optimization Method, which iteratively selects different tile sizes based on execution time of computer program. It searches for best tile sizes, which will provide minimum program's execution time. Thus, time spent on iterative search for best parameters of the optimization method will allow to find that parameters that will improve performance of computer program in best possible way. Once optimization parameters had been found, they will be used in optimized program in any number of computing devices.

The scientific novelty of the obtained results is as follow.

In first time:

– An iterative process of parallelization of loop operators of microprocessor programs is proposed, which differs from known methods by determining of optimal solution for dividing of iterative space of loop operators into separate blocks.

– The estimation of expediency of code optimization of programs was proposed. Experimental data which confirm improvement of efficiency of calculations for execution time and power consumption in most cases for tiling method were obtained.

– Smart Tiling Method was developed. The Method searches for optimal size of rectangular tiles of iterative space partitioning of microprocessor program loop operators. The Method is based on Discrete Particle Swarm Optimization Method and can be applied to arbitrary computational loops written with C or C++ programming languages.

Improved:

– The method of estimation of execution time of test programs for using different tiling methods. The method of estimation is based on automatic sequential multiple run of test programs and collecting of obtained results of performance, which can be further analyzed.

– Discrete Particle Swarm Optimization Method by determining of its coefficients – social, individual, and initial velocity inertia. With found coefficients a process of searching of tile size for problem of speed improvement is performed faster than classical method.

– Using Discrete Particle Swarm Optimization Method by determining the best initial locations of particles, which reduces number of iterations to find the optimal solution.

The thesis presents measurement data which indicate not only acceleration of most test programs by using of tiling method, but also an improvement of energy efficiency of calculations. This fact indicates of prospects of using the method in "green" calculations.

In the thesis also energy efficiency of computations for the same test programs and dame data dimension for two different hardware platforms was compared. It was

made for Raspberry Pi 3 board and PC based on CPU Intel® Core ™ i5-4670K. The obtained data show that in 92.8% of cases Raspberry Pi 3 board is more energy efficient.

This research can be used for computer programs written in C and C ++ programming languages. The advantage of proposed Smart Tiling Method is its versatility; it can be used for computer programs of any purpose, hardware platform and used compiler. The disadvantage of the method is its iterative nature – a requirement to measure of execution time of programs for each proposed pair of tile sizes. Selection of best tile sizes can take a long time if large number of particles and iteration is used, also if execution time of computer program is long enough.

Keywords: optimization of computer programs, energy efficient calculations, accelerating of computer programs, tiling method, Discrete Particle Swarm Optimization Method, Smart Tiling Method.

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Chemeris A. Influence of Software Optimization on Energy Consumption of Embedded Systems / Chemeris A., Lazorenko D., Sushko S. // Green IT Engineering: Components, Networks and Systems Implementation. Studies in Systems / Kharchenko V., Kondratenko Y., Kacprzyk J. (eds) – Cham.: Springer, 2017. – Vol. 105. – С. 111–133.

2. Чемерис А.А. Исследование быстродействия и энергопотребления при автоматической оптимизации методами разбиения на блоки и распараллеливания для вычислений на платформе X64 / А.А. Чемерис, С.В. Сушко // Моделювання та інформаційні технології. – Київ, 2017. – Вип. 80. – С. 52-60.

3. Сушко С.В. Вплив розмірів блоків розбиття операторів циклів на час виконання комп'ютерних програм / С.В. Сушко, О.А. Чемерис // Моделювання та інформаційні технології. – Київ, 2018. – Вип. 82. – С.110-117.

4. Sushko S. Dependency between Tiles' Sizes and Program Execution Time / Sushko S., Chemerys A. // Measurement Automation Monitoring. – Gliwice, 2018. – Vol. 64, No.2, – P. 28-30, Wydawnictwo PAK, ISSN 2450-2855.

5. Chemerys A. Analysis and Optimization of the Sizes of the Iteration Space Tiles During the Parallelization of Program Loop Operators / Chemerys A., Sushko S. // Advances in Cyber-Physical Systems. – Київ, 2018. – Випуск 3, Номер 1, С. 1-6.

6. Сушко С.В. Визначення енергоефективності обчислень на різних обчислювальних системах / С.В. Сушко, О.А. Чемерис // Моделювання та інформаційні технології. – Київ, 2018. – Вип. 85. – С. 34-39.

Наукові праці, які засвідчують апробацію матеріалів дисертації

1. Чемерис А.А. Исследование эффективности выполнения программ, оптимизированных на основе полиэдральной модели /

А.А. Чемерис, С.В. Сушко // Сборник трудов 5й международной конференции SIMULATION-2016. – Киев, 2016. – С. 241-244.

2. Чемерис А.А. Сравнение эффективности автоматической оптимизации на основе полиэдральной модели / А.А. Чемерис, С. В. Сушко // Summer InfoCom 2016: Матеріали II Міжнародної науково-практичної конференції, м. Київ, 1-3 червня 2016 р. – К.:Вид-во "Інжиніринг", 2016. – С. 74-76.

3. Чемерис А.А. Оценка снижения времени выполнения тестовых программ при применении автоматической оптимизации на основе полиэдральной модели на Raspberry Pi 3 / А.А. Чемерис, С.В. Сушко // Winter InfoCom 2016: Матеріали III Міжнародної науково-практичної конференції, м. Київ, 1-2 грудня 2016 р. – К.:Вид-во «Інжиніринг», 2016. – С. 63-65.

4. Chemeris A. The Dependence of Microprocessor System Energy Consumption on Software Optimization / A. Chemeris, S. Sushko // 2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO), April 18-20, 2017, Kyiv, ISBN: 978-1-5386-1700-7, P. 451-454.

5. Сушко С.В. Вплив розмірів блоків розбиття на час виконання програм. / С.В. Сушко // Науково-технічна конференція молодих вчених і спеціалістів ІПМЕ ім. Г. Є. Пухова НАН України, – Київ, 12 травня 2018, – С. 34-37.

6. Sushko S. Increasing An Energy Efficiency Of Computational System By Using Automatic Software Optimization / S. Sushko, A. Chemeris // 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT-2018), May 24-27 2018, Kyiv, ISBN 978-1-5386-5902-1, P. 613-617.

7. Сушко С. Вплив розмірів блоків розбиття операторів циклів на час виконання комп'ютерних програм / Сушко С. Чемерис О. // Збірник праць 6-ї міжнародної конференції Моделювання-2018, 12-14 вересня 2018, Київ, С. 234-238.

8. Чемерис О.А. Методи штучного інтелекту при оптимізації роботи мікропроцесорних систем. / О.А. Чемерис, С.В. Сушко // XII Міжнародна

Науково-Практична Конференція Комп’ютерні Системи та Мережні Технології 28-30 березня 2019, Київ, С. 127-129.

9. Chemeris A. Usage of Discrete Particle Swarm Optimization Method for the Searching of Optimal Tile Size / A. Chemeris, S. Sushko // 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T-2019), October 8-11, 2019, Kyiv, P.202-206. ISBN: 978-1-7281-4183-1.

10. Сушко С.В. Універсальні оптимізаційні методи програмного забезпечення / Сушко С.В. // Науково-практична конференція Безпека енергетики в епоху цифрової трансформації, 20 грудня 2019, Київ, С. 89-91.

ЗМІСТ

СПИСОК ТЕРМІНІВ ТА УМОВНИХ СКОРОЧЕНЬ

ПК	Персональний комп’ютер
ПЗ	Програмне забезпечення
КП	Комп’ютерна програма
ТП	Тестова програма
ЕОМ	Електронна обчислювальна машина
КМОН	Компліментарна структура метал-оксид-напівпровідник
КІХ	Кінцева імпульсна характеристика
МРОЦНБ	Метод розбиття обчислювальних циклів на блоки (tiling method)
МРЧ	Метод рою часток (Particle Swarm Optimization Method)
ДМРЧ	Дискретний метод рою часток (Discrete Particle Swarm Optimization Method)
OpenMP	Специфікація для набору директив компілятора, підпрограм бібліотеки та змінних середовища, які можуть бути використані для визначення паралелізму високого рівня в програмах Fortran та C/C++
ПЛІС	Програмована логічна інтегральна схема
МІБР	Метод інтелектуального блочного розбиття (smart tiling)
SoC	Система на чипі (System on Chip)
S2S перетворення	Перетворення вихідного коду у оптимізований вихідний код (source-to-source transformation)

ВСТУП

Актуальність теми. Потужність обчислювальних систем стрімко розвивається використовуючи новітні розробки у галузі апаратних компонентів. Водночас, також розвиваються обчислювальні алгоритми та методи обчислень.

Задля максимально ефективного використання апаратних можливостей постійно вдосконалюються компілятори, створюються нові мови програмування та збагачуються новим синтаксисом існуючи мови програмування.

Аналіз літературних джерел показав, що існує дуже багато засобів і методів оптимізації, які використовуються на різних рівнях оптимізації. Найбільше методів та алгоритмів оптимізації використовується в компіляторах. Недоліком вдосконалення оптимізаційних можливостей компіляторів є обмеженість налаштувань оптимізації компілятора та прихована логіка його роботи. В той же час окремим напрямком досліджень щодо оптимізації програмного коду є перетворення вихідного коду в інший вихідний код, так зване S2S перетворення. Такий підхід дозволяє зосередитися на одному або декількох методах оптимізації та водночас може використовуватися сумісно з іншими засобами оптимізації, наприклад він доповнює оптимізаційні методи компіляторів.

Проблемою оптимізації програмного забезпечення займались і займаються як науково-академічні організації так і науково-дослідні відділи виробників програмного забезпечення, як в Україні, так і в світі. Варто виділити великий внесок в теорію і практику розпаралелювання програм таких науковців, як, зокрема, П. Фотріє, А. Фрабле, В. П'ю, М. Лем, А. Дарта, Вл.В. Воєводіна, Н.А. Лиходіда, В.А. Вальковського, І.А. Каляєва, В.П. Гергеля, В.Г. Хорошевського, Ю.В. Капітонової, О.А. Летичєвського, В.М. Бєлецького, А.Ю. Дорошенка та інших.

Різноманіття методів оптимізації, можливість або неможливість використання одночасно декількох методів оптимізації складає множину методів оптимізації. Деякі з методів оптимізації мають власні параметри, які також впливають на ефективність оптимізації. Правильний підбір набору методів оптимізацій та значень їх параметрів може додатково підвищити ефективність оптимізації. Деякі параметри оптимізації явно задаються компілятору, деякі встановлюються на етапі попереднього аналізу коду або евристично. Незважаючи на вагомі результати досліджень, в них не завжди приділяється достатньо уваги саме вибору найбільш ефективного методу та його параметрів для кожного конкретного прикладу. Саме тому підбір параметрів оптимізаційного методу для конкретної частини програмного коду є актуальною задачею для пошуку найбільш ефективного результату оптимізації. В даній роботі розглядається підвищення ефективності методів розбиття на блоки та розпаралелювання та пропонуються підходи щодо підбору найкращих розмірів блоків розбиття для методу розбиття на блоки.

Дана робота досліджує метод розбиття на блоки, що виконує S2S перетворення універсального програмного коду мовами програмування C або C++, який може виконуватися швидше та/або більш енергоефективно. Розглядається ефективність даного методу на різноманітних тестових даних. Пропонується спосіб підбору параметрів методу, що призводить до максимальної ефективності методу оптимізації з точки зору часу обчислень. Запропоновано використання дискретного методу рою часток в задачі пошуку кращих розмірів блоків розбиття. Отриманий метод дістав назву метод інтелектуального блочного розбиття.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційну роботу виконано у межах відомчих науково-дослідних тем НАН України, а саме: «Розвиток методів зниження енергоспоживання обчислювальних систем за рахунок оптимізації обробки масивів даних» (шифр – ФРІСК)» (номер державної реєстрації 0114U000879) – проведено дослідження зниження енергоспоживання при проектуванні

мікропроцесорних систем за рахунок ефективного розпаралелювання циклів; «Розвиток теорії, розробка новітніх інформаційних технологій в задачах комплексного моделювання та управління процесами перетворення та використання енергії (шифр: НОВІНТЕХ)» (номер державної реєстрації 0117U004347) – розроблено метод визначення оптимальних розмірів прямокутних блоків розбиття ітераційного простору операторів циклу мікропроцесорних програм на основі дискретного методу рою часток; «Розвиток теорії, розробка методів та засобів реалізації гібридних експертно-моделюючих комп'ютерних систем в задачах комплексного управління перетворенням енергії (шифр – ГІБРИД)» (номер державної реєстрації 0112U000050) – проведено дослідження щодо комбінування різних методів оптимізації програм з метою вибору найкращої в плані часу виконання програми комбінації.

Дисертаційна робота виконана в Інституті проблем моделювання в енергетиці ім. Г.Є. Пухова Національної академії наук України.

Метою роботи є підвищення ефективності функціонування мікропроцесорних систем за рахунок розпаралелювання та оптимізації програмного забезпечення.

Задача роботи полягає в розробці методів оптимізації паралельних програм для мікропроцесорних систем, доведенні ефективності цих методів та впровадження їх у програмні додатки.

Основні задачі дослідження відповідно до поставленої мети полягають у наступному:

1. Виконати аналіз методів оптимізації комп'ютерних програм, що спрямовані на підвищення ефективності обчислень (швидкодію, енергоефективність, використання пам'яті, тощо);

2. Вдосконалити метод оцінки часу виконання тестових програм при використанні різноманітних методів розбиття на блоки;

3. Розробити алгоритм та програму, що виконує в автоматичному режимі вимірювання часу виконання тестових програм;

4. Розробити алгоритм та програму, що виконує пошук мінімального часу виконання тестових програм для різних розмірів блоків розбиття використовуючи дискретний метод рою часток;

5. На основі результатів дослідження розробити метод пошуку найкращих розмірів блоків розбиття на базі дискретного методу рою часток;

6. Впровадити результати роботи.

Об'єктом дослідження є процес оптимізації програмного коду операторів циклів мікропроцесорних програм.

Предметом дослідження є комплекси автоматичної розробки програмного забезпечення мікропроцесорних систем.

Методи досліджень базуються на використанні теорії множин, методах трансформації обчислювальних циклів та еволюційних оптимізаційних методах.

Наукова новизна одержаних результатів полягає в наступному:

Вперше:

Запропоновано ітераційний процес розпаралелювання операторів циклів мікропроцесорних програм, який відрізняється від відомих методів визначенням оптимального рішення щодо розбиття ітераційного простору оператора циклу на окремі блоки.

Запропоновано оцінку доцільності оптимізації коду програм і отримано експериментальні дані, що засвідчують покращення ефективності обчислень (за часом або енергоспоживанням), в більшості випадків при застосуванні методу розбиття на блоки.

Розроблено метод інтелектуального блочного розбиття, що виконує пошук оптимальних розмірів прямокутних блоків розбиття ітераційного простору операторів циклу мікропроцесорних програм на основі дискретного методу рою часток, який може бути застосовано до довільних обчислювальних циклів написаних мовами програмування С або С++.

Удосконалено:

Метод оцінки часу виконання тестових програм при використанні різноманітних методів розбиття на блоки, якій базується на автоматичному послідовному багаторазовому запуску тестових програм і фіксації отриманих результатів швидкодії, що можуть бути в подальшому проаналізовані.

Дискретний метод рою часток шляхом визначення коефіцієнтів методу, а саме – початкового коефіцієнту інерції, індивідуального та соціального коефіцієнтів, при яких пошук розмірів блоків розбиття в задачі прискорення швидкодії виконується швидше класичного методу.

Використання дискретного методу рою часток шляхом визначення кращих початкових даних для розташування часток рою, що зменшує число ітерацій для знаходження оптимального рішення.

Практичне значення одержаних результатів полягає в тому, що розроблений в дисертаційній роботі метод інтелектуального блочного розбиття може бути використаний для оптимізації програмного забезпечення написаного мовами програмування C або C++ широкого спектру застосування. Оскільки вказаний метод оптимізації використовує S2S перетворення, то безпосередньо алгоритм оптимізованої програми та використовуваний компілятор не мають значення. Запропонований метод оптимізації є універсальним відносно типу задач. Метод може бути використаний з будь-яким компілятором та апаратною платформою. Експериментальні дослідження виявили, що найбільш вагоме пришвидшення інструкцій досягається на апаратній платформі з великою кількістю кешу. На мікроконтролері, що не має кешу, прискорення не спостерігалось.

Особистий внесок здобувача. Усі основні положення та результати дисертаційної роботи отримані автором самостійно. У роботах, виконаних у співавторстві, автору належать такі результати: в праці [1] запропоновано методи вимірювання та виконано виміри часу виконання та енергоспоживання тестових програм на платформі Raspberry Pi 3 для різних методів розбиття; в праці [2] виконано виміри часу виконання та енергоспоживання тестових програм на платформі x64 для різних методів розбиття, введено поняття

коефіцієнту енергоефективності та отримано значення коефіцієнтів енергоефективності; в праці [3] запропоновано використання різноманітних розмірів блоків розбиття з метою визначення їх впливу на швидкодію тестових програм; в праці [4] проведено аналіз використання різноманітних блоків розбиття; в праці [5] визначено складний характер впливу параметрів розбиття та запропоновано напрямки для визначення таких параметрів; в праці [6] виконано дослідження та аналіз енергоефективності обчислень на різних апаратних платформах.

Апробація результатів дослідження. Основні положення та результати дисертаційної роботи доповідалися і обговорювалися на науково-технічній конференції молодих вчених та спеціалістів ІПМЕ ім. Г.Є. Пухова НАН України (м. Київ, 2016), на Міжнародній конференції Моделювання-2016 (м. Київ, 2016), на Міжнародній науково-практичній конференції Summer InfoCom Advanced Solutions 2016 (м. Київ, 2016), на Міжнародній науково-практичній конференції Winter InfoCom Advanced Solutions 2016 (м. Київ, 2016), на міжнародній конференції IEEE International Conference on Electronics and Nanotechnology (ELNANO) (м. Київ, 2017), на Зимовій школі ALIOT (м. Чернівці, 2018), на науково-технічній конференції молодих вчених та спеціалістів ІПМЕ ім. Г.Є. Пухова НАН України (м. Київ, 2018), на Міжнародній конференції IEEE International Conference on Dependable Systems, Services and Technologies (DESSERT-2018) (м. Київ, 2018), на Міжнародній конференції Моделювання-2018 (м. Київ, 2018), на Міжнародній конференції Reconfigurable Ubiquitous Computing 2018 (м. Дзівнув, Польща, 2018), на Міжнародній науково-практичній конференції комп'ютерні системи та мережні технології (м. Київ, 2019), на Міжнародній конференції IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T-2019) (м. Київ, 2019), на Науково-практичній конференції «Безпека енергетики в епоху цифрової трансформації» (м. Київ, 2019).

Публікації. За результатами виконаних досліджень опубліковано 16 наукових праць, серед яких: 1 стаття у періодичних наукових виданнях інших держав, 1 розділ в колективній монографії, що індексується міжнародною наукометричною базою Scopus, 4 наукових статті (у тому числі 3 статті у наукових фахових виданнях), 8 – у збірниках матеріалів міжнародних конференцій, з яких 3 індексуються міжнародною наукометричною базою Scopus та 2 у збірниках матеріалів науково-практичних конференцій.

Структура та обсяг дисертаційної роботи. Дисертація складається з анотації, вступу, чотирьох розділів, висновків, списку використаних джерел, що містить 155 найменувань і 6 додатків. Дисертація має 126 сторінок основного тексту, 74 рисунки, 15 таблиць, 35 сторінок додатків. Загальний обсяг дисертації становить 161 сторінку.

Подяки. Висловлюю щиру подяку науковому керівнику, доктору технічних наук, старшому науковому співробітнику Чемерису Олександру Анатолійовичу за слушні рекомендації, корисні поради та плідну співпрацю.

РОЗДІЛ 1. АНАЛІЗ ШВИДКОДІЇ ТА ЕНЕРГОСПОЖИВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ

1.1 Ефективність обчислень

Впродовж розвитку обчислювальної техніки апаратне та програмне забезпечення постійно вдосконалюється. Водночас, з екстенсивним розвитком застосовується і інтенсивний розвиток, а саме значна увага приділяється більш ефективному використанню наявних ресурсів. Наприклад, більш ефективне використання ресурсів центру обробки даних може суттєвим чином вплинути на енергоспоживання, тим самим на кількість електроенергії необхідної для системи кондиціонування і, як наслідок, на операційні витрати. Ефективніше використання мобільних пристроїв дозволяє довше працювати без підзарядки та застосовувати апаратні пристрої з кращими обчислювальними можливостями. Ефективне використання вбудованих систем дозволяє розширювати функціональні та сервісні можливості пристроїв [104].

Ефективність обчислень це багатофакторна характеристика. По-перше, це міра спроможності апаратного комплексу та компілятора реалізовувати програмний код високого рівня. По-друге, ефективність обчислень може бути визначена, як доля реальної обчислювальної потужності відносно пікової обчислювальної потужності, що доступна для даного апаратного забезпечення. По-третє, ефективність обчислень базується на ефективність алгоритмів, що використовуються [61]. Вказані фактори стосуються усіх обчислювальних систем – персональних комп’ютерів, серверів, мобільних пристроїв [25], [47], ПЛІС [109], контролерів [59], SoC [39], [78], вбудованих систем [24], [79], кластерів [117] і т. і.

Ефективність обчислень розглядається також в контексті об’єднання обчислювальних засобів. Наприклад у [103] наводяться оцінки ефективності різноманітних варіантів комплексування обчислювальних засобів в системі із визначеним складом апаратури. Виділимо такий показник як коефіцієнт зниження реальної продуктивності, що характеризує витрати продуктивності

обчислювальних засобів на організацію сумісної роботи комплексуємих ЕОМ або процесорів.

$$K_k = \frac{P}{\sum_{i=1}^{N} V_i}, \quad (1.1)$$

де V_i – ефективна продуктивність i-го обчислювального засобу при індивідуальному функціонуванні і обчислюванню визначеного класу задач;

P – те саме для системи в цілому,

N – кількість комплексуємих електронних обчислювальних машин або процесорів.

Ефективність реалізації деякого алгоритму або класу алгоритмів на ЕОМ майже цілком визначається співвідношенням структури вибраного алгоритму і структури ЕОМ [57], [73]. Це ж відноситься і до обчислювальних систем. При практичному використанні обчислювальних систем гнучкість логічної структури алгоритму, її здатність до перетворення набувають особливо важливого значення [26], [28], [29].

Для визначення ефективності реалізації алгоритмів на обчислювальних системах необхідно проаналізувати можливі структури алгоритмів [118], [119]. Умовно алгоритми поділяють на 3 групи: послідовні, паралельні та послідовно-паралельні [69]. Найбільш ефективно на обчислювальних системах реалізуються паралельні алгоритми [71], [72]. Розробляються методи автоматизації для максимально ефективного використання паралельних програм [107]. Варто відзначити, що наведено ділення алгоритмів на групи є достатньо умовним, оскільки детермінованість структури обчислювальної системи, надає можливість виконувати необхідні перетворення структур алгоритмів [124], у випадках, коли таке перетворення можливе. Останнім часом перетворення алгоритмів на паралельні набуває нового прогресу оскільки сучасні обчислювальні системи стають все більш багатоядерними.

1.2 Обґрунтування впливу ПЗ на швидкодію та рівень енергоспоживання обчислювальних пристроїв

Оптимізація програмного забезпечення є актуальною задачею для практичного застосування обчислювальних систем різного призначення. У загальному випадку, оптимізоване ПЗ споживає менше ресурсів для своєї роботи та/або виконується швидше. Параметр оптимізації в кожному випадку вибирається індивідуально. Як правило, це час виконання, розмір коду програми або розмір даних. Розглядаючи обчислювальну систему як апаратно-програмний комплекс, слід також брати до уваги його енергоспоживання в одиницю часу та повну енергію, необхідну для обчислення алгоритму. У цьому контексті оптимізація також надає можливість зниження загальної енергії обчислення, а також, в деяких випадках, можливість зниження енергоспоживання [30], [38]. Слід однак враховувати, що при більш ефективному використанні ресурсів процесора [58], при розпаралелюванні завдань енергоспоживання може зрости, при цьому загальна необхідна енергія обчислень може знизитися.

Для різних обчислювальних систем за призначенням можна виділити наступні ефекти від оптимізації:

– мобільні обчислювальні системи можуть мати більш компактні розміри або більш тривалий час автономної роботи;

– вбудовані обчислювальні системи та системи реального часу можуть мати більш широкий функціонал виконуваних дій [99];

– серверні обчислювальні системи можуть значно скоротити експлуатаційні витрати за рахунок зниження енергоспоживання власне серверів та систем охолодження та кондиціонування.

Таким чином, оптимізація ПЗ покращує різноманітні характеристики обчислювальних систем і тим самим підвищує ефективність застосування ПЗ для практичного застосування. Окремо варто відзначити методологію розроблення мікропроцесорів, що з самого початку ставить за мету низьке енергоспоживання обчислювальних систем [55].

Підвищення ефективності обчислень може бути досягнуто багатьма шляхами зважаючи на багатогранну визначеність ефективності. Звернемо увагу на енергоспоживання як складову частину ефективності обчислень [131], [132], [136], [137], [145], [146], [147], [148].

Оптимізація ПЗ з метою прискорення швидкодії – один з найбільш вживаних напрямків оптимізації. Більш швидке ПЗ може виконувати більше функціональних можливостей за той же час. У випадках, коли час виконання програми не настільки критичний, більш швидке завершення виконання програми дозволяє переводити мікропроцесор у режим зниженого енергоспоживання до наступного запуску програми.

Для прикладу розглянемо складові частини енергоспоживання для транзисторів, виконаних за сучасною технологією КМОН [77], [85], що широко застосовується для виготовлення інтегральних схем [92], [93]. Загалом виокремлюють 2 типи енергоспоживання на рівні КМОН структури – динамічне та статичне енергоспоживання [24]. В більш широкому сенсі виокремлюють 4 джерела енергоспоживання як зображено на Рисунку 1.1.

Рисунок 1.1 – Структура енергоспоживання КМОН технології

Один з параметрів, а саме динамічне енергоспоживання, з’являється тільки під час перемикання транзистору. Транзистор може бути як внутрішнім

регістром процесору, так і зовнішною пам'яттю. Загалом, динамічне енергоспоживання може досягати 80% у всьому енергоспоживанні пристрою. Якщо P_{dyn} це динамічна потужність, C – загальна електрична ємність ланцюгу, V_{dd} – діапазон зміни напруги, α – фактор активності, f – частота перемикань, то динамічну потужність можна представити у вигляді формули (1.2):

$$P_{dyn} = fCV_{dd}^2 \propto. \quad (1.2)$$

Відповідно до (1.2), зменшення енергоспоживання досягається зменшенням тих змінних, що входять в праву частину виразу. Зменшення ємності схем досягається технологічними засобами. Аналогічно, зменшення напруги живлення та частоти перемикань схеми призведуть до зменшення енергоспоживання. Це можна робити програмними засобами, але, зменшуючи частоту, зменшуємо продуктивність обчислень. Зменшення напруги має свої технологічні обмеження, оскільки зменшується завадостійкість. Таким чином, зменшення параметрів, що входять до формули, має своє технологічне та функціональне обмеження.

Найбільш перспективним є зменшення коефіцієнта активності схем, який безпосередньо залежить від програми, що виконується. При роботі комп’ютерних програм перемикання логічних рівнів відбуваються у будь-яких електронних вузлах обчислювальної системи – регістрах процесору, його кешу, зовнішньої пам’яті, шини даних, шини програм та шини керування. При зменшенні кількості перемикань досягається зменшення енергоспоживання всього пристрою [144].

Отже, енергетична ефективність пов’язана з обчислювальною. Скорочення кількості обчислень не тільки прискорює швидкодію, але ще й впливає на енергоспоживання [114].

Зменшення кількості перемикань можна досягнути шляхом вдосконалення програмного забезпечення. Таким чином, оптимізоване програмне забезпечення може підвищувати ефективність різними шляхами – через прискорення часу виконання програми і через зменшення енергоспоживання внаслідок цього.

Останніми роками важливе значення приділяються так званим «зеленим» обчисленням [60]. Зелені обчислення являють собою комплекс підходів, що стосуються безпечних для навколишнього середовища технологій обчислень та інформаційних технологій [66], [67]. Це наука та практика проектування, виготовлення, використання та утилізації комп'ютерів, серверів та їх підсистем. Цей підхід націлений на те, щоби використовувати обчислювальну техніку ефективно та з мінімальним або нульовим впливом на оточуюче середовище [74].

Зелені обчислення це не тільки міри, що спрямовані на зниження енергоспоживання [97]. Такий підхід використовують також на етапі проектування та розробки обчислювальних систем. Зелені обчислення також позначають зниження використання токсичних матеріалів, переробку, повторне використання обладнання. Таким чином, зелені обчислення це не тільки більш ефективне використання електроенергії. Це також комплексний підхід, що використовується на всіх етапах життя компонентів обчислювальної системи [5], [8].

Важливою частиною зелених обчислень є зелене програмне забезпечення [34], [35], [36], [37], [38]. Зелене програмне забезпечення це таке комп'ютерне програмне забезпечення, що може бути розроблено та використано ефективно з мінімальним впливом на навколишнє середовище або без впливу на нього. Беручи до уваги дані енергоспоживання компонентів сервера, можна прийти до висновку, що на кожен ват обчислювальних ресурсів витрачається декілька ват роботи супутнього обладнання. Зниження енергоспоживання за рахунок використання зеленого програмного забезпечення знижує тим самим і додаткове енергоспоживання усієї обчислювальної системи. Отже, зниження енергоспоживання на рівні виконання програмного забезпечення має мультиплікативний ефект та має практичний вплив на сумарне енергоспоживання. У той же час, розробка більш ефективного програмного забезпечення потребує кваліфікованих зусиль математиків, алгоритмістів та розробників програмного забезпечення.

Предметом розгляду даної роботи являються методи оптимізації програмного забезпечення за часом та енергоспоживанням.

1.3 Рівні оптимізації ПЗ

Більшість алгоритмів і способів оптимізації апаратно залежні, тобто мають різну ефективність на різному апаратному забезпеченні. Прикладом може бути автоматичне розпаралелювання послідовних алгоритмів і створення нового процесу обчислення для паралельної програми конкретної обчислювальної архітектури. Також варто відмітити такий самий функціонал для компіляторів, які за своїм призначенням жорстко прив’язані до конкретної апаратної платформи.

Інший підхід являє собою практично незалежний від архітектури обчислювальної системи процес. У випадку S2S перетворення на виході процесу оптимізації виконується зворотна генерація трансформованої програми мовою високого рівня. Трансформована програма бути як послідовна, так і розпаралелена на багатопроцесорних комп’ютерах. Отриманий оптимізований вихідний код буде мати різну ефективність на різних апаратних платформах, проте сам процес S2S перетворення використовує апаратно незалежні алгоритми і методи.

Розглядаючи практичні підходи оптимізації, слід виділити їх досить велику кількість і різну сферу застосування. Зокрема, в залежності від рівня застосування оптимізаційні підходи можна розділить на такі:

- оптимізація на рівні алгоритму;
- оптимізація на рівні команд;
- оптимізація на рівні операцій процесора.

Використовується трьохрівнева ієрархія методів оптимізації КП, що дозволяє найбільш повно використовувати всі наявні ресурси для покращення обраних параметрів програм. Рівні методів оптимізації наведено на Рисунку 1.2.

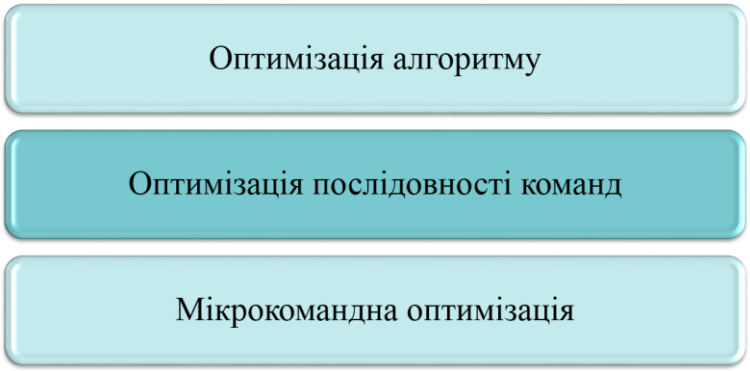

Рисунок 1.2 – Рівні оптимізації програмного забезпечення

Алгоритмічна оптимізація виникає на першому етапі розробки ПЗ і містить в собі вибір найкращого підходу щодо вирішення поставленого завдання з урахуванням апаратної платформи, доступних ресурсів та інших обмежень конкретної реалізації. Більшість алгоритмів цифрової обробки сигналів, криптографії, чисельного аналізу, математичного моделювання та багатьох інших спрямовані саме на максимальну швидкодію з урахуванням мінімального споживання ресурсів. Оптимізація на рівні алгоритму являє собою вдосконалення загальних методів обчислень. Наприклад, швидке перетворення Фур'є це вдосконалення дискретного перетворення Фур'є, яке має обмеження на допустимі вхідні дані. Оптимізація цього рівня виконується зазвичай математиком і розробником програмного забезпечення. Розробка більш досконалих і ефективних алгоритмів є предметом розгляду наукових, дослідницьких та виробничих центрів.

Оптимізація реалізації алгоритму в командах мови високого рівня являє собою задачу виконання деякого визначеного набору дій безпосередньо в командах, зрозумілих компілятору. Більша частина цього етапу оптимізації покладається на розробника ПЗ. Саме від його умінь і навичок залежить те, який код буде представлений компілятору. У деяких випадках цей етап є найбільш вузьким місцем в ланцюгу оптимізацій, оскільки вимагає високої кваліфікації та навичок розробника.

Оптимізація на рівні процесора це процес перетворення операторів мови високого рівня деяким кодом, якій може виконувати процесор. Цю роботу

зазвичай виконує компілятор. Оптимізація реалізації мови високого рівня в машинних кодах процесора працює безпосередньо з вихідними кодами ПЗ. Цей етап, як правило, виконується компілятором, який, з огляду на всі ресурси апаратної частини обчислювальної системи, генерує бінарний код. При цьому цільова функція оптимізації, як правило, задається через набір опцій компілятора. У випадках, коли код програми пишеться на асемблері, цей етап оптимізації також стає відповідальністю розробника ПЗ.

Розглядаючи етапи оптимізації ПЗ, слід зазначити етап оптимізації реалізації алгоритму в командах мови високого рівня. На цьому етапі здійснюється перехід від деякої математичної абстракції до безпосередньо вихідного коду програми. Оптимізація цього етапу здійснюється в ручному режимі з використанням профілювання та різних відлагоджувальних засобів. Така оптимізація, як було згадано вище, значно залежить від кваліфікації розробника ПЗ. Тому існує певна зацікавленість в додаткових засобах оптимізації ПЗ. Підходи щодо оптимізації цього етапу надають компілятори, вони здатні в значній мірі змінювати порядок коду, модифікувати, розгортати код тіла циклу. У той же час компілятори можуть не охоплювати всього спектра можливих оптимізаційних підходів. Оптимізація на рівні команд є використовуваною і перспективною технологією, оскільки послідовність операцій може змінюватися у широкому діапазоні залишаючи при цьому програмне забезпечення семантично еквівалентним. Задачі оптимізації на рівні команд досліджуються і вирішуються практично з моменту створення компіляторів. Насправді кожен компілятор не просто перетворює команди мови високого рівня в машинні коди чи інші коди, що виконуються. Компілятор також виконує зміни в послідовності виконання команд. Це може бути як відносно прості зміни, наприклад розгортання малого циклу в послідовність безумовних операцій, так і складні операції – перетворення декількох арифметичних чи логічних операцій в інші, зміна порядку обчислень, балансування виконання великих обчислювальних задач на процесорних ядрах, тощо. Розробник програмного забезпечення значною

мірою відповідальний за оптимізацію на цьому кроці. Фактично ефективність роботи компілятора в багато чому залежить від оптимізації на даному етапі. Документація компілятора не завжди відображає методи і засоби, які використовує компілятор на даному етапі.

В даній роботі особлива увага приділяється оптимізації послідовності команд, що дозволяє не змінювавши алгоритм програми та компілятор досягати покращення результатів роботи програм.

Беручи до уваги можливості щодо ручної оптимізації коду програм, слід зазначити, що така оптимізація досить тривала і для великих проектів може займати досить значний час. При цьому ефективність внесення нових оптимізаційних підходів буде знижуватися. Таким чином, постає дилема, скільки часу витрачати на оптимізацію коду з урахуванням того, що подальша оптимізація буде все менше позначатися на фінальному результаті.

Окремим напрямком оптимізації ПЗ є використання паралельних обчислень [53]. Використання паралельних обчислювальних систем дозволяє значно збільшити продуктивність обчислень [105]. Однак, ефективне використання паралельних обчислень можливе тільки при розробці ефективних паралельних програм. Очевидно, що створення такого програмного забезпечення залежить від рівня програміста. І навіть програми, створені програмістом високого рівня, можуть бути оптимізовані для отримання більш високої швидкодії, зокрема, за рахунок виявлення ділянок коду, які можливо виконувати паралельно.

При розробці засобів компілювання комп'ютерних програм значна увага приділяється засобам оптимізації коду. Це стосується як послідовних програм, так і паралельних. Проте, засобам автоматичного розпаралелювання алгоритмів приділяється особлива увага.

На основі деякого логічного подання алгоритмів або на структурному рівні у вигляді деякої діаграми, або на системному рівні, використовуючи програмний код високого рівня, засоби автоматичного розпаралелювання складаються, в загальному вигляді, з трьох стадій обробки алгоритму. На

першій стадії проводиться аналіз алгоритму, на другій розробляються правила трансформації даного алгоритму, а на третій – генерація коду для паралельної обчислювальної системи.

Для першої стадії – аналізу алгоритму, з метою оптимізації або розпаралелювання, потрібно зібрати необхідну інформацію про структуру програми, а саме, про команди програми, що виконуються, і їх взаємодії між собою. В якості моделі для виявлення паралелізму і можливостей оптимізації програмних додатків паралельних комп'ютерів в даній роботі обраний граф залежностей.

Одним з напрямків оптимізації на рівні команд є клас оптимізацій, що використовують S2S перетворення. Такий підхід має декілька переваг:

- оптимізації S2S перетворення є апаратно незалежними;
- оптимізації S2S перетворення доповнюють інші оптимізаційні методи компілятора;
- оптимізації S2S перетворення є трансформацією вихідного коду і можуть бути описані математичною моделлю;
- розробка алгоритму S2S перетворення значно простіша чим розробка нового компілятору;
- можливість трансформації вихідного коду як в ручному, так і в деяких випадках, в автоматичному режимах;
- відносна швидкість і легкість тестування та впровадження.

Недоліком оптимізацій S2S перетворення можна назвати необхідність використовувати декілька програмних засобів, що виконують різні етапи оптимізації: програма для створення математичної моделі по вихідному коду, програма трансформації моделі, програма відновлення вихідного коду з моделі. Останнім часом з’явилися програмні продукти, що включають або використовують всередині низку програмних пакетів, що можуть виконувати трансформацію вихідного коду в автоматичному режимі [5], [6], [7], [30], [31], [32].

Оскільки більшість часу на обчислення витрачається в обчислювальних циклах над великою кількістю однотипних даних, то розпаралелювання і оптимізація саме обчислювальних циклів має найбільший потенціал для оптимізації програмного забезпечення.

1.4 Постановка задачі оптимізації програмного забезпечення

Оптимізація програмного забезпечення це процес модифікації або трансформації вихідного програмного коду з метою покращення цільової функції, що оптимізується, при якому результат виконання програми не змінюється. Цільовою функцією оптимізації ПЗ можуть бути один або декілька характеристик програми, наприклад скорочення часу виконання програми, зменшення коду програм або даних, мінімізація енергозатрат, зменшення кількості операцій вводу-виводу та інші. Процес оптимізації спрямований безпосередньо на покращення заздалегідь обраного параметру. В процесі оптимізації інші параметри виконання програми можуть погіршуватися. Наприклад, при зменшенні часу виконання може збільшуватися код програми.

Реалізація процесу оптимізації програмного забезпечення здійснюється за допомогою послідовності перетворень, алгоритмів, методів, що аналізують програму та змінюють її для отримання семантично еквівалентного варіанта, що більш ефективний з точки зору якогось набору цілей оптимізації. Деякі проблеми оптимізації коду є NP-повними чи навіть такими, що не можуть бути розв’язаними [112]. На практиці багато з них вирішуються евристичними методами, що дають результат за задовільний час обробки вихідного коду програми.

Оптимізацію ПЗ виконують задля зменшення деякого цільового параметру.

Розглядаючи оптимізацію програмного забезпечення як процес спрямованого застосування або перебору доступних методів оптимізації з визначеним набором параметрів можна описати формулою (1.3):

$$\begin{cases} f(M_l, P_k) \to min \\ l = 1..L \\ k = 1..K \\ P_{kmin} \le P_k \le P_{kmax} \end{cases} \quad (1.3)$$

де f – цільова функція оптимізації ПЗ. Зазвичай, цільовою функцією оптимізації може бути один або декілька параметрів, що потребують покращення – зменшення часу виконання, енергоспоживання, розміру пам'яті програм, даних тощо. Дана робота розглядає зменшення часу виконання програм як цільову функцію оптимізації ПЗ.

M_l – множина можливих методів оптимізації;

l – індекс метода оптимізації. $l = 1 \ldots L$, де L – число доступних методів оптимізації.

P_k – значення параметрів методів оптимізації.

k – індекс параметра метода оптимізації. $k = 1 \ldots K$, де K – число параметрів метода оптимізації M_l.

Вираз (1.3) означає, що для оптимізації конкретної частини коду потрібно використати декілька оптимізаційних методів зі своїми оптимізаційними параметрами. Вибір методів оптимізації може бути обраний заздалегідь (визначений явно або встановлений за замовчуванням) або визначений в результаті оцінки коду за допомогою різноманітних метрик. Зазвичай розробники використовують ті методи оптимізації, що надаються компілятором за замовчуванням або використовують спрощену конфігурацію методів оптимізації. Наприклад, ключі оптимізації за часом можуть мати декілька можливих значень: –O0, –O1, –O2, –O3. При цьому довідкова інформація компілятора не розкриває всього переліку оптимізаційних методів, що використовуються для кожного конкретного ключа оптимізації. Деякі з методів оптимізації є параметричними і мають один або декілька регульованих параметрів, що також впливають на результат оптимізації.

Таким чином, процес оптимізації програмного забезпечення має складнощі через те, що вибір набору конкретних методів і їх конкретних

параметрів невизначений заздалегідь і вимагає додаткових експериментів для перевірки ефективності та їх комбінацій. Через архітектурні особливості та обмеження підбір методів та їх параметрів унікальний для кожного практичного випадку.

Основними проблемами оптимізації КП є такі:

- Глибока оптимізація програмного забезпечення потребує високої кваліфікації розробника програмного забезпечення;
- Більшість алгоритмів задана для однопотокового процесу, в той час як сучасні мікропроцесори багатоядерні, і адаптація одноядерних програм для ефективного використання на багатоядерному мікропроцесорі потребує складної перебудови алгоритму;
- Наявність великої кількості методів оптимізації та їх параметрів;
- Деякі програми, такі як декодування звуку, відео, обробка пакетів даних згідно мережевих протоколів, бездротової передачі даних, супутникового зв'язку, мають бути обчислені за визначений короткий проміжок часу;
- Наявність задач, в яких виникає необхідність виконання великої кількості операцій над великою кількістю вхідних даних;
- Використання малопотужного апаратного забезпечення, що має дуже обмежені обчислювальні ресурси.

Вивчаючи задачу автоматичного розпаралелювання, приходимо до висновку, що в циклічних фрагментах алгоритмів зосереджена більша частина обчислювальної роботи [42], [44], [46]. Ефективність того чи іншого підходу до автоматичного розпаралелювання програм визначається знаходженням кількості та виду залежностей в алгоритмах. Основою програм розпаралелювання є граф залежностей, представлений у вигляді тієї чи іншої моделі. Те, наскільки точно і повно буде побудований граф залежностей, визначає ефективність результуючої розпаралеленої програми. Після побудови графа залежностей алгоритму вирішуються завдання розподілу обчислень на процесори обчислювальної системи.

При цьому об'єктами вирішення цих задач є три види просторів. Кожен простір може розглядатися як точки одновимірної або багатовимірної структури.

1. Простір ітерацій (ітераційний простір) є множиною динамічно виконуваних в процесі обчислень екземплярів ітерацій, тобто множиною комбінацій значень, які приймають індекси циклів. Простір ітерацій є множиною ітерацій, ідентифікатори яких задаються значеннями, що зберігаються в індексних змінних. Гніздо циклів глибиною *n* (тобто *n* вкладених циклів) має *n* індексних змінних і, відповідно, моделюється *n*-мірним простором. Простір ітерацій обмежено нижньою і верхньою границями індексів циклів.

2. Простір даних являє собою множину елементів масиву, до яких виконується звернення. Йдеться про масиви даних (вектори, матриці тощо), представлених індексними змінними. У загальному випадку, на логічному рівні абстракції, масив даних має розмірність *n*. Фізично в пам'яті машини він представляє лінійний одновимірний масив комірок пам’яті.

3. Простір процесорів є множина процесорів в обчислювальній системі. Як правило, процесори пронумеровані. Вважаємо, що простір процесорів – *n*-мірний масив, що залежить від архітектури обчислювальної системи. Так, для обчислювальних систем зі спільною пам'яттю, до яких належать багатопотокові архітектури, простір процесорів є одновимірним масивом даних.

Фактично потрібно визначити ефективне об'єднання цих трьох просторів для конкретних архітектур обчислювальних систем. Особливістю даного підходу є ітеративний характер процедури розпаралелювання, обумовлений процедурою трансформації і необхідністю уточнення результатів розпаралелювання за рахунок перевірки паралелізму алгоритму.

Загальна процедура розпаралелювання представлена на Рисунку 1.3. Тут під скануванням розуміємо процедури лексичного і синтаксичного аналізу, які забезпечують інформацією про змінні в програмі [22]. Результатом роботи

блоку Аналіз є фактично побудований граф залежностей програми, який далі методами планування обчислень розбивається на окремі блоки з урахуванням балансування обчислювального навантаження і мінімізації пересилань даних між цими блоками.

Після цього проводиться генерація вихідного тексту мовою високого рівня, який може бути оброблений стандартними засобами компіляції для відповідної обчислювальної системи. Такий підхід, що називається «трансформацією програм», визначає те, що на вхід подається програма мовою високого рівня і на виході також отримуємо програму на тій же мові високого рівня, яка містить оператори, що забезпечують її паралельне виконання.

Практично потрібно розв’язати такі задачі: поетапно застосувати перетворення, забезпечуючи еквівалентність обчислень після кожного кроку перетворень; визначити найбільш трудомісткі цикли; провести аналіз залежностей циклів і використання даних; визначити перетворення циклу, які приведуть до більш ефективного виконання програми на паралельній ОС в порівнянні з послідовною реалізацією; поетапно застосувати перетворення, забезпечуючи еквівалентність обчислень після кожного кроку перетворень.

Аналіз існуючих методів і програмних засобів визначає, що вони 1) не можуть розпаралелити чимало мовних конструкцій, що мають паралелізм; 2) є досить складними в реалізації; 3) вимагають багато часу для розпаралелювання (особливо це стосується точних методів), не гарантуючи при цьому задовільний результат.

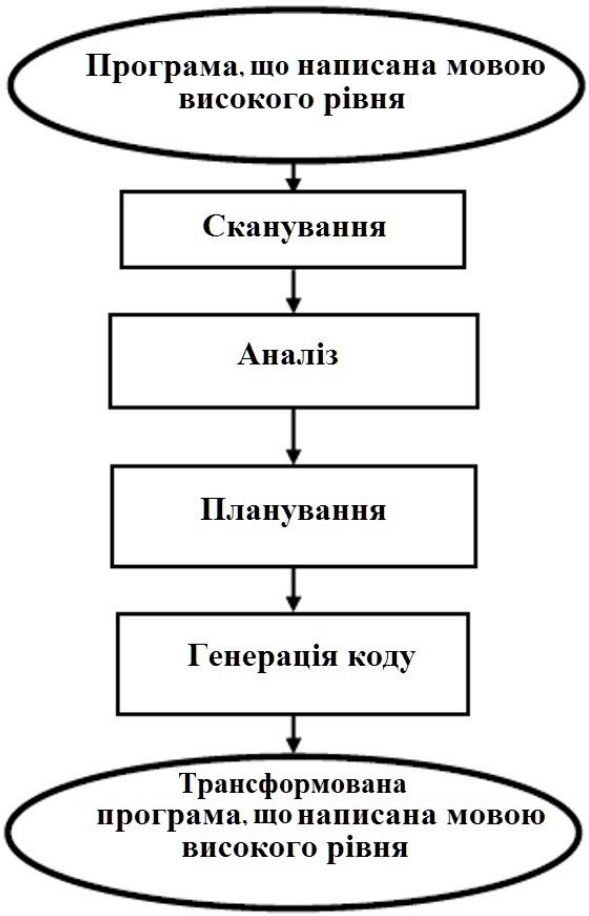

Рисунок 1.3 – Загальна блок-схема процесу розпаралелювання програм мовою високого рівня

Таким чином, завданням програми, що виконує розпаралелювання, є розробка загального підходу до розпаралелювання програм, створення ефективних методів трансформації програм в частині циклічних фрагментів алгоритмів, реалізація даних методів у вигляді швидкодіючих програмних засобів розпаралелювання початкового коду програм, написаного мовою високого рівня.

1.5 Методи оптимізації обчислювальних циклів

Виходячи з завдання оптимізації за часом обчислення, неважко помітити, що більшу частину часу виконання програм процесор проводить в досить невеликих областях пам'яті програми, а саме в обчислювальних циклах. Виконуючись багаторазово, такі ділянки коду програм при малих розмірах можуть виконуватись переважну частину часу роботи програми. Отже, саме

обчислювальні цикли являють собою найбільш ефективне місце для оптимізації за часом виконання.

Основними підходами щодо оптимізації циклів є їх розпаралелювання і модифікація тіла циклів [82], [83], [129], [130], [133], [135]. Розпаралелювання циклів найбільш ефективно в багатоядерних системах [139], [143] в тих випадках, коли обчислювані дані не використовуються як вхідні на наступних ітераціях циклу [95], [96]. В іншому випадку виділяють залежності даних [76], які можуть значно впливати як на розпаралелювання, так і на інші модифікації циклів. Відзначаючи потенційно значний виграш за часом виконання програм від розпаралелювання, слід також мати на увазі, що не весь код може бути розпаралелюваним [100], [101], [102]. Відповідно до закону Амдала [103], прискорення, яке може бути теоретично досягнуто на обчислювальній системі з декількох процесорів p, в порівнянні з однопроцесорним рішенням не буде перевищувати величини:

$$S_p = \frac{1}{a+\frac{1-a}{p}}\ , \tag{1.4}$$

де S_p – відносне прискорення обчислень,

p – число процесорів в системі,

a – частина обчислень, яка може бути розрахована тільки послідовно.

Як випливає з формули, частина обчислень, яка не може бути розпаралелена, матиме ключовий вплив для обчислювальних систем з великою кількістю ядер.

Розглядаючи можливі способи модифікації обчислювальних циклів слід відзначити велику кількість існуючих підходів і технологій перетворення обчислювальних циклів [91], [142]. Головні підходи по модифікації циклів наведено нижче:

- *Розпаралелювання (parallelization).* Розбиття всіх операцій обчислювального циклу на частини меншої розмірності та одночасне виконання всіх частин на декількох обчислювальних ядрах;

- *Векторизація (vectorization)*. Групування ідентичних операцій в єдиний вектор, що може бути виконаний спеціальними командами процесора за один такт;
- *Розбиття (fission/distribution)* всіх операцій циклу на декілька менших циклів тієї ж розмірності;
- *Злиття (fusion/combining)* декількох циклів в один [68], [140];
- *Зміна послідовності (interchange/permutation)* циклів передбачає зміну порядку виконання змінних циклу;
- *Зворотне обчислення (reversal)* циклу змінює напрямок ітерування циклу;
- *Програмний конвеєр (software pipelining)* змінює послідовність виконання операцій всередині циклу задля зниження затримки між блоками процесору;
- *Метод розбиття на блоки (tiling)* виконує розбиття ітераційного простору на частини меншого розміру;
- *Інверсія перевірки умови виходу з циклу (inversion)* – зміна циклу while на цикл do-while;
- *Винесення розрахунків (loop-invariant code motion)* – винесення розрахунку кількості ітерацій циклу зовні циклу;
- *Реорганізація циклу (skewing)* – перетворення доступу до масиву у внутрішньому циклі так, щоб залишалися залежності тільки відносного зовнішнього циклу;
- *Зменшення розмірності виконання (splitting/peeling)* – розбиття циклу на кілька інших, ітерованих по частинах;
- *Винесення перевірок (unswitching)* – переміщення умови всередині циклу зовні циклу.
- *Розгортання (unrolling)* циклу виконує код циклу послідовно, без операторів циклу.

Наведені методи в деяких випадках можуть покращувати результат у разі сумісного використання. Деякі з методів, наприклад злиття і розбиття циклів, є взаємно виключними.

Вказані методи зазвичай використовуються компілятором, при цьому розробнику не надається інформація щодо використаних методів. Розробник програмного забезпечення у явному виді може тільки увімкнути розгортання обчислювального циклу.

Методи оптимізації обчислювальних циклів мають різну ефективність на різних обчислювальних циклах. Підбір конкретних методів задля найкращого результату вимагає розуміння структури циклу, його залежностей, зав’язків та особливості роботи пам’яті. Ефективність методів може бути достеменно оцінена на конкретному прикладі тільки шляхом верифікації на конкретному програмному коді і конкретному апаратному забезпеченні.

В даній роботі найбільшу увагу приділено методу розбиття обчислювальних циклів на блоки та розпаралелюванню.

1.6 Оптимізаційні можливості сучасних компіляторів

Оскільки оптимізація має значну важливість, компілятори мають в своєму складі різні модулі оптимізації, які активуються параметрами командного рядка [98], [121]. У той же час, з’явилися пакети програмного забезпечення, які, не виконуючи безпосередньо компіляції, виконують, згідно закладеному в них підходу, S2S перетворення, вихід якого потім може бути зкомпільовано стандартними засобами. Такий підхід дозволяє команді розробників оптимізатора сконцентруватися на реалізації безпосередньо алгоритму оптимізації, а не займатися повторною розробкою власне компілятора.

Паралелізація однопоточної програми полягає в розподілу множини операцій з даними, що розміщені в пам’яті, на множину процесорних елементів деякої конкретної топології. Мультипроцесорна комп’ютерна

система має розв’язати задачу балансування процесорів для зменшення числа невикористаних процесорів. Такий підхід призводить до зменшення часу виконання програми. Іншою задачею зростання ефективності є мінімізація передач даних поміж вузлами мультипроцесорної комп’ютерної системи. Розв’язання призводить до більш ефективного розпаралелювання.

Перетворювання обчислювальних циклів, таких як розбиття циклів та злиття циклів може значно спростити паралелізацію та підвищити її ефективність. Паралелізація найефективніша в обчислювальних циклах з великою кількістю ітерацій. Більша кількість ітерацій робить можливим балансування навантаженням з урахуванням наявності більшої кількості задач для розподілення поміж потоків.

Ручна оптимізація є копітким і тривалим процесом. В загальному випадку неможливо стверджувати, що досить велике ПЗ оптимізоване на 100%. Завжди будуть ділянки, незначні щодо всього обсягу коду, де ще може бути застосований якийсь спосіб оптимізації. Виходячи з цього, оптимізація застосовується в тих ділянках коду, де вона матиме найбільший вплив на кінцевий результат. Дослідження на реальних задачах показують, що найбільш часто використовувані ділянки кодів – обчислювальні цикли. Основна частина часу виконання програми витрачається саме в них. При цьому, чим більше вкладених циклів, тим, відповідно, частіше виконується внутрішній цикл. Таким чином, основні зусилля по оптимізації швидкодії слід докладати саме до обчислювальних циклів.

Сучасні компілятори мов високого рівня пройшли великий шлях розвитку від примітивних трансляторів вихідного коду у бінарний код до надзвичайно складних систем компіляції. Використовуючи інформацію про апаратні можливості цільових процесорів, глибокий аналіз залежностей коду та власне аналізатор вихідного коду сучасні компілятори спроможні генерувати дуже швидкий та ефективний код. Як правило, для кожної з мов високого рівня існує декілька достатньо розповсюджених сучасних компіляторів і деяка кількість таких компіляторів, що не оновлюються або з

самого початку були створені для високоспеціалізованої апаратної платформи. Розробники програмного забезпечення використовують компілятор, що є стандартним для середовища розробки або використовують такий, що вже зарекомендував себе як більш ефективний.

Наприклад, для мови програмування C++ використовують Microsoft Visual C++, Intel C++, GNU C++, G++, Builder C++ та інші компілятори. Компілятори виконують внутрішню оптимізацію коду за власними алгоритмами та методами, які, вочевидь, є різноманітними для різних виробників. Розробник програмного забезпечення може впливати на результати компіляції через ключі компіляції, що подаються як параметр до кожного файлу. Незважаючи на різноманітність компіляторів, основні ключі компіляції для різних компіляторів тієї ж мови програмування можуть збігатися. В середовищах розробки ключі компіляції задаються в графічному меню.

Компілятор надає лише грубі можливості для оптимізації коду. Розробник може впливати лише декількома ключами на оптимізаційні параметри. При цьому опис компілятора не розкриває безпосередніх алгоритмів, що використовуються компілятором в своїй роботі. Також не розкриваються методи оптимізації, що використовуються за замовчуванням. Таким чином, маючи потенційні великі здатності та гнучкість у конфігурації, компілятор не надає можливість безпосереднього керування усіма своїми параметрами.

Серед найбільш популярних та розповсюджених для мови програмування C++ варто виділити такі опції компілятора:

- рівень оптимізації коду;
- рівень оптимізації програм;
- пріоритет швидкого коду або малої програми;
- міжпроцедурні оптимізації;
- вибір набору інструкцій;
- паралелізація;

- розгортання циклів.

Проведено дослідження щодо впливу опцій компіляції компілятора Intel C++ на час виконання двох тестових програм. Спочатку було виміряно час виконання програми при зміні опцій компіляції по одній, а потім підібрано набір опцій при яких час виконання був найменший. Отримані дані наведені в таблицях 1.1 – 1.3.

Таблиця 1.1 – Результати роботи компілятора для поелементного KIX фільтра

Опція компілятора	**Значення опції**	**Отримане число тактів**	**Зміни відносно початкової конфігурації, %**
Optimization	Ox	306	-12.3
	O3	340	-2.6
Inline Function Expansion	Only Inline	303	-13.2
	Any	257	-26.4
Favor Size or Speed	Speed	389	11.5
Interprocedural Optimization	Single File	340	-2.6
	Multi File	275	-21.2
Loop Unrolling	Compiler	500	43.3
Parallelization	Yes	281	-19.5
Enable Enhanced Command Set	CORE-AVX2	405	16.0

Таблиця 1.2 – Результати роботи компілятора для блочного KIX фільтра

Опція компілятора	**Значення опції**	**Отримане число тактів**	**Зміни відносно початкової конфігурації, %**
Optimization	Ox	38550	10.7
	O3	38093	9.4
Inline Function Expansion	Only Inline	38090	9.4
	Any	15059	-56.8
Favor Size or Speed	Speed	37699	8.2
Interprocedural Optimization	Single File	37536	7.8
	Multi File	25432	-27.0
Loop Unrolling	Compiler	42781	22.8
Parallelization	Yes	37014	6.3
Enable Enhanced Command Set	CORE-AVX2	33268	-4.5

Таблиця 1.3 – Набір опцій для найшвидшої обробки блочного KIX фільтра

Набір опцій, при яких отримано найшвидший час виконання компілятора	**Отримане число тактів**	**Зміни відносно початкової конфігурації, %**
Optimization: O3 Inline Function Expansion: Any Suitable Favor Size or Speed: Neither Interprocedural Optimization: Multi File Loop Unrolling: Compiler Decide Parallelization: No Enable Enhanced Command Set: CORE AVX2	7004	-79.9

Як слідує з отриманих даних, вибір опцій компіляції, що призводить до мінімального часу виконання програми, не є очевидним. Варто відзначити, що сумісне використання декількох опцій, що можуть пришвидшити програму, у найкращому випадку дає синергетичний ефект.

Процес оптимізації на рівні компілятора це найнижчий рівень оптимізації програм. Разом з цим рівнем оптимізації можуть використовуватися і перші два: високорівнева оптимізація алгоритму та оптимізація послідовності команд.

1.7 Висновки до Розділу 1

В розділі розглянуто сучасний стан проблеми оптимізації та розпаралелювання ПЗ, визначено задачі, що постають при оптимізації ПЗ, проаналізовано методи, які використовують для автоматичного розпаралелювання циклічних частин алгоритмів. Розпаралелювання послідовних програм має на увазі відображення множини операцій над даними, розміщеними в пам'яті, на множину процесорних елементів певної топології.

Оптимізація програмного забезпечення це така зміна вихідних кодів програмного забезпечення, при якій досягається покращення по одному або декільком оптимізованим параметрам без зміни результатів обчислення програми. Як правило, параметри, що оптимізуються – час виконання програми, зменшення пам’яті даних або пам’яті програм, зменшення енергоспоживання.

Розглянуто циклічні частини алгоритмів та методи, які використовуються при оптимізації таких частин з метою прискорення швидкодії.

В розділі зазначено складний процес ручної оптимізації та розпаралелювання коду, що вимагає високої кваліфікації розробника та витрат часу. З цього випливає необхідність у використанні автоматичних та

автоматизованих систем оптимізації коду, що дозволяють знизити кваліфікацію розробника та час процесу оптимізації.

В розділі наведено результати оптимізації ТП за допомогою вбудованих засобів компілятора. При використанні великого набору оптимізаційних методів отримано суттєве пришвидшення ТП. Водночас вказано на прихованість повного переліку використаних методів оптимізації, що використовує компілятор.

Метою роботи є підвищення ефективності функціонування мікропроцесорних систем за рахунок розпаралелювання та оптимізації програмного забезпечення.

За наявності великого вибору методів оптимізації випливає головна мета дисертаційної роботи – покращення роботи існуючого методу оптимізації ПЗ.

РОЗДІЛ 2. МЕТОДИ ОПТИМІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз сучасних наукових підходів та методів оптимізації

Як було зазначено у Розділі 1, підходи щодо оптимізації циклів досить великі та різноспрямовані. Компілятори давно використовують велике різноманіття оптимізаційних методів явно через опції компіляції або приховано від розробника. У той же час слід зазначити, що не всі із зазначених технік оптимізації однозначно підвищують продуктивність всередині обчислювальних циклів. Деякі з цих технік не сумісні або навіть взаємно зворотні по виконуваних дій.

Перш ніж проводити аналіз підходів до оптимізації програмного забезпечення, надаємо декілька визначень ключових понять, на яких базуються методи розпаралелювання та оптимізації програмного забезпечення [141], [56].

Далі наведемо основні визначення.

Визначення. Між двома операторами S1 і S2 існує залежність, якщо вони обидва звертаються до однієї і тієї ж чарунки пам'яті, і, принаймні, одне з цих звернень є запис [103].

Відомо, що в програмах виділяють чотири типи залежностей, пов'язаних з доступом до одних і тих же чарунок пам'яті.

Розглянемо два оператора S_i і S_j ($S_i \prec S_j$) і відповідно до [65] визначимо типи залежностей в програмі. Тут використовується знак '$\prec$', який визначає лексикографічний порядок операторів S_i і S_j, тобто оператор S_i передує в програмі оператору S_j. Обидві команди виконують операції читання-запис з пам'яттю комп'ютера.

Визначення. Лексикографічний порядок (впорядкована послідовність) n -мірного декартового добутку [152].

Лексикографічний порядок (впорядкована послідовність) n-мірного декартового добутку $A^n = A \otimes A \otimes ... \otimes A$ визначається наступним чином: якщо a=(a_1, a_2, … , a_n) та b = (b_1, b_2, …, b_n), то a $\prec$ b, якщо $a_1 < b_1$ або $a_1=b_1$, $a_2=b_2$, …, $a_k = b_k$, та $a_{k+1}<b_{k+1}$, де $1 \le k \le n-1$.

Залежність за даними *(Data – Flow Dependence)* між операторами S_i та S_j ($S_i \prec S_j$) виникає в тому випадку, коли оператор програми S_j зчитує дані, які обчислюються в S_i.

S_i : $X = F_1(In(S_i))$;

S_j : $Y = F_2(In(S_j))$; $X \in In(S_j)$ & $j \ge i+1$;

де $In(S)$ – множина вхідних змінних оператора S. Залежність за даними визначається так званим порядком «запис перед читанням» [62.]. Відповідно до позначень, які використовуються в [6], напишемо дану залежність як $S_i \delta S_j$, що інтерпретується як «читання змінної в S_j залежить від запису цієї змінної в S_i».

Залежність по виходу (*Output Dependence*) двох операторів S_i та S_j ($S_i \prec S_j$) виникає в тому випадку, коли запис однієї і тієї ж змінної здійснюється в обох операторах, тобто S_i: $X = F_1(In(S_i))$;

S_j : $X = F_2(In(S_j))$; $j \ge i+1$.

Ця залежність запобігає від використання при виконанні будь-якого поточного оператора неправильних значень X. Позначається залежність по виходу як $S_i \delta^0 S_j$.

Антизалежність (*Antidependece)* між операторами у програмі S_i і S_j ($S_i \prec S_j$) виникає в тому випадку, якщо читання змінної з пам'яті проводиться раніше, ніж її запис в пам'ять комп'ютера, тобто S_i : $X = F_1(In(S_i))$; $Y \in In(S_i)$

S_j : $Y = F_2(In(S_j))$; $j \ge i+1$.

Антизалежність запобігає зміну Si до виконання Sj, що може привести до невірних значень. Записуємо цю залежність у вигляді $S_i \delta^{-1} S_j$.

Залежність четвертого типу є залежність по управлінню. Вона визначається операторами програми, які змінюють порядок виконання

операторів в залежності від деякої умови. Так, наприклад, у фрагменті коду мовою С

```
S1:    if( x == 0 )
S2:            y = ToDo( arg );
```

Таким чином, в програмах визначені наведені вище типи залежностей між операторами. Ці залежності характерні для будь-яких ділянок програм [138].

Розглянемо вкладені обчислювальні циклі, кожен з яких визначений своєю індексною змінною. Множина індексних змінних визначає індексний вектор вкладених циклів $I=(I_1,I_2,..,I_n)$.

Для будь-якого циклу, в якому індекс циклу I змінюється від значення L до U з кроком S, номер ітерації i дорівнює значенню $(I - L + S) / S$, де I – це значення індексу для цієї ітерації.

Для вкладених n циклів вектор ітерації I для самого внутрішнього циклу є вектором, що містить ціле число ітерацій для кожного циклу в порядку вкладеності циклів. Іншими словами, номер ітерації багатовимірного вкладеного циклу визначається відповідно до форми $I = \{i_1,i_2 ....,i_n)$, де i_k, $1\le k\le n$, являє собою номер ітерації циклу для рівня вкладеності k.

Визначення. Ітераційний простір це множина всіх цілочисельних векторів $I = (I_1, I_2, ..., I_n)$, що задовольняють нерівності:

$$L_i \le x_i \le \mathrm{U}_i, i = 1..n, \qquad (2.1)$$

Нерівність (2.1) визначає межі циклу, що обмежують ітераційний простір опуклим багатогранником.

Визначення. Нехай два входження u та v мають d спільних циклів. Розглянемо множину усіх ітерацій $I_k=(I_{k1}, I_{k2}, \ldots, I_{kn1})$ та $J_k=(J_{k1}, J_{k2}, \ldots, J_{kn2})$, де $1\le k\le m$, $n_1\ge d$, $n_2\ge d$, таких, що $v[J_k]$ залежить від $u[I_k]$. Кожен вектор $(J_{k1} – I_{k1}, J_{k2} – I_{k2}, \ldots, J_{kd} – I_{kd})$ (при фіксованому k) будемо називати *вектором відстані залежності v від u*, а множину різних векторів $D=(J_{k1} – I_{k1}, J_{k2} – I_{k2}, \ldots, J_{kd} – I_{kd})$ будемо називати *множиною векторів відстані залежності v від u*.

Визначити тип існуючої залежності за даними і можливість розпаралелювання циклу по виду вектору відстаней складно. Тому вводиться поняття вектору напрямків для циклу. Компоненти вектору напрямків d є символами, які визначаються наступним чином:

$$\begin{cases} d_i = "=", D_i = 0; \\ d_i = "<", D_i < 0; \\ d_i = ">", D_i > 0; \end{cases} \quad (2.2)$$

Нехай дві операції S і S* виконуються в тілі циклу и мають залежність. Напрямок «>» означає, що операція S звертається до загальної ділянки пам'яті M на деяку кількість ітерацій пізніше, ніж операція S*. Напрямок «<» означає, що операція S звертається до загальної ділянки пам'яті M на деяку кількість ітерацій раніше, ніж операція S*. Напрямок «=» означає, що операції S і S* звертаються до загальної ділянки пам'яті M на одній і тій же ітерації циклу.

Відстань залежності грає важливу роль при аналізі циклу на паралельність. Її значення дозволяє визначати тип залежності, що виникає, за даними та можливість розбиття ітераційного простору на зони відповідальності для паралельного виконання.

Ітераційний простір це спосіб представлення обчислювальних циклів у вигляді n-мірного опуклого багатогранника, вузли якого являють собою кожну обчислювальну ітерацію циклу. Приклад коду, його ітераційного простору у матричній формі та його графічне представлення зображено на Рисунку 2.1.

Для математичного опису обчислювальних циклів використовується кілька підходів. Одним з найбільш опрацьованих і наочних методів опису обчислювальних циклів є поліедральна модель або модель багатогранників. Дана модель дозволяє представити довільний обчислювальний цикл як деяку математичну абстракцію. Далі модель може бути модифікована будь-яким із способів, який не змінює результатів безпосередньо вихідних даних і потім модель може бути перетворена назад до початкового коду. Такий підхід до оптимізації дозволяє гнучко вибирати методи оптимізації. У той же час підхід до оптимізації за схемою S2S перетворення містить в собі також ту перевагу,

що розробник концентрується на вузькій прикладній задачі перетворення обчислювальних циклів, а не розробляє надзвичайно складний компілятор для перетворення в бінарний код. Після етапу перетворення коду в оптимізований вихідний код компілятор виконає всі додаткові оптимізації на своєму рівні.

```
for(i1 = 1; i1 < N; i1++)
  for(i2 = 1; i2 < i1; i2++)
    for(i3 = 1; i3 < N; i3++)
    {operations}
```

Ітераційний простір

$$D = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 0 \end{bmatrix} \begin{pmatrix} i1 \\ i2 \\ i3 \\ N \\ 1 \end{pmatrix} \geq 0 \begin{vmatrix} i1 \geq 1 \\ i1 \leq N \\ i2 \geq 1 \\ i2 \leq i1 \\ i3 \geq 1 \\ i3 \leq N \end{vmatrix}$$

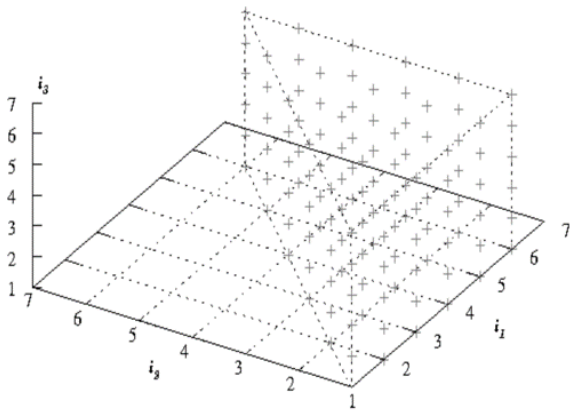

Рисунок 2.1 – Приклад ітераційного простору для потрійного обчислювального циклу

Для вирішення проблеми оптимізації циклів використовуються рішення на основі зазначеної поліедральної моделі (polyhedral model) [52]. Поліедральна модель є математичною основою для оптимізації вкладених циклів оптимізації програм. Поліедральна модель обробляє кожну ітерацію циклу всередині вкладених циклів як точки решітки всередині математичних об'єктів, названих багатогранники, виконує афінне перетворення або, в деяких випадках, неафінне перетворення, а потім перетворює змінені багатогранники в еквівалентні, але оптимізовані (в залежності від цільової задачі оптимізації) вкладені цикли. Цей оптимізаційний метод полягає у розділенні ітераційного простору початкового обчислювального циклу, що може складатись з деяких змінних, на блоки меншого розміру. Розбиття масиву на менші блоки та зберігання їх у кеші призводить до частого використання кешу, зменшення кеш-промахів та зменшення вимог на розмір кешу. Таким чином покращується локальність даних.

Поліедральна модель забезпечує абстракцію для подання вкладених обчислювальних циклів і їх залежностей з використанням точок в багатогранниках [49], [50], [51]. Послідовне виконання і перепризначення може підвищити продуктивність за рахунок розпаралелювання, а також підвищення локальності даних [54]. З огляду на останні досягнення, поліедральна модель досягла рівня зрілості в різних аспектах – зокрема, в якості потужного проміжного представлення для виконання перетворень і генерації коду після перетворень [90]. Автоматична оптимізація до недавнього часу була ключовою відсутньою ланкою. Зовсім нещодавно з'явилися автоматичні засоби оптимізації, що використовують поліедральну модель.

Далі модель може бути модифікована будь-яким із способів, який не змінює результатів безпосередньо вихідних даних і потім модель може бути перетворена назад до початкового коду. Після етапу перетворення коду в оптимізований вихідний код компілятор виконає всі додаткові оптимізації на своєму рівні оптимізацій [10-16].

Поліедральна модель дозволяє застосовувати різні методи оптимізації обчислювальних циклів. У загальному випадку, схема роботи з вихідним кодом має вигляд, як показано на Рисунку 2.2:

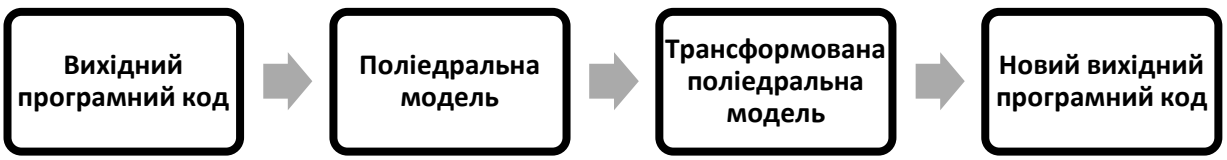

Рисунок 2.2 – Схема використання поліедральної моделі

Поліедральна модель дозволяє працювати з обчислювальними циклами довільної вкладеності. Наприклад, такий подвійний обчислювальний цикл:

```
for(i = 0; i < n; i ++)
    for(j = 0; j < i + 2; j++)
        A[i][j] = A[i – 1][j] + A[i][j – 1];
```

Можна представити в графічному вигляді поліедральної моделі, як показано на Рисунку 2.3.

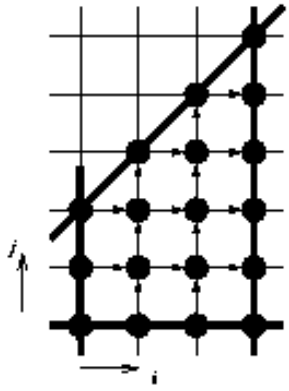

Рисунок 2.3 – Приклад графічного представлення поліедральної моделі

Після перетворення даної моделі модифікована поліедральна модель має вигляд, як показано на Рисунку 2.4:

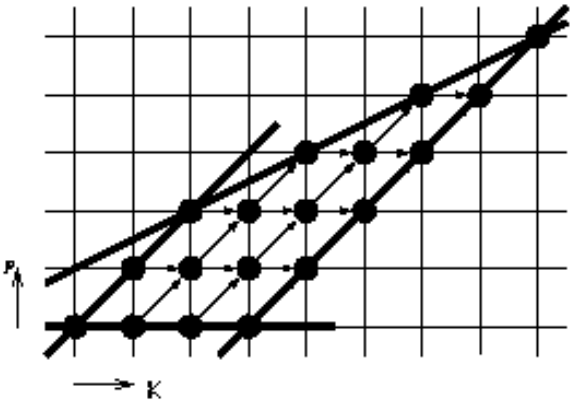

Рисунок 2.4 – Приклад графічного представлення модифікованої поліедральної моделі

Вихідний код фрагмента програми, згенерований на основі моделі на Рисунку 2.4 матиме вигляд:

```
for(k = 0; k < 2*n + 2; k ++)
    for(p = max(0, k – n); p < min(k, k/2 + 1); p++)
        A[k – p][p] = A[k – p – 1][p] + A[k – p][p – 1];
```

2.2 Оптимізація S2S перетворень

Як було зазначено, одним зі способів реалізації оптимізацій ПЗ є S2S перетворення. З точки зору розробки та запуску КП додається додатковий крок – S2S перетворення, що передує компіляції програми. Загальна схема використання такого методу наводиться на Рисунку 2.5:

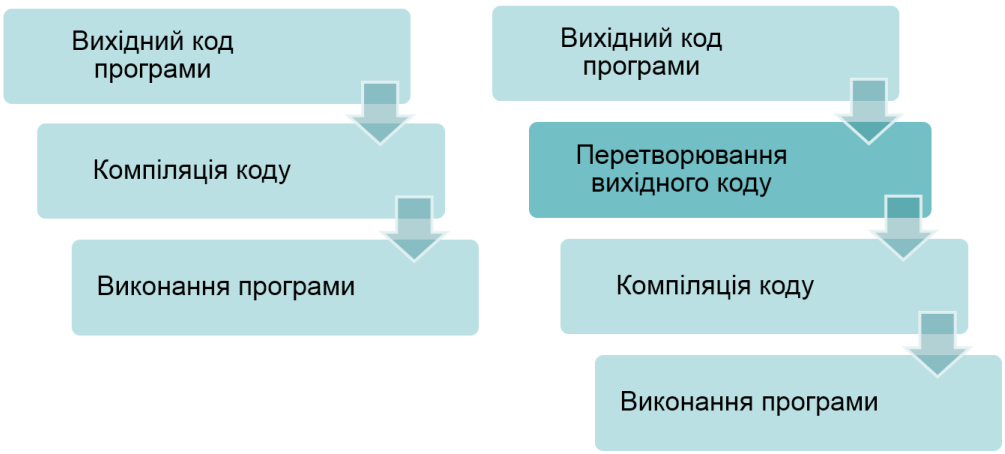

Рисунок 2.5 – Порівняння традиційної компіляції програм та компіляції з оптимізацією вихідного коду

Такий підхід розширює можливості по оптимізації та дозволяє використовувати різноманітні методи оптимізації без створення або змінення компіляторів [70]. Перевагою S2S перетворення є його гнучкість та можливість застосовувати різноманітні методи оптимізації, в тому числі ті, що використовують поліедральну модель або модель графів.

Окремо варто відмітити, що оптимізації S2S перетворення є перспективним з точку зору апаратної незалежності. Окремо варто відмітити можливість отримання синергетичного ефекту від одночасного використання оптимізаційних методів S2S перетворення та оптимізаційних методів компілятора, що призведе до покращення цільової функції відносно використання тільки оптимізаційних методів компілятора. Дійсно, оптимізації на рівні коду не впливають на оптимізації на рівні команд процесора, тому сумісне використання методів оптимізацій S2S перетворення та оптимізацій компілятора може взаємно доповнювати один одне.

Варто виділити переваги та недоліки оптимізаційних методів, що базуються на S2S перетворюванні:

Переваги:

- апаратна незалежність та незалежність від компілятора;
- доповнюють оптимізаційні методи компілятора;
- розробка одного конкретного методу потребує значно менше зусиль ніж розробка компілятора з нуля;
- наявність наукових досліджень та напрацювань у відкритому доступі.

Недоліки:

- реалізація оптимізаційного методу в існуючому компіляторі швидша за створення реалізації того ж самого методу шляхом S2S перетворення.

Загалом, всі оптимізаційні методи можна поділити на дві великі групи: детерміновані та ітеративні. Детерміновані методи на основі аналізу коду згідно закладених правил одразу створюють оптимізований код, який використовується одразу. Ітеративні методи, що не так розповсюджені, спочатку створюють первісний оптимізований код, а згодом його покращують для досягнення кращих результатів оптимізації.

Загалом, ітеративні підходи мають дуже широке застосування при вирішенні задач різного класу, наприклад, у [125] пропонується метод розв’язку систем нелінійних алгебраїчних рівнянь, що використовує ітеративні кроки пошуку.

Приклад послідовного ітеративного підбору кращих параметрів оптимізації для підвищення швидкодії наведено на Рисунку 2.6.

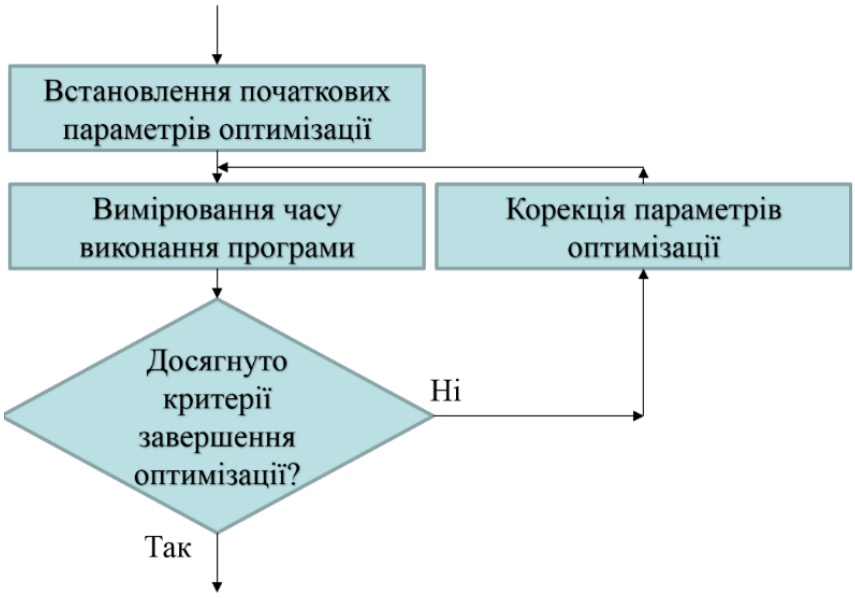

Рисунок 2.6 – Загальна схема пошуку найкращих параметрів оптимізаційного методу для прискорення швидкодії виконання програм

Використання принципу ітеративності при оптимізації ПЗ дозволяє знаходити більш ефективні методи оптимізації ПЗ та параметри методів оптимізації ПЗ для кожного конкретного коду. Це досягається шляхом оцінки ефективності використаних методів оптимізації та їх параметрів на кожній ітерації та оцінюванням отриманих результатів.

2.3 Метод розбиття обчислювальних циклів на блоки

Розглядаючи різноманіття підходів модифікації обчислювальних циклів, слід відзначити такий метод як розбиття циклу на блоки (tiling method). Це один з ефективних методів трансформації обчислювальних циклів. Даний метод вводить додаткові вкладені цикли, які розбивають весь простір поліедральної моделі на невеликі блоки. Обчислення відбувається спочатку по кожному блоку. Розмір блоку вибирається в кожному конкретному випадку індивідуально. Такий підхід покращує локальність даних, що виражається в більш ефективному використанні кеша і внутрішніх регістрів процесора, і це може призводити до зниження навантаження на шину пам'яті, прискоренню роботи обчислень, зниження енергоспоживання. Приклад розбиття на блоки представлений на Рисунку 2.7.

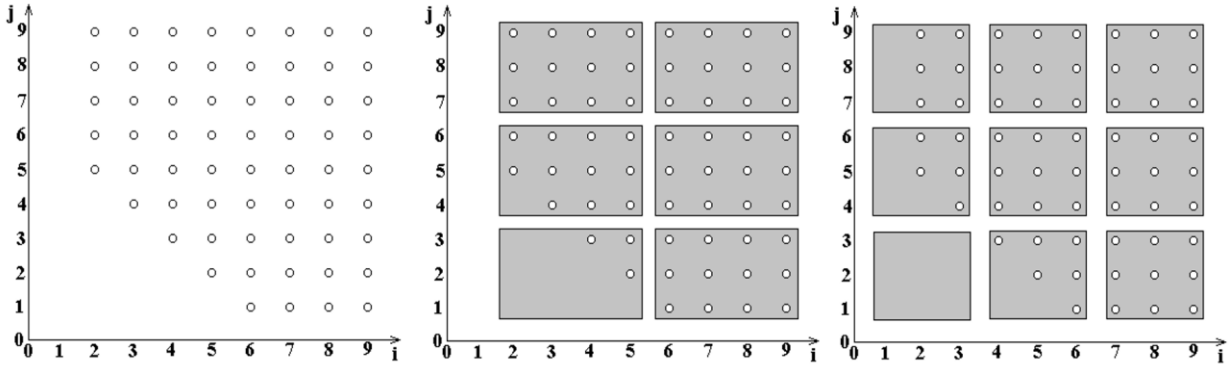

Рисунок 2.7 – Приклад використання поліедральної моделі з методом розбиття на блоки

Всі операції виконуються на цілочисельному базисі, що визначає та обмежує індекси змінних циклу. Вони визначають розмірності та розміри ітераційного простору.

Як було показано в [149], [150] та [151] розбиття на блоки та розпаралелювання часто призводять до покращення в термінах швидкодії та енергоефективності. Проте вказані дослідження не розкривають, як саме впливає розмір блоку розбиття на вказані характеристики і якщо впливає, то в якій мірі та який характер цієї залежності.

Інший метод оптимізації – паралелізація, що дозволяє використовувати для обчислень всі ядра замість одного.

З метою прискорення швидкодії та енергоефективності обчислень пропонується використання МРОЦНБ та метод розпаралелювання. Суть МРОЦНБ в розбитті великого ітераційного простору на один або декілька блоків розбиття меншого розміру.

Наприклад розглянемо такий подвійний цикл:

```
for (i = 1; i <= 10; i++)
  for (j = 1; j <= 10; j++)
  {
    …
  }
```

Графічна інтерпретація ітераційного простору для такого подвійного циклу зображена в лівій частині Рисунку 2.8. Стрілками відображено перехід між точками ітераційного простору. У випадку використання МРОЦНБ, як вже було зазначено, ітераційний простір розбивається на ітераційні простори меншого розміру, так звані блоки, і обхід ітераційного простору буде виконуватись в середині блоків розбиття [40], [43], [45], як зображено на правій частині Рисунку 2.8.

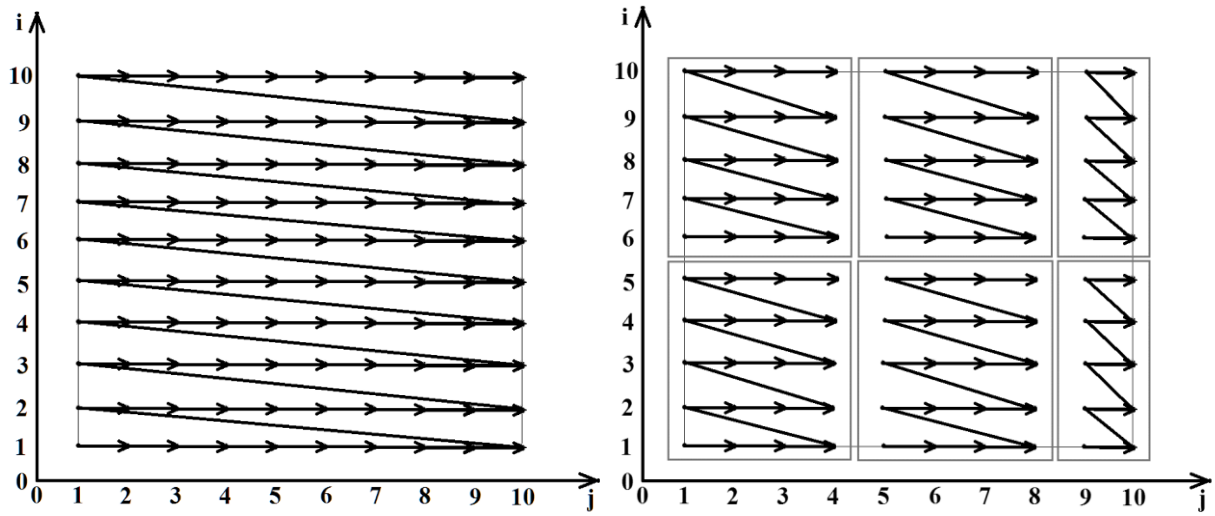

Рисунок 2.8 – Приклад розбиття ітераційного простору на блоки розміром 5x4

МРОЦНБ покращує локальність даних, а також надає можливість виконання нових блоків на різних ядрах мікропроцесора, що сприятиме ефективності розпаралелювання КП. Однак ефективне розпаралелювання можливе тільки при відсутності залежностей між операціями різних блоків.

Метод розбиття обчислювальних циклів на блоки останнім часом отримав достатній розвиток в дослідницькій сфері. Оскільки для цього методу достатньо S2S перетворення, цей метод може бути використаний без розробки компілятора. Методи трансформації ітераційного простору застосовується такими програмними пакетами PIPS [4], Polaris [17], OPC [122], [152], [153], [154], [155], Petit [63], Omega Calculator [64], [84], Suif [94].

Для задачі верифікації методу розбиття на блоки використано пакет програм Pluto [18-21]. Цей пакет включає низку програмних модулів, що

почергово спочатку створюють поліедральну модель на основі вихідного коду мовами C або C++, потім власне сам пакет Pluto модифікує цю поліедральну модель згідно заданих методів розбиття на блоки та паралелізму, далі інші програмні модулі [88], [89] спрощують отриману модель не змінюючи лексикографічної послідовності, і на останньому етапі програмні модулі створюють вихідний код мовами програмування C або C++ по отриманій моделі.

Таким чином, програмний пакет Pluto є прикладом використання підходу оптимізації S2S перетворень. Кроки послідовності обробки програмного пакету Pluto зображено на Рисунку 2.9

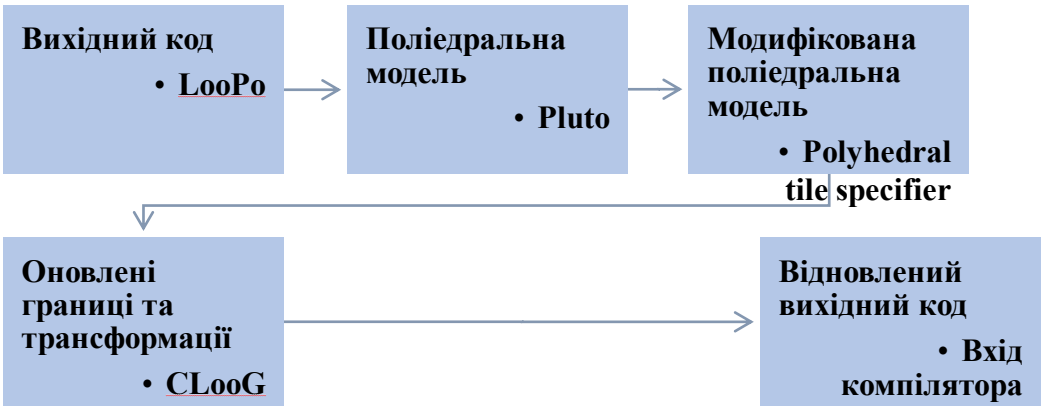

Рисунок 2.9 – Послідовність обробки вихідного коду у пакеті Pluto

Переваги використання пакету програм Pluto:

- Програма використовує декілька модифікацій методу розбиття на блоки;
- Програма може застосовувати методи розбиття на блоки як поодинці, так і спільно з розпаралелюванням з застосуванням бібліотеки OpenMP [75];
- Наявність документації по встановленню з переліком необхідних сторонніх пакетів.

Розглядаючи сучасні дослідження щодо розпаралелювання КП варто відзначити, що вони направлені безпосередньо на підвищення паралелізму обчислень, що призводить до мінімального простою наявних ядер. Такі

дослідження висвітлено в роботах [10], [26], [29], [30], [53], [100], [134]. Зважаючи на вагомі результати досліджень в розв'язанні задачі розпаралелювання КП, варто також відзначити, що не завжди приділяється увага дослідженню поєднання розпаралелювання КП з одночасною оптимізацією програм. Деякі з методів оптимізації КП, наприклад метод розбиття на блоки, можуть бути ефективно поєднані з розпаралелюванням КП, та, теоретично, підсилювати ефективність розпаралелювання, оскільки на рівні обчислювальних циклів створюються блоки, що можуть бути оброблені різними ядрами мікропроцесору.

Окремо варто відзначити, що для деяких методів оптимізації КП, наприклад для зазначеного метода розбиття на блоки, існують власні параметри, що впливають на ефективність самого оптимізаційного метода явно, так можуть і неявно впливати на ефективність методів розпаралелювання. Таким чином, підбір параметрів оптимізації також може впливати на ефективність розпаралелювання та оптимізації ПЗ загалом.

Зважаючи на вказані факти, пропонується ітераційний процес розпаралелювання операторів циклів мікропроцесорних програм. Ітераційна сутність процесу виявляється у тому, що використовується розпаралелювання одночасно з оптимізаційними методами декілька разів і на кожній ітерації використовуються різні параметри оптимізаційного методу. При цьому для кожної ітерації виконується оцінка цільової функції оптимізації. Перевагою такого ітеративного процесу є те, що через зміну параметрів оптимізаційного методу поступово виконується пошук кращого результату оптимізації та розпаралелювання.

Використовуючи схему S2S перетворення, що зображено на Рисунку 2.9 як базову, розширимо її з метою опису запропонованого ітераційного процесу. Отриману схему зображено на Рисунку 2.10. Вказаний процес може бути застосований на заздалегідь задану кількість ітерацій або мати деякий критерій завершення. В розділі 3 даної дисертації буде розроблено конкретний

метод пошуку кращих розмірів блоків розбиття, що базується на даному ітеративному процесі.

Рисунок 2.10 – Схема ітераційного процесу розпаралелювання операторів циклів мікропроцесорних програм

2.4 Визначення ефективності методів розбиття на блоки та розпаралелювання на тестових програмах

Для оцінки впливу методу розбиття на блоки на час виконання програм та на енергоспоживання було виконано чотири серії експериментів. Перші дві серії експериментів було виконано на компактному одноплатному комп’ютері Raspberry Pi 3 (чотириядерний центральний процесор ARM Cortex-A53, 1.2 ГГц, кеш L1 32KB, L2 512KB, пам'ять 1ГБ LPDDR2 900 МГц). Третя та четверта серії експериментів було виконано на сучасному ПК з процесором Intel (чотириядерний центральний процесор Intel® Core™ i5-4670K, 3.4 ГГц, кеш L1 4*64KB, L2 4*256KB, L3 6МБ, пам'ять 16ГБ DDR3 1333 МГц).

В якості тестових програм використано набір тестових програм з пакету Polybench 4.1 [80], [81]. Цей пакет містить близько 30 невеликих тестових програм написаних мовою програмування С, що реалізують різноманітні

класи алгоритмів – операції з векторами, матрицями, лінійної алгебри, обробки зображень тощо. Даний пакет тестових програм використовується науковцями та розробниками, як релевантний приклад різноманітних тестових програм з відомими алгоритмами та відомою обчислювальною складністю. Опис тестових програм, їх обчислювальна складність та розміри необхідної пам'яті наведено на Рисунку 2.11.

Benchmark	Description
2mm	2 Matrix Multiplications (D=A.B; E=C.D)
3mm	3 Matrix Multiplications (E=A.B; F=C.D; G=E.F)
adi	Alternating Direction Implicit solver
atax	Matrix Transpose and Vector Multiplication
bicg	BiCG Sub Kernel of BiCGStab Linear Solver
cholesky	Cholesky Decomposition
correlation	Correlation Computation
covariance	Covariance Computation
doitgen	Multiresolution analysis kernel (MADNESS)
durbin	Toeplitz system solver
dynprog	Dynamic programming (2D)
fdtd-2d	2-D Finite Different Time Domain Kernel
fdtd-apml	FDTD using Anisotropic Perfectly Matched Layer
gauss-filter	Gaussian Filter
gemm	Matrix-multiply C=alpha.A.B+beta.C

Benchmark	Description
gemver	Vector Multiplication and Matrix Addition
gesummv	Scalar, Vector and Matrix Multiplication
gramschmidt	Gram-Schmidt decomposition
jacobi-1D	1-D Jacobi stencil computation
jacobi-2D	2-D Jacobi stencil computation
lu	LU decomposition
ludcmp	LU decomposition
mvt	Matrix Vector Product and Transpose
reg-detect	2-D Image processing
seidel	2-D Seidel stencil computation
symm	Symmetric matrix-multiply
syr2k	Symmetric rank-2k operations
syrk	Symmetric rank-k operations
trisolv	Triangular solver
trmm	Triangular matrix-multiply

Програма	Число операцій	Пам'ять	O(опер)	O(память)
correlation	n^3+8n^2+3n	$2n^2+2n$	$O(n^3)$	$O(n^2)$
covariance	n^3+4n^2+2n	$2n^2+n$	$O(n^3)$	$O(n^2)$
2mm	$5n^3+n^2$	$4n^2$	$O(n^3)$	$O(n^2)$
3mm	$6n^3$	$4n^2$	$O(n^3)$	$O(n^2)$
atax	$4n^2$	n^2+3n	$O(n^2)$	$O(n^2)$
bicg	$4n^2$	n^2+4n	$O(n^2)$	$O(n^2)$
doitgen	$2n^4$	n^3+n^2+n	$O(n^4)$	$O(n^3)$
mvt	$4n^2$	n^2+4n	$O(n^2)$	$O(n^2)$
gemm	$3n^3+n^2$	$3n^2$	$O(n^3)$	$O(n^2)$
gemver	$10n^2+n$	n^2+8n	$O(n^2)$	$O(n^2)$
gesummv	$4n^2+3n$	$2n^2+3n$	$O(n^2)$	$O(n^2)$
symm	$\frac{5}{2}n^3+\frac{5}{2}n^2$	$3n^2$	$O(n^3)$	$O(n^2)$
syr2k	$3n^3+\frac{7}{2}n^2+\frac{1}{2}n$	$3n^2$	$O(n^3)$	$O(n^2)$
syrk	$\frac{3}{2}n^3+2n^2+\frac{1}{2}n$	$3n^2$	$O(n^3)$	$O(n^2)$
trmm	n^3	$3n^2$	$O(n^3)$	$O(n^2)$

Програма	Число операцій	Пам'ять	O(опер)	O(память)
cholesky	$\frac{1}{3}n^3+\frac{1}{2}n^2+\frac{1}{6}n$	n^2	$O(n^3)$	$O(n^2)$
durbin	$2n^2+3n-5$	$2n$	$O(n^2)$	$O(n)$
gramschmidt	$2n^3+n^2+n$	$3n^2$	$O(n^3)$	$O(n^2)$
lu	$\frac{4}{3}n^3-\frac{3}{2}n^2+\frac{1}{6}n$	n^2	$O(n^3)$	$O(n^2)$
ludcmp	$\frac{4}{3}n^3+\frac{1}{2}n^2-\frac{5}{6}n$	n^2+3n	$O(n^3)$	$O(n^2)$
trisolv	n^2	n^2+2n	$O(n^2)$	$O(n^2)$
deriche	$32n^2$	$4n^2$	$O(n^2)$	$O(n^2)$
floyd-warshall	$2n^3$	n^2	$O(n^3)$	$O(n^2)$
nussinov	$\frac{1}{6}n^3-\frac{1}{2}n^2+\frac{1}{3}n$	n^2+n	$O(n^3)$	$O(n^2)$
adi	$30n^3$	$4n^2$	$O(n^3)$	$O(n^2)$
fdtd-2d	$11n^3$	$3n^2+n$	$O(n^3)$	$O(n^2)$
heat-3d	$30n^4$	$2n^3$	$O(n^4)$	$O(n^3)$
jacobi-1d	$6n^2$	$2n$	$O(n^2)$	$O(n)$
jacobi-2d	$10n^3$	$2n^2$	$O(n^3)$	$O(n^2)$
seidel-2d	$9n^3$	n^2	$O(n^3)$	$O(n^2)$

Рисунок 2.11 – Опис та обчислювальна складність і необхідна пам'ять тестових програм з пакету Polybench

В першій та третій серії експериментів виконано виміри часу виконання тестових програм до та після застосування МРОЦНБ та розпаралелювання для двох апаратних платформ: для плати Raspberry Pi 3 та ПК на базі Intel® Core™ i5-4670K відповідно. Методика виміру часу виконання тестових програм

складається з програмних вимірів часових міток між початком та закінченням роботи обчислювальних циклів тестових програм. Час виконання тестових програм приймається як різниця між другою та першою часовими мітками. Задля більш точного вимірювання і відкидання флуктуацій часу виконання, вимірювання часу виконання виконувалось п’ять разів для кожної тестової програми, відкиданням мінімально і максимально отриманих часів та усередненням трьох середніх часів виконання програм.

Друга та четверта серії експериментів присвячена вимірюванню енергоспоживання тих самих тестових програм при тих самих методах розбиття на блоки та розпаралелюванні на вказаних апаратних платформах.

Методика вимірювання енергоспоживання відрізняється від методики вимірювання часу виконання. Оскільки час роботи програм часто становить менше секунди, ватметр не в змозі виконати вимірювання за такий короткий час. Тому для вимірювання енергоспоживання використовуються модифіковані тестові програми у яких обчислювальний цикл, енергоспоживання якого вимірюється, додано в середину нескінченного циклу. Такий підхід дозволяє отримати стале енергоспоживання тестової програми, обчислювальний цикл якої може виконуватися навіть за дуже короткий час.

Задля прискорення часу обробки всіх тестових програм створено скрипт, що виконує послідовний запуск всіх тестових програм, обробку отриманих результатів та представлення часу виконання у зручному вигляді.

Результати отримані для апаратної платформи на Raspberry Pi 3 перших двох експериментів, а саме відносного прискорення виконання програм та відносного збільшення енергоспоживання тестових програм представлені в таблицях 2.1 та 2.2.

Результати відносного прискорення виконання програм та відносного збільшення енергоспоживання тестових програм, отримані в експериментах 3 та 4 для ПК на базі Intel® Core™ i5-4670K, представлені в таблицях 2.3 та 2.4.

Таблиця 2.1 – Відносне прискорення часу виконання тестових програм для платі Raspberry Pi 3 при використанні різних методів розбиття на блоки та розпаралелюванні.

Тестова програма	**Опції оптимізації програми Pluto**							
	Tile	**Tile Parallel**	**Tile L2Tile Parallel**	**Innerpar**	**Innerpar Tile Parallel**	**Tile Multipar Parallel**	**Diamond-tile**	**Diamond-tile Parallel**
correlation	1,71	3,44	1,80	1,96	3,57	2,30	1,95	3,62
covariance	1,81	3,71	1,72	1,83	3,74	2,26	1,75	3,78
gemm	1,09	2,15	2,14	1,03	2,15	1,69	1,02	2,22
gemver	3,56	7,14	7,78	3,75	7,40	4,67	2,76	7,45
gesummv	1,05	2,45	2,16	1,20	2,47	1,34	1,02	2,75
symm	1,02	0,99	1,07	1,03	1,07	1,01	1,04	0,98
syr2k	2,05	5,53	1,58	1,86	5,03	2,84	1,95	5,62
syrk	1,26	2,36	0,85	1,26	2,33	1,19	1,17	2,29
trmm	1,28	4,49	1,25	1,43	4,52	2,30	1,28	5,41
2mm	1,11	2,44	0,89	0,90	2,44	1,65	1,09	2,48
3mm	1,16	2,39	0,85	1,00	2,40	1,67	1,16	2,42
atax	0,64	1,41	1,46	0,97	1,42	1,04	0,64	1,46
bicg	0,66	1,47	1,46	1,02	1,48	1,01	0,66	1,51
doitgen	1,15	1,72	0,74	1,28	1,72	1,31	1,08	1,74
mvt	1,33	4,13	4,14	0,84	4,13	2,21	1,29	4,21
cholesky	0,88	1,95	0,77	0,90	2,25	1,26	0,95	2,08
durbin	0,98	0,98	1,01	1,07	0,98	1,00	1,01	0,99
gramschmidt	1,28	2,69	1,05	1,00	2,66	1,74	1,25	2,69
lu	1,76	4,49	1,82	1,84	4,62	2,78	1,65	3,95
ludcmp	1,11	1,10	1,05	1,04	1,16	1,03	1,04	1,05
trisolv	0,86	1,18	1,83	1,01	2,32	0,88	0,86	1,21
deriche	0,98	1,05	0,96	1,01	1,01	0,95	0,98	1,02
floyd-warshall	0,62	0,89	0,39	1,01	1,02	0,59	0,63	0,90
nussinov	1,03	3,13	1,12	1,00	2,51	1,85	1,01	2,96
fdtd-2d	1,16	2,24	0,98	1,11	3,28	1,77	1,00	1,00
heat-3d	0,96	0,95	1,00	1,07	1,85	1,09	1,00	1,00
jacobi-2d	1,04	1,73	1,00	0,99	2,60	1,56	1,04	1,00
seidel-2d	0,99	2,05	1,00	0,99	2,72	1,52	1,02	2,06

Таблиця 2.2 – Відносне збільшення енергоспоживання тестових програм для платі Raspberry Pi 3 при використанні різних методів розбиття на блоки та розпаралелюванні.

Тестова програма	**Опції оптимізації програми Pluto**							
	Tile	**Tile Parallel**	**Tile L2Tile Parallel**	**Innerpar**	**Innerpar Tile Parallel**	**Tile Multipar Parallel**	**Diamond-tile**	**Diamond-tile Parallel**
correlation	0,96	1,16	1,13	0,98	1,13	1,01	0,95	1,14
covariance	0,96	1,13	1,14	0,98	1,15	1,01	0,97	1,13
gemm	0,95	1,38	1,36	1,00	1,37	1,11	0,97	1,36
gemver	1,04	1,57	1,59	1,04	1,58	1,25	1,03	1,57
gesummv	1,02	1,56	1,59	1,03	1,59	1,20	1,02	1,58
symm	1,01	0,99	1,01	1,01	1,02	1,01	1,00	1,02
syr2k	0,97	1,43	1,45	1,02	1,43	1,13	1,00	1,47
syrk	0,99	1,25	1,28	1,00	1,27	1,05	1,00	1,29
trmm	0,98	1,38	1,38	1,00	1,38	1,12	0,98	1,38
2mm	0,99	1,29	1,28	1,02	1,31	1,15	0,99	1,31
3mm	0,98	1,31	1,29	1,02	1,28	1,14	0,99	1,29
atax	1,03	1,50	1,48	1,04	1,49	1,22	1,04	1,49
bicg	1,04	1,50	1,50	1,02	1,50	1,23	1,05	1,50
doitgen	1,00	1,46	1,46	0,98	1,46	1,15	1,01	1,46
mvt	1,01	1,50	1,49	1,00	1,50	1,18	1,02	1,50
cholesky	1,01	1,22	1,21	0,95	1,18	0,98	0,91	1,22
durbin	0,99	1,01	0,99	1,00	1,01	1,01	1,00	1,01
gramschmidt	0,97	1,41	1,42	1,00	1,43	1,13	1,24	1,42
lu	0,90	1,18	1,16	0,94	1,15	1,01	1,01	1,18
ludcmp	0,99	1,01	1,01	1,00	1,01	1,01	1,00	1,01
trisolv	0,99	1,45	1,46	1,01	1,52	1,22	0,99	1,46
deriche	1,00	1,00	1,01	1,00	1,02	1,02	1,00	1,01
floyd-warshall	0,94	1,31	1,32	1,01	1,30	1,06	0,94	1,32
nussinov	1,00	1,41	1,42	1,00	1,34	1,16	1,01	1,42
fdtd-2d	0,90	1,12	1,05	0,97	1,17	1,01	1,00	1,00
heat-3d	0,97	0,99	0,99	1,00	1,14	1,01	1,00	1,00
jacobi-2d	0,91	1,07	1,07	0,98	1,16	1,03	1,00	1,00
seidel-2d	0,96	1,07	1,08	1,00	1,14	1,05	0,96	1,14

Таблиця 2.3 – Відносне прискорення часу виконання тестових програм для ПК на базі Intel® Core™ i5-4670K при використанні різних методів розбиття на блоки та розпаралелюванні.

Тестова програма	**Опції оптимізації програми Pluto**								
	Tile	**Tile Parallel**	**Tile L2Tile Parallel**	**Innerpar**	**Innerpar parallel**	**Innerpar Tile Parallel**	**Tile Multipar Parallel**	**Diamond-tile**	**Diamond-tile Parallel**
correlation	4,91	11,18	3,93	4,63	7,78	11,43	6,32	5,01	10,85
covariance	4,99	10,86	3,85	4,70	9,63	10,87	6,47	4,92	10,75
gemm	2,31	9,70	11,62	1,44	3,88	9,51	4,97	2,29	9,39
gemver	5,17	16,10	16,08	14,39	22,07	15,90	9,12	5,03	14,37
gesummv	0,89	2,83	2,40	1,68	4,63	2,98	1,64	0,89	2,78
symm	1,05	1,03	1,05	1,02	1,02	1,05	1,05	1,05	1,05
syr2k	1,41	5,26	1,44	1,42	5,63	5,22	2,64	1,32	5,20
syrk	0,99	2,85	1,08	0,99	2,24	2,14	1,43	1,00	2,85
trmm	3,43	17,52	3,62	5,23	1,22	17,38	9,06	3,46	16,43
2mm	1,94	5,61	1,86	1,61	6,14	5,56	3,75	1,95	5,54
3mm	1,63	4,71	1,57	1,31	5,03	4,75	3,14	1,63	4,73
atax	0,40	1,26	1,27	0,99	1,45	1,24	0,69	0,39	1,25
bicg	0,82	2,64	2,52	2,12	2,85	2,56	1,46	0,77	2,00
doitgen	5,02	4,00	3,43	6,00	0,25	4,03	5,22	5,03	3,90
mvt	3,78	11,32	12,84	1,16	3,68	11,48	7,16	3,58	9,99
cholesky	1,26	2,98	1,16	1,00	1,80	3,09	1,62	1,27	3,02
durbin	1,00	1,00	0,99	0,99	1,00	1,00	1,00	1,00	1,00
gramschmidt	1,77	3,13	1,40	1,04	2,85	3,13	2,58	1,77	3,06
lu	1,49	3,80	1,74	1,56	1,40	4,45	2,53	1,50	3,73
ludcmp	1,00	1,00	1,00	0,99	1,00	1,00	0,98	0,96	0,99
trisolv	0,71	1,79	1,57	1,22	1,54	1,95	1,02	0,74	1,31
deriche	1,28	1,98	1,58	1,00	1,91	2,22	1,25	1,19	1,40
floyd-warshall	0,83	1,62	1,02	0,99	0,42	1,63	1,12	0,83	1,61
nussinov	1,05	3,07	1,20	1,01	2,21	2,62	1,93	1,06	3,09
fdtd-2d	0,91	1,52	0,39	0,94	2,87	2,27	1,33	1,00	1,00
heat-3d	2,07	2,11	1,00	2,55	7,98	3,28	2,22	1,00	1,00
jacobi-2d	0,97	1,78	1,00	0,96	3,59	2,19	1,43	1,00	1,00
seidel-2d	1,19	2,80	1,00	1,01	3,89	3,75	1,97	1,00	1,00

Таблиця 2.4 – Відносне збільшення енергоспоживання тестових програм для ПК на базі Intel® Core™ i5-4670K при використанні різних методів розбиття на блоки та розпаралелюванні.

Тестова програма	**Опції оптимізації програми Pluto**								
	Tile	**Tile Parallel**	**Tile L2Tile Parallel**	**Innerpar**	**Innerpar parallel**	**Innerpar Tile Parallel**	**Tile Multipar Parallel**	**Diamond-tile**	**Diamond-tile Parallel**
correlation	1,01	1,39	1,28	1,04	1,38	1,38	1,13	1,02	1,39
covariance	1,04	1,44	1,30	1,06	1,50	1,48	1,16	1,04	1,44
gemm	0,98	1,54	1,63	1,01	1,52	1,55	1,16	0,98	1,55
gemver	1,01	1,30	1,31	1,11	1,45	1,31	1,10	1,02	1,30
gesummv	1,02	1,31	1,29	1,06	1,41	1,31	1,12	1,01	1,30
symm	1,01	1,01	1,01	0,99	1,01	1,00	1,00	1,01	1,00
syr2k	1,00	1,43	1,25	1,01	1,49	1,43	1,13	1,00	1,43
syrk	0,99	1,34	1,23	0,97	1,30	1,34	1,09	0,98	1,34
trmm	1,01	1,58	1,28	1,05	1,39	1,57	1,21	1,01	1,58
2mm	1,02	1,43	1,28	1,02	1,50	1,42	1,16	1,01	1,43
3mm	1,03	1,44	1,28	1,02	1,47	1,43	1,18	1,03	1,44
atax	0,95	1,21	1,23	1,02	1,30	1,21	1,04	0,94	1,21
bicg	1,00	1,27	1,30	1,07	1,38	1,27	1,10	0,99	1,28
doitgen	1,06	1,41	1,29	1,07	1,23	1,40	1,19	1,07	1,41
mvt	1,03	1,33	1,35	1,00	1,26	1,33	1,14	1,03	1,34
cholesky	1,02	1,36	1,27	1,01	1,29	1,36	1,12	1,01	1,35
durbin	1,00	1,00	1,02	1,00	1,00	1,00	1,00	1,00	1,00
gramschmidt	0,99	1,30	1,25	0,99	1,34	1,30	1,10	0,99	1,30
lu	1,02	1,39	1,30	1,03	1,35	1,43	1,15	1,02	1,39
ludcmp	0,98	1,00	1,00	1,00	1,00	1,00	0,99	0,99	1,00
trisolv	0,96	1,25	1,23	1,00	1,27	1,27	1,06	0,96	1,25
deriche	1,00	1,26	1,27	1,00	1,27	1,28	1,06	1,00	1,25
floyd-warshall	1,02	1,34	1,30	1,00	1,27	1,36	1,13	1,02	1,35
nussinov	1,01	1,33	1,27	1,01	1,30	1,33	1,12	1,01	1,34
fdtd-2d	0,97	1,29	1,17	0,97	1,47	1,33	1,10	1,00	1,00
heat-3d	1,03	1,28	1,00	1,03	1,48	1,33	1,13	1,00	1,00
jacobi-2d	1,00	1,32	1,00	0,99	1,41	1,36	1,13	1,00	1,00
seidel-2d	1,00	1,26	1,00	1,01	1,22	1,24	1,09	1,00	1,00

2.5 Аналіз результатів застосування оптимізаційних методів

Аналізуючи дані пришвидшення виконання тестових програм, можна виділити той факт, що оптимізація різних тестових програм методами розбиття на блоки призводить до різних результатів і демонструє різну ефективність.

В залежності від ефективності оптимізації можна виділити такі типи тестових програм:

- всі методи оптимізації завжди пришвидшують виконання програм;
- в деяких випадках оптимізація пришвидшує виконання, в інших сповільнює;
- застосування усіх використаних методів оптимізації не призводить до суттєвого прискорення або сповільнення часу виконання тестових програм.

Узагальнена статистика розподілу зміни часу виконання та енергоефективності обчислень ТП при використанні різноманітних методів розбиття на блоки та розпаралелюванні для плати Raspberry Pi 3 представлена на Рисунках 2.12 та 2.13.

Узагальнена статистика розподілу зміни часу виконання та енергоефективності обчислень ТП при використанні різноманітних методів розбиття на блоки та розпаралелюванні для ПК на базі Intel® Core™ i5-4670K представлена на Рисунках 2.14 та 2.15.

Отримані дані свідчать, що для плати Raspberry Pi 3 статистично найбільш часто спостерігається значне прискорення (>50%) – у 45% випадків та незначні зміни швидкодії (у межах ±5%) – у 23% випадків. Для енергоспоживання було зафіксовано незначні зміни у 47% та значне підвищення у 27% випадків.

Для ПК на базі Intel® Core™ i5-4670K також статистично найбільш часто спостерігається значне прискорення (>50%) – у 62% випадків та незначні зміни швидкодії (у межах ±5%) – у 16% випадків. Для енергоспоживання було зафіксовано значне підвищення у 45% та незначні зміни у 37% випадків.

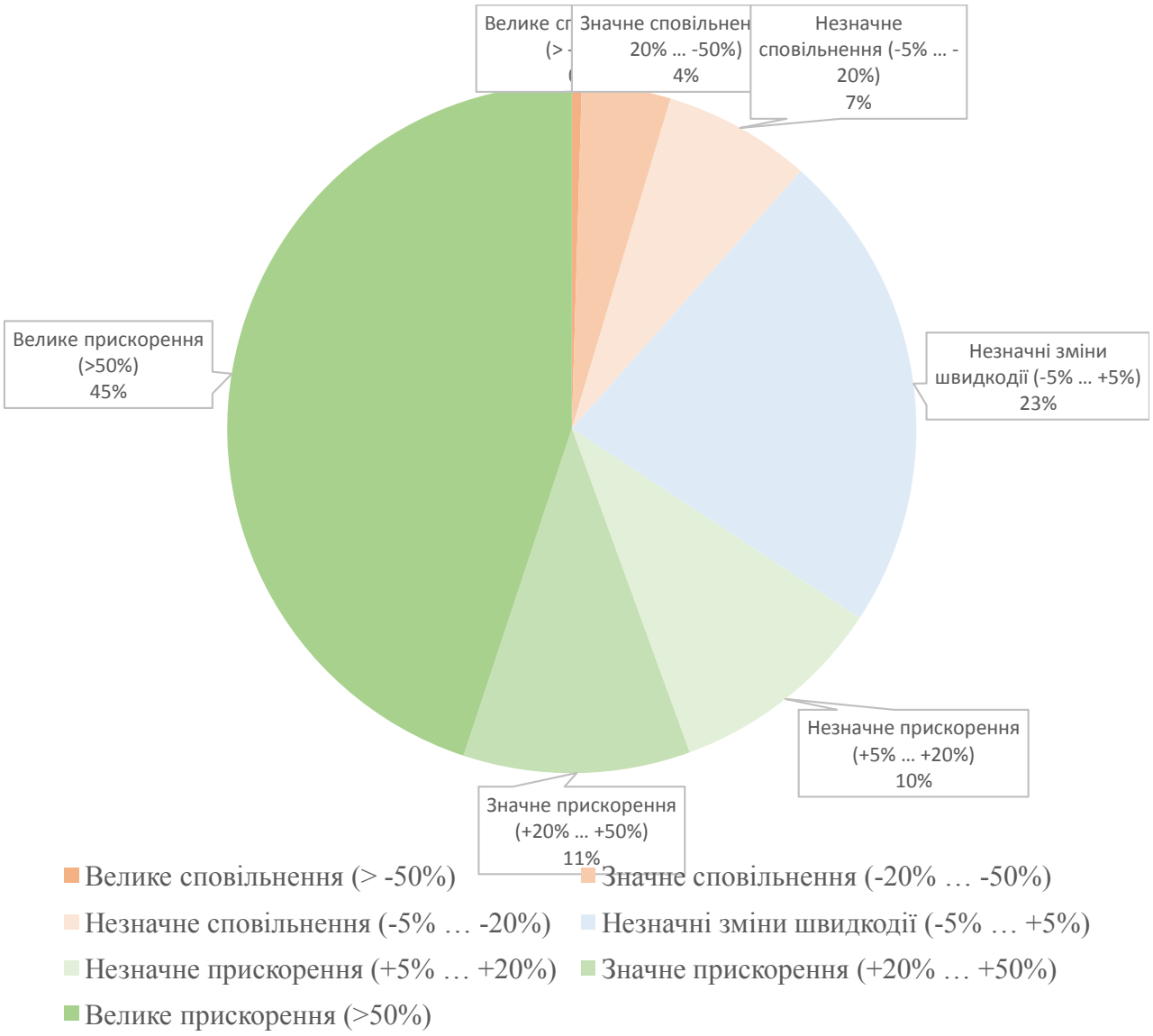

Рисунок 2.12 – Статистика розподілу зміни часу виконання ТП при застосуванні методів розбиття на блоки та розпаралелюванні для плати Raspberry Pi 3

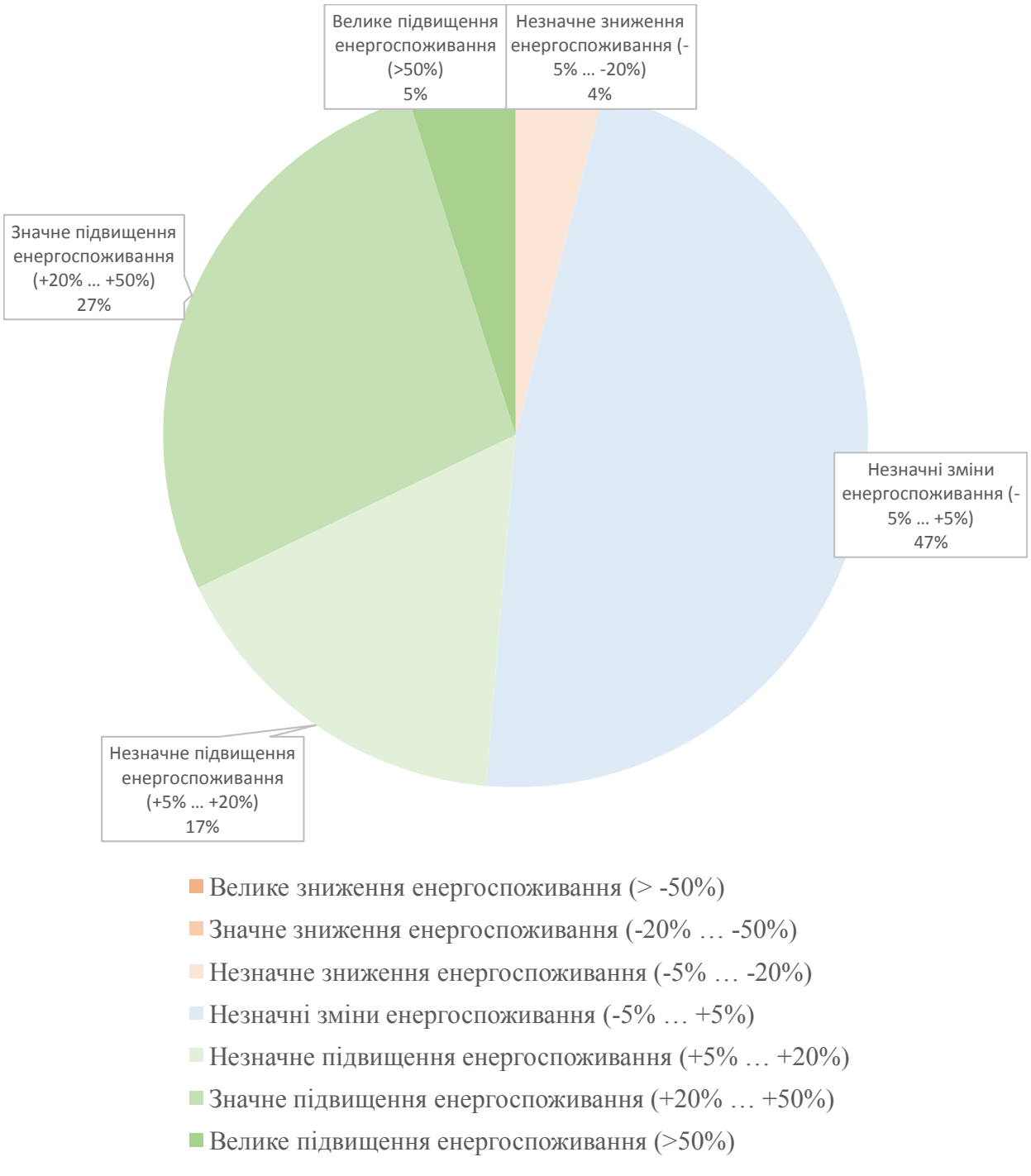

Рисунок 2.13 – Статистика розподілу зміни енергоспоживання обчислень ТП при застосуванні методів розбиття на блоки та розпаралелюванні для плати Raspberry Pi 3

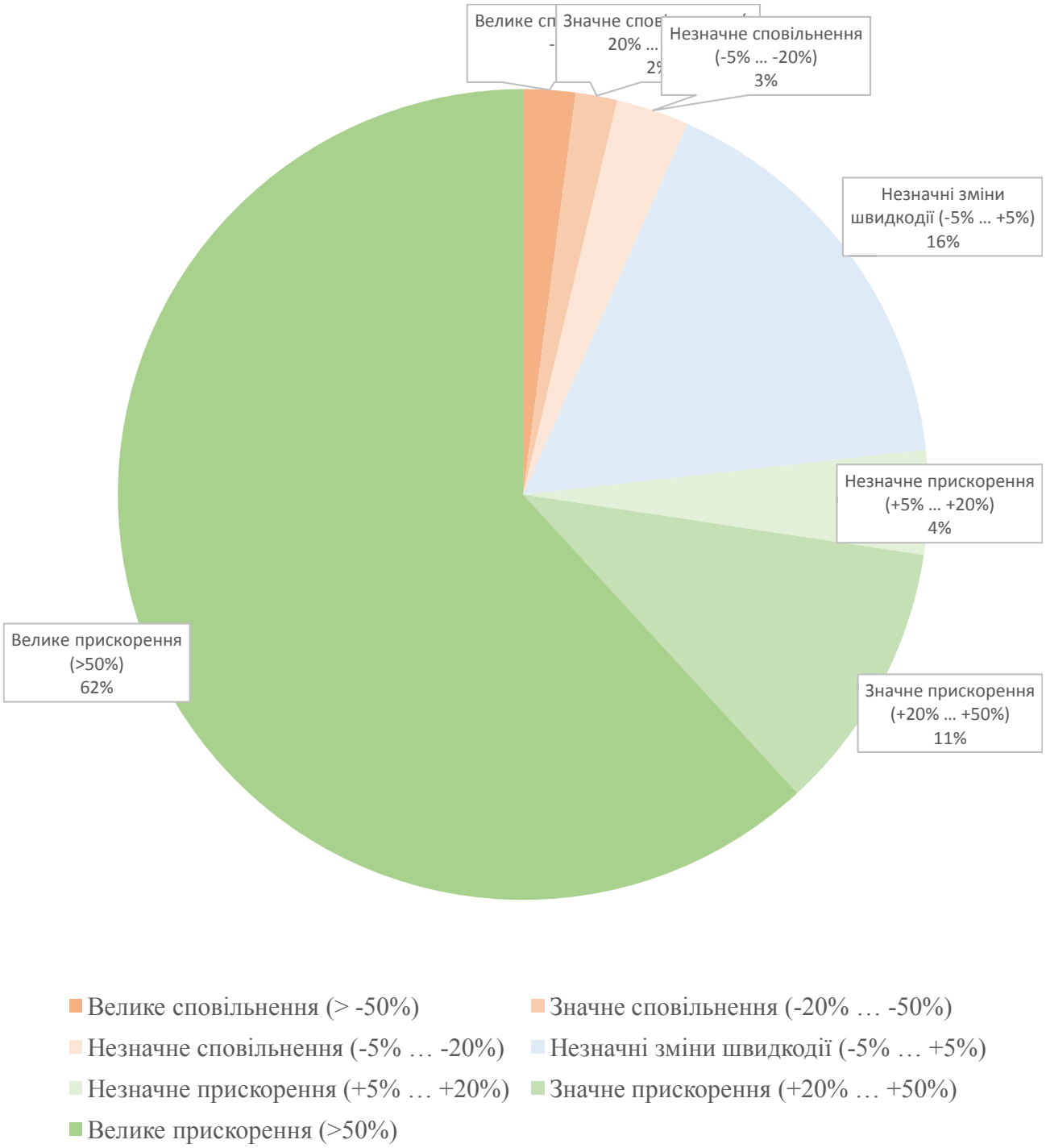

Рисунок 2.14 – Статистика розподілу зміни часу виконання ТП при застосуванні методів розбиття на блоки та розпаралелюванні для ПК на базі Intel® Core™ i5-4670K

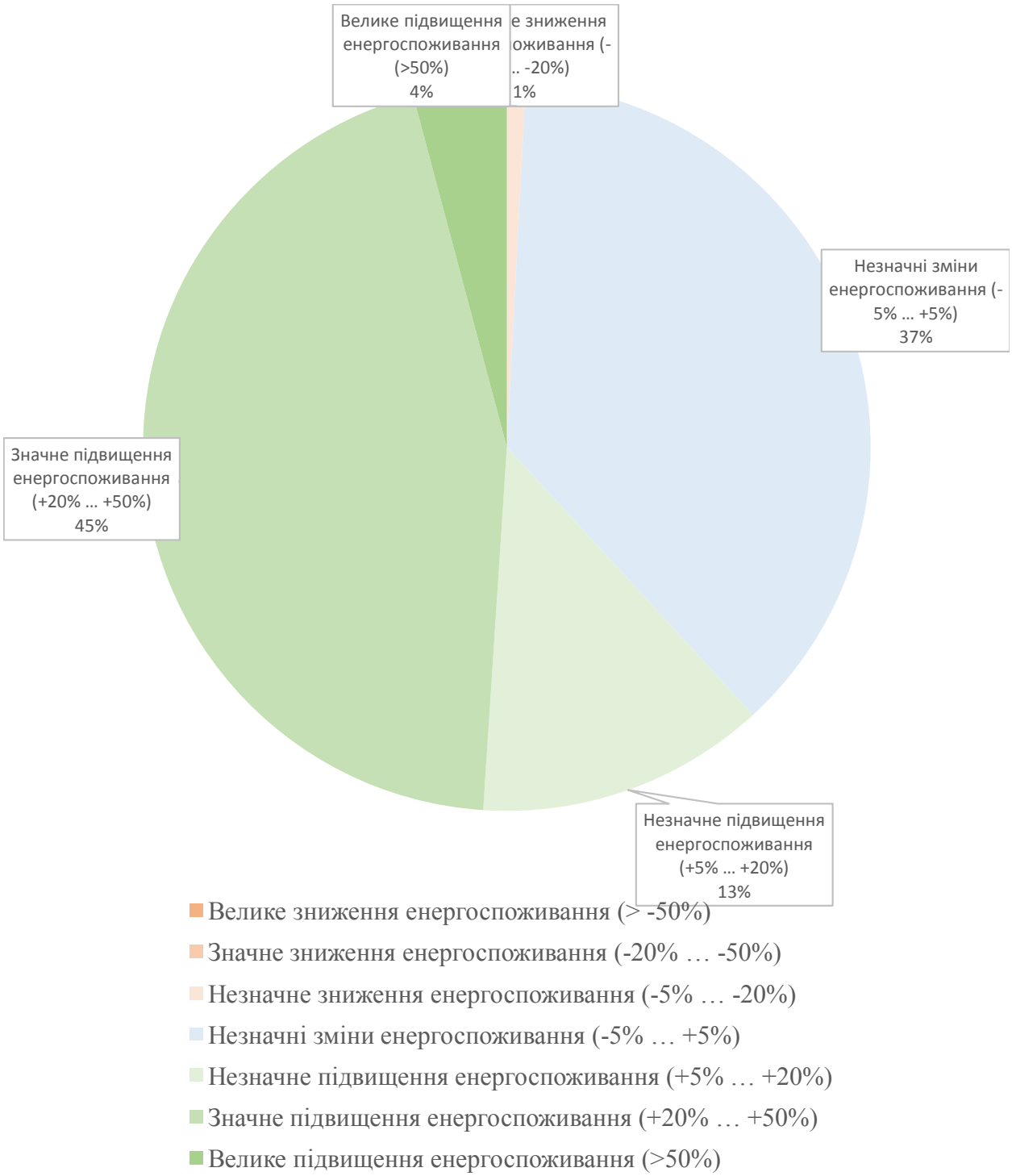

Рисунок 2.15 – Статистика розподілу зміни енергоспоживання обчислень ТП при застосуванні методів розбиття на блоки та розпаралелюванні для ПК на базі Intel® Core™ i5-4670K

Узагальнюючи отримані дані варто відзначити, що більшість ТП скоротила час виконання та дещо підвищила енергоспоживання обчислень при використанні методів розбиття на блоки. Водночас є низка ТП, час виконання

котрих майже не змінюється при використанні будь-яких методів розбиття на блоки та розпаралелюванні.

2.6 Енергоефективність обчислень для двох різних апаратних платформ

Експерименти 1-4 виконувались для тих самих тестових програм з такою ж розмірністю даних, проте на різних апаратних платформах, що дає можливість порівнювати енергоефективність обчислень. Для цього розрахуємо необхідну електричну енергію як добуток електричної потужності на час виконання програми згідно формули 2.8:

$$E = Pt. \tag{2.8}$$

Застосуємо формулу до результатів вимірювання на двох апаратних платформах, що було представлено в таблицях 2.1-2.4. Після цього визначимо відношення необхідних електричних енергій для двох платформ. Отримані дані наведено в таблиці 2.5.

Порівняння споживаної електричної енергії, необхідної для обчислень тестових програм на платформах на базі Raspberry Pi 3 та Intel® Core™ i5-4670K дає можливість стверджувати, що в 26 випадках із 28 для обчислення тієї ж програми на Raspberry Pi 3 потрібно використати менше електричної енергії. При цьому варто зазначити, що тривалість обчислень на Raspberry Pi 3 більше якнайменше в 6 разів ніж на платформі x64.

Виходячи з аналізу результатів обчислень, отриманих в експериментах 1-4 стає можливо стверджувати, що використання МРОЦНБ та розпаралелювання доцільне на більшості задач з метою прискорення швидкодії та підвищення енергетичної ефективності обчислень.

2.7 Залежність часу виконання тестових програм від значень розмірів блоків розбиття

Аналізуючи результати вимірів двох експериментів можна звернути увагу на те, що метод розбиття на блоки параметричний, тобто має параметри своєї роботи, власне це розміри блоків розбиття. Зважаючи на те, що в загальному випадку вони можуть бути будь якими цілими числами, а

безпосередньо в тесті використовувалась за замовчуванням одна пара значень, а саме 32 та 32, то з цього випливає практичне запитання, чи вплине на швидкість обчислень зміна розмірів блоків у деяких діапазонах значень.

Для перевірки цієї гіпотези проведено експеримент 5, в якому для кожної з тестових програм на настільному ПК на базі чотириядерного процесора Intel® Core™ i5 4670K з 16ГБ оперативної пам’яті виміряно час виконання ТП для багатьох різних розмірів блоків розбиття. Експерименти проводилися для двох комбінацій – методу розбиття на блоки та для розпаралелювання одночасно з розбиттям на блоки. Результати отриманих даних наведено в додатках Б і В відповідно. Приклади отриманих залежностей зображені на Рисунках 2.16-2.19.

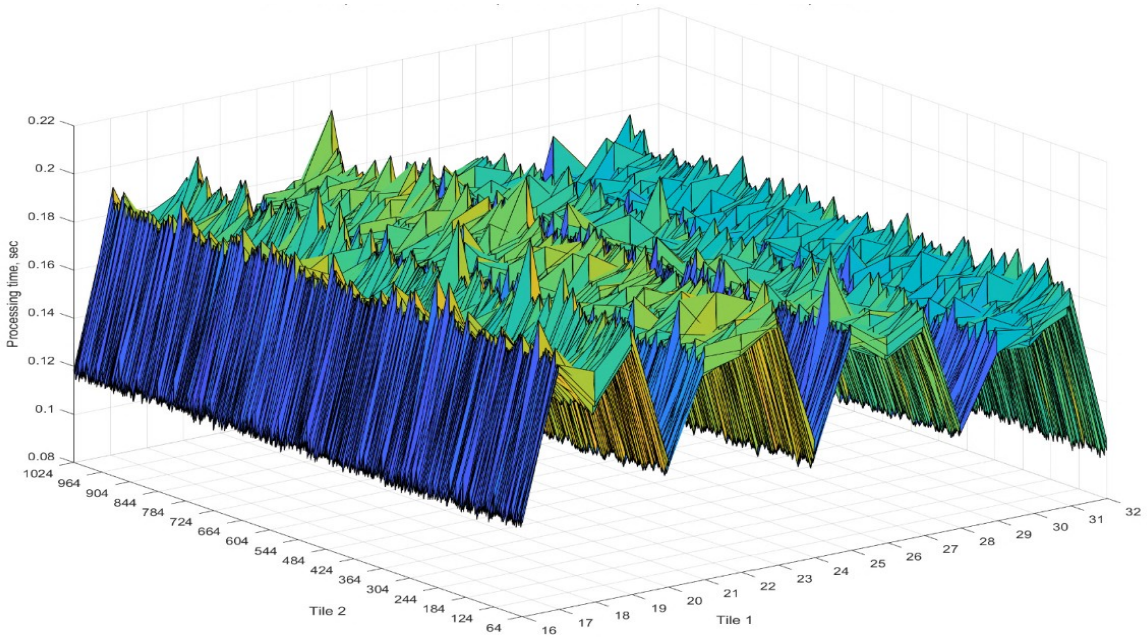

Рисунок 2.16 – Приклад 1 залежності часу виконання від розмірів блоків розбиття

Таблиця 2.5 – Порівняння необхідної електричної енергії, необхідної для розрахунку тестових програм для двох апаратних платформ.

Тестова програма	**Intel Core i5-4670K**			**Raspberry Pi 3**			
	Час виконання, с	**Електрична потужність, Вт**	**Споживана електрична енергія E1, Вт*с**	**Час виконання, с**	**Електрична потужність, Вт**	**Споживана електрична енергія E2, Вт*с**	$\frac{E_1}{E_2}$
correlation	0,0088	109	0,954	0,344	1,880	0,647	1,47
covariance	0,0086	111	0,951	0,352	1,875	0,661	1,44
gemm	0,7605	116	88,224	22,789	1,895	43,186	2,04
gemver	0,3985	105	41,839	4,826	1,870	9,024	4,64
gesummv	0,0240	106	2,543	0,418	1,905	0,797	3,19
symm	0,0112	109	1,223	0,411	1,890	0,777	1,57
syr2k	0,0160	110	1,756	0,996	1,865	1,857	0,95
syrk	0,0043	111	0,482	0,190	1,800	0,343	1,41
trmm	0,0044	109	0,483	0,156	1,825	0,285	1,69
2mm	0,0133	109	1,452	0,432	1,820	0,786	1,85
3mm	0,0203	108	2,191	0,644	1,835	1,182	1,85
atax	0,0054	112	0,601	0,187	1,890	0,354	1,70
bicg	0,0113	106	1,195	0,192	1,895	0,364	3,28
doitgen	0,4980	112	55,775	10,445	1,845	19,271	2,89
mvt	0,0618	104	6,425	0,506	1,850	0,936	6,86
cholesky	0,0083	107	0,886	0,189	1,950	0,368	2,41
durbin	0,0094	108	1,017	0,292	1,885	0,550	1,85
gramschmidt	0,0152	110	1,673	0,451	1,880	0,848	1,97
lu	0,0185	108	1,994	0,863	1,975	1,705	1,17
ludcmp	0,0177	114	2,017	0,588	1,975	1,161	1,74
trisolv	0,0094	112	1,053	0,190	1,935	0,368	2,86
deriche	0,7520	104	78,209	4,229	1,955	8,267	9,46
floyd-warshall	0,0819	108	8,842	1,589	2,040	3,241	2,73
nussinov	0,0217	105	2,273	0,407	1,850	0,752	3,02
fdtd-2d	0,0100	115	1,146	0,639	2,005	1,280	0,89
heat-3d	0,0205	109	2,236	0,914	1,885	1,722	1,30
jacobi-2d	0,0142	110	1,557	0,590	1,975	1,164	1,34
seidel-2d	0,1531	100	15,309	1,344	1,780	2,392	6,40

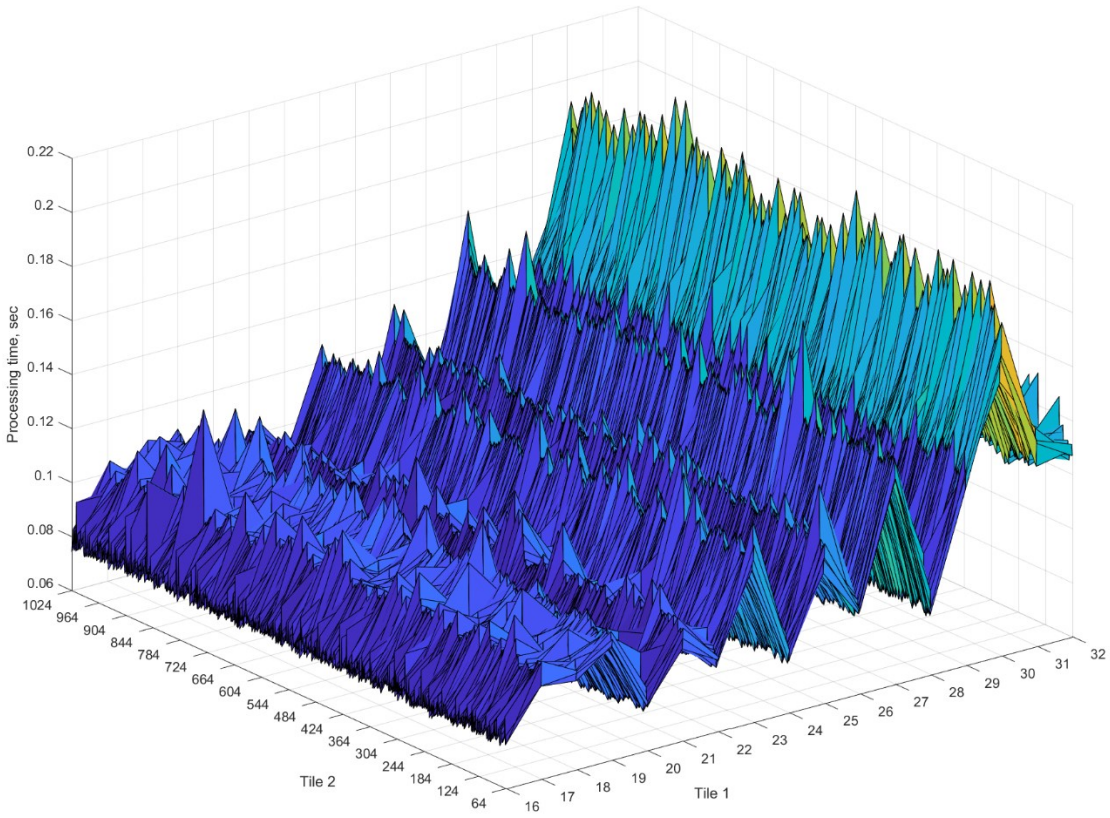

Рисунок 2.17 – Приклад 2 залежності часу виконання від розмірів блоків розбиття

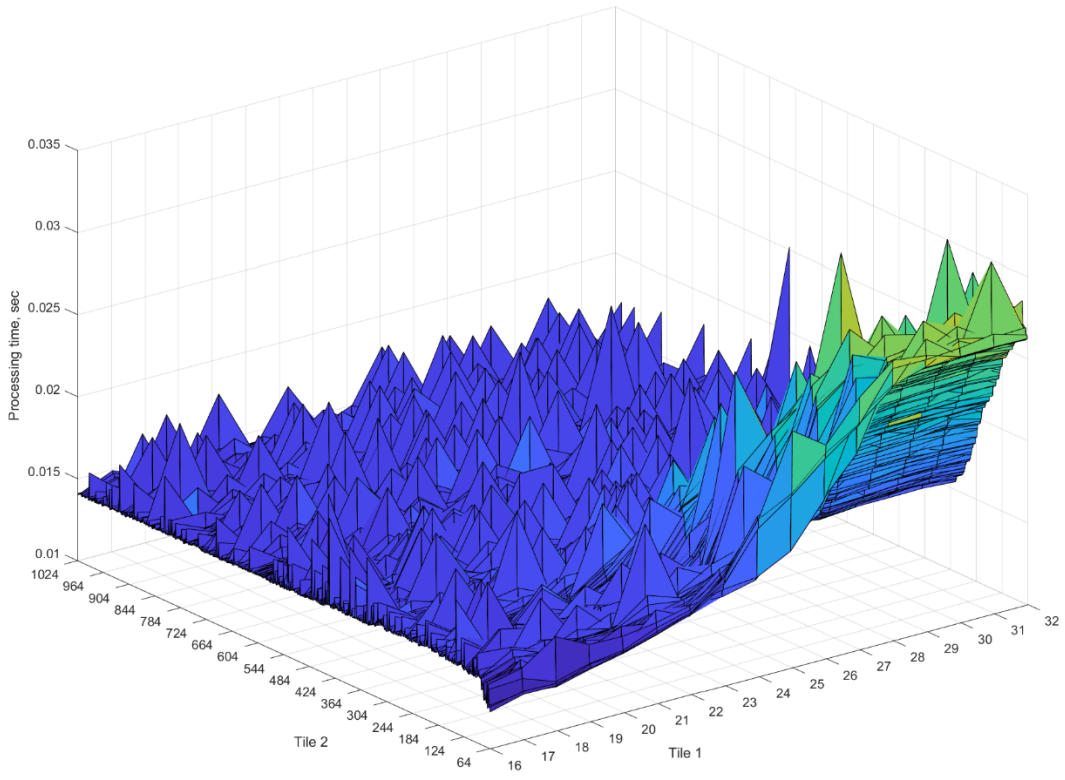

Рисунок 2.18 – Приклад 3 залежності часу виконання від розмірів блоків розбиття

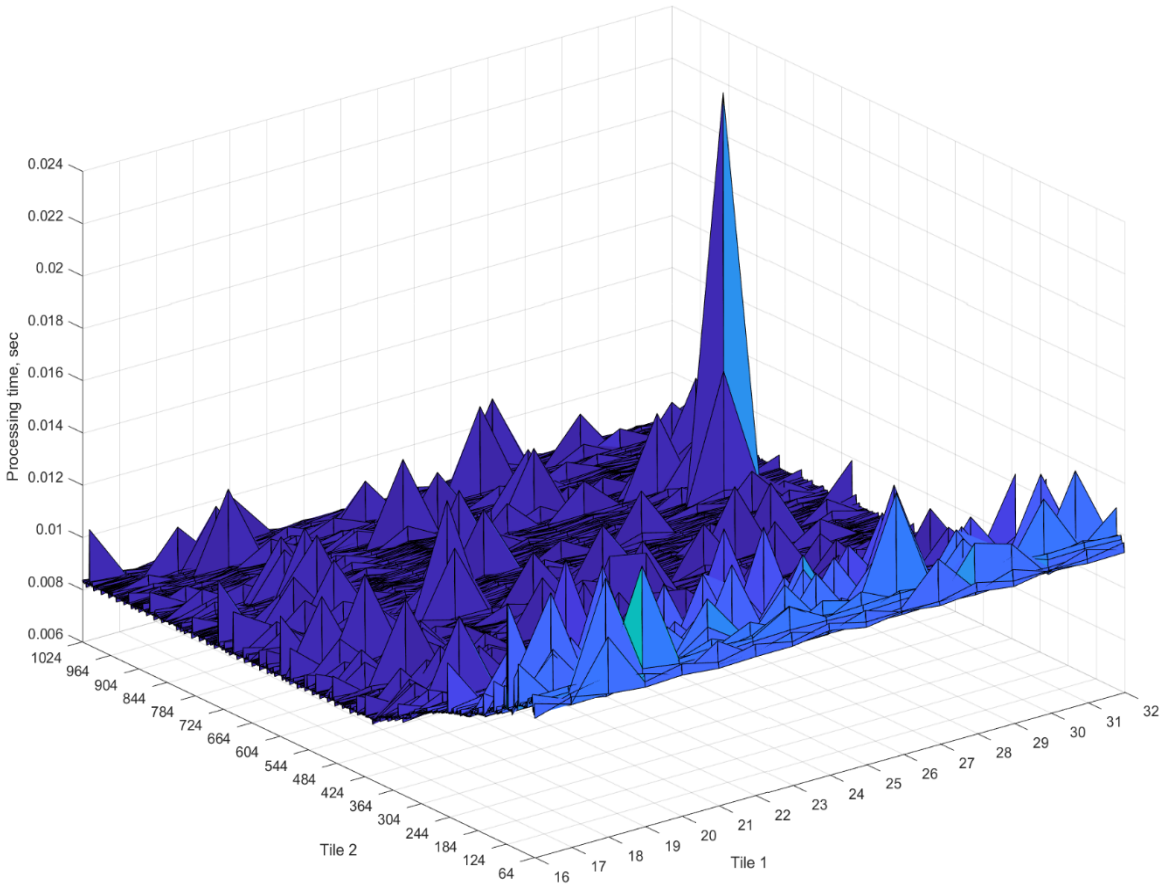

Рисунок 2.19 – Приклад 4 залежності часу виконання від розмірів блоків розбиття

Отримані результати значно відрізняються в залежності від розмірів блоків розбиття та тестової програми. Залежності мають складний характер і не мають схожих патернів для різних тестів.

В деяких випадках, наприклад таких, як зображено на Рисунку 2.19, час виконання програм може значно зростати на деяких значеннях. Це викликано роботою операційної системи, що виконує фонові процеси під час вимірювання часу виконання основних програм. Такі значення є хибними, вони зменшуються при повторному вимірі і тому не мають впливати на загальні висновки. Задля уникнення таких факторів використовується п’ятикратне вимірювання з усередненням трьох середніх значень. Однак, в деяких випадках, внутрішні процеси операційних систем тривають відносно довгий час і це призводить до високих піків часу виконання ТП. Частіше такий ефект спостерігається для ТП, в яких задіяно розпаралелювання. В такому

випадку задіються в роботі всі наявні ядра і найменші додаткові навантаження на систему призводять до зниження часу виконання ТП. Якщо задіяне тільки одне ядро мікропроцесора для обчислення ТП, фонові процеси не впливають суттєво на час виконання ТП.

Враховуючи складну залежність часу виконання від розміру блоків введемо дві оцінки для кожної з характеристик. Перша оцінка – відношення середнього часу виконання до мінімального, друга – відношення максимального часу виконання до мінімального. Такі оцінки дозволяють оцінити потенційне можливе прискорення для випадково вибраного розміру блоку в середньому та максимально можливе прискорення часу виконання. Отримані дані наведено в таблиці 2.6.

Таблиця 2.6 – Відношення часів виконання тестових програм для різних розмірів блоків розбиття.

Тестова програма	**Tile**		**Tile + Innerpar + Parallel**	
	Середній/ Мінімальний	**Максимальний/ Мінімальний**	**Середній/ Мінімальний**	**Максимальний/ Мінімальний**
2mm	1.02	2.49	1.51	43.77
3mm	1.02	1.46	1.39	13.74
atax	1.13	1.78	1.46	32.85
bicg	1.17	2.30	1.42	46.51
cholesky	1.05	1.55	2.74	28.62
covariance	1.08	1.87	2.03	254.57
doitgen	1.64	2.29	1.52	2.99
durbin	1.01	1.58	1.01	2.03
gemm	1.13	1.74	1.22	2.41
gemver	1.21	1.54	1.24	5.86
gesummv	1.25	2.75	1.29	13.75
gramschmid	1.65	2.63	2.05	16.28
lu	1.11	2.96	2.22	48.82
mvt	1.20	2.43	1.33	8.23
symm	1.02	1.67	1.02	1.57
syr2k	1.20	1.97	1.53	15.84
syrk	1.04	2.28	1.62	28.84

Аналіз отриманих даних дає підстави стверджувати, що розмір блоків загалом суттєво впливає на час виконання програм. Час однопотокового розбиття на блоки, отриманий при використанні випадкових значень блоків,

зазвичай більше до 20% чим мінімальний час. В деяких випадках цей час може бути більше до 65%. Час багатопотокового розбиття на блоки, отриманий при використанні випадкових значень блоків, зазвичай більше до 40% мінімального часу. В деяких випадках цей час може бути більшим в два рази. В обох конфігураціях найгірший час виконання у випадку, коли розміри блоків були обрано дуже невдало може бути прискорено в декілька разів.

Варто відмітити те, що характер залежностей дуже неоднорідний. Не можна виділити один чи декілька шаблонів залежностей часу виконання від розмірів блоків. Цей факт суттєво ускладнює знаходження оптимальних значень. Найкращі значення блоків розбиття не можуть бути визначені до тестування або на основі деяких значень.

Як слідує з результатів вимірювання, середнє значення часу виконання програм може бути на 1-65% більше ніж найкраще можливе, а максимальний час виконання більше на 67-296% від мінімального для однопотокового методу розбиття на блоки. Якщо разом з методом розбиття на блоки використовується метод розпаралелювання, то різниця загалом ще більша – для середнього 1-174% та 57-2530% максимального. Дуже низькі значення швидкодії для деяких з тестів в методі розпаралелювання пояснюються внутрішніми процесами операційної системи ПК, що короткостроково потребує багато обчислювальних ресурсів. Це негативно впливає на час виконання тестової програми в цей час.

Таким чином, створюються передумови для подальшого прискорення виконання програм шляхом кращого підбору розмірів блоків розбиття.

2.8 Висновки до Розділу 2

У розділі розглянуто теоретичні засади та існуючи методи оптимізації ПЗ. Наведено можливі підходи та методи, що спрямовані на оптимізацію обчислювальних циклів. Особливу увагу приділено поліедральній моделі та ітеративному простору як ефективним засобам оптимізації обчислювальних

циклів. Розглянуто напрямок оптимізації, що виконує S2S перетворювання. Наведено переваги та недоліки цього підходу.

Розглянуто метод розбиття обчислювальних циклів на блоки, що виконує перетворення обчислювальних циклів. Для отримання даних було створено та використано ПЗ, що виконує вимірювання часу виконання ТП в автоматичному режимі.

Запропоновано ітераційний процес розпаралелювання операторів циклів мікропроцесорних програм, що може бути використаний для одночасного розпаралелювання та розбиття ітераційного простору на блоки з метою покращення цільової функції оптимізації ПЗ. Запропонований ітераційний процес дозволяє в автоматичному режимі досягти оптимальних параметрів розпаралелювання операторів циклів програм.

Отримано експериментальні дані, що підтверджують ефективність методу розбиття на блоки на більшості тестових задач як з точки зору часу виконання програм так і з точки зору енергоефективності обчислень. З метою оцінки впливу розмірів блоків розбиття на час виконання програм було виконано виміри часу виконання для різних розмірів блоків розбиття. Отримані результати дають підстави стверджувати що:

- характер залежностей між розмірами блоків розбиття і часом виконання для різноманітних ТП кардинально різний і не має специфічних залежностей чи патернів;
- підбір кращих розмірів блоків розбиття призведе ще до більш швидкого часу виконання програм;
- для підбору розмірів блоків розбиття потрібно використовувати алгоритм, що зважаючи на складну залежність, буде здатний знайти глобальний або локальний мінімум часу виконання.

Отримані експериментальні дані дають змогу стверджувати, що більш ретельний підбір розмірів блоків розбиття приведе до додаткового збільшення ефективності відносно розмірів блоків, які було використано за замовчуванням.

РОЗДІЛ 3. ЗАДАЧА ПОШУКУ ОПТИМАЛЬНИХ РОЗМІРІВ БЛОКІВ РОЗБИТТЯ

3.1 Метод рою часток

Наявність великої розбіжності між часом виконання для різних розмірів блоків розбиття створює можливість досягти прискорення часу виконання програм за рахунок правильного підбору оптимальних розмірів блоків.

Виходячи зі складної залежності часу виконання КП від розмірів блоків розбиття, що випливає з Рисунків 2.16-2.19 пропонується використання дискретного методу рою часток як алгоритму пошуку мінімуму часу виконання. Головна перевага даного методу для вирішення поставленої задачі – наявність багатьох агентів пошуку, що дозволяє охоплювати пошуковий діапазон найкращим чином, уникаючи пошуку тільки навколо локального мінімуму [7].

Перед розглядом алгоритмів ройового інтелекту як системи [120], [108], [110], [111], [112], [113], потрібно ввести позначення:

$f(\mathbf{X})$ – скалярна цільова функція, для якої потрібно знайти максимальне або мінімальне значення, для визначеності подальшого опису будемо вважати, що якщо це не обумовлено, то потрібно знайти мінімальне;

$\mathbf{X}$ – вектор змінних параметрів, від яких залежить цільова функція;

D – область допустимих значень $\mathbf{X}$, $D \in R^{|\mathbf{X}|}$, простір пошуку рішень;

$G(\mathbf{X})$ – функція, що задає обмеження на $\mathbf{X}$;

$|S|$ – кількість агентів рою;

$S = \{s_1, s_2, \ldots, s_{|S|}\}$ – множина всіх агентів рою;

$\mathbf{X}_{ij}$ – вектор змінних параметрів i-го агенту на j-й ітерації алгоритму, іншими словами, положення агента, його позиція.

$\mathbf{X}_{ij}^{best}$ – найкраще положення і-го агента від 1-ї до j-й ітерації алгоритму;

$\mathbf{X}^{opt}$ – оптимальне значення вектора змінних параметрів (оптимальне рішення);

f^{opt} – оптимальне значення цільової функції;

$\varphi(\mathbf{X}) = f(\mathbf{X})$ – значення фітнес-функції в положенні **X**;

$\mathbf{X}_j^{best}$ – найкраще значення вектора змінних параметрів, яке було отримано серед усіх агентів від *1*-ї до *j*-й ітерації алгоритму;

$\mathbf{X}_{final}^{best}$ – найкраще значення вектора змінних параметрів, знайдене роєм до моменту завершення роботи (квазіоптимальне рішення), використання евристичних методів не гарантує рівність $\mathbf{X}_{final}^{best}$ *і* $\mathbf{X}^{opt}$;

$S_j^{best}(n)$ – *n* агентів рою, що займають на j-й ітерації найкращі положення, тобто таких що $\varphi(\mathbf{X}ij) \geq \varphi(\mathbf{X}kj)$, $s_i \in S_j^{best}(n)$, $sk \in S \setminus S_j^{best}(n)$.

$\boldsymbol{C}_j = c(Sj) = c(\mathbf{X}1j, \mathbf{X}2j, \ldots, \mathbf{X}|s|j, \varphi(\mathbf{X}1j), \varphi(\mathbf{X}2j), \ldots, \varphi(\mathbf{X}|s|j))$ – точка «центру ваги» рою, деяка позиція **X**, отримана усередненням положень всіх агентів з урахуванням їх фітнес-функцій.

Розглядається задача мінімізації:

$$f^{opt} = f(\boldsymbol{X}^{opt}) = \min_{X \in D} f(\boldsymbol{X}). \qquad (3.1)$$

Схема алгоритмів ройового інтелекту включає в себе наступні етапи.

1. Генерація початкових станів агентів. Деяким чином в просторі пошуку розподіляються агенти. Номер ітерації $j = 1$.

2. Обчислення фітнес-функції для кожного з агентів. Для більшості алгоритмів ройового інтелекту для цього необхідно просто обчислити $\varphi(\mathbf{X}_{ij}) = f(\mathbf{X}_{ij})$. Для інших, наприклад, колонії мурах, обчислення цільової функції можливо тільки після реалізації евристичного поведінки агентів, для них крок 2 при j = 1 пропускається.

3. Міграція (переміщення). На цьому кроці реалізується головна особливість алгоритмів ройового інтелекту – виконання кожним агентом своїх дій на підставі:

– своїх власних правил;

– правил, що реалізують непрямий обмін з іншими особинами;

– стохастичного поведінки.

4. Перевірка умови завершення. Якщо умова виконана, процес завершується, значення $\mathbf{X}_{final}^{best}$ буде кінцевим результатом, інакше відбувається перехід до кроку 2 (зі збільшенням номера ітерації j на одиницю).

Умовами завершення роботи алгоритму можуть бути: досягнення заданого числа ітерацій; знаходження рішення не гірше деякого наперед заданого; стагнація процесу (коли $\mathbf{X}_j^{best}$ не покращується протягом заданого числа ітерацій). Крім того, при реалізації даних алгоритмів слід передбачити можливість користувачу переривати процес пошуку в будь-який момент або продовжувати його необмежено довгий час.

Аналіз кроку 3 дозволяє визначити, чи є даний алгоритм алгоритмом ройового інтелекту. Як правило, в формулах, що визначають поведінку агентів, є або елемент, пов'язаний одним ($\mathbf{X}_j^{best}$) або декількома найкращими положеннями ($S_j^{best}(n)$), які були знайдені всім роєм (непрямий обмін інформацією про своїх рішеннях), або «центр ваги» ($\boldsymbol{C}_j$). Наприклад, в алгоритмі рою часток для організації обміну інформацією між агентами використовується $\mathbf{X}_j^{best}$, алгоритмі рою бджіл – $S_j^{best}(n)$, в мавпячому пошуку – $\boldsymbol{C}_j$. Більш складним є мурашиний алгоритм, де для непрямого обміну досвідом використовується граф, ваги дуг якого змінюються в залежності від того, наскільки гарне рішення було отримано під час руху по ним. В еволюційних алгоритмах непрямий обмін відсутній, він замінюється процесом відбору агентів з найкращим значенням фітнес-функції. До таких алгоритмам відносяться генетичний, бур’яновий, алгоритм зростаючих дерев. Так, наприклад, канонічний алгоритм бактеріальної оптимізації не відноситься до алгоритмів ройового інтелекту, але модифікація, в яку введені правила тяжіння і відштовхування бактерій один від одного, є алгоритмом ройового інтелекту, про що свідчить і її назва «алгоритм роїння бактерій».

Алгоритм рою часток (Particle Swarm Optimization – PSO) був спочатку розроблений для моделювання соціальної поведінки і заснований на поведінці зграй птахів [47], [115], [116].

Зграя птахів завжди діє скоординовано, кожна птиця діє згідно простим правилам, стежить за іншими птахами і узгоджує свій рух з ними. Знайшовши джерело їжі, птиця повідомляє про нього всю зграю. Саме цей факт створює колективну поведінку і ройовий інтелект. Джерела їжі зазвичай розташовані

випадковим чином і одному птахові дуже складно швидко знайти їх. Тільки в тому випадку, якщо птахи будуть обмінюватися інформацією, вся зграя зможе вижити.

При переході до моделі слова «птах» замінено на слово «частка» і вводяться наступні положення:

– частки існують в світі, де час дискретний;

– частки оцінюють своє становище за допомогою фітнес-функції;

– кожна частка знає позицію в просторі, в якій вона знайшла найбільшу кількість їжі (своя найкраща позиція);

– кожна частка знає позицію в просторі, в якій знайдено найбільшу кількість їжі серед всіх позицій, в яких були всі частки (загальна найкраща позиція);

– частки мають тенденцію прагнути до кращих позицій, в яких були самі і до загальної найкращій позиції;

– частки випадковим чином змінюють свою швидкість, так що описана тенденція визначає лише усереднене рух часток;

– частки мають інерцію, тому їх швидкість в кожен момент часу залежить від швидкості в попередній момент;

– частки не можуть покинути задану область пошуку.

Основна ідея алгоритму полягає в переміщенні часток в просторі рішень. Кожна частка «пам'ятає» найкращу точку в просторі рішень, в якій була, і прагне до неї повернутися, але підпорядковується також закону інерції і має схильність до невеликої зміни напрямку руху. Однак цих правил недостатньо для переходу до системи, так як не задані зв'язки між елементами. Як зв'язку використовується так звана спільна пам'ять, суть якої в тому, що кожна частка знає координати найкращою точки серед усіх, в яких була будь-яка частка рою. Таким чином, найкраще рішення, знайдене роєм, в кожен момент часу відоме всім його агентам. У підсумку на рух частки впливають прагнення до свого найкращого стану, прагнення до найкращого серед всіх часток положенню, інерційність і випадкові відхилення [41].

Формула для класичного методу рою часток наведена нижче в (3.2):

$$v_{i,t+1} = wv_{i,t} + c_1 r_1 (m_i - x_{i,t}) + c_2 r_2 (g - x_{i,t}), \quad (3.2)$$

де $v_{i,t}$ – швидкість i частки на t ітерації;

$x_{i,t}$ – координата i частки на t ітерації;

c_1, c_2 – коефіцієнти відповідно індивідуальної та соціальної поведінки часток;

r_1, r_2 – випадкові величини у діапазоні від 0 до 1, що обчислюються для кожної частки на кожній ітерації;

m, g – локальний і глобальний мінімуми;

w – інерція швидкості;

$i=1..N$ – кількість часток (розмір популяції);

$t=1..M$ – кількість ітерацій.

Суть методу зводиться до ітеративного пошуку мінімуму функції кожною часткою, що інформаційно пов’язані. Початкові координати часток в оригінальному МРЧ задаються випадково у межах вибраного діапазону пошуку. Ройові методи використовуються для вирішення широкого класу задач [128].

В роботі використано дискретний метод рою часток, оскільки координатами вузлів ітераційного простору можуть бути зазвичай цілі числа. Для такого методу використовується та сама формула, що і для класичного методу рою часток (3.2), проте додаються обмеження – $x_{i,t} \in Z$ та $v_{i,t} \in Z$. Для виконання цих умов початкові координати часток обираються цілими і на кожній ітерації значення швидкості округляється до цілого.

3.2 Застосування дискретного методу рою часток для підбору кращих розмірів блоків розбиття

З метою визначення швидкості та ефективності пошуку найкращих розмірів блоків розбиття, проведено серію експериментів щодо використання дискретного методу рою часток.

Досліджено пошук параметрів розбиття на прикладі зображеному на Рисунку 3.1.

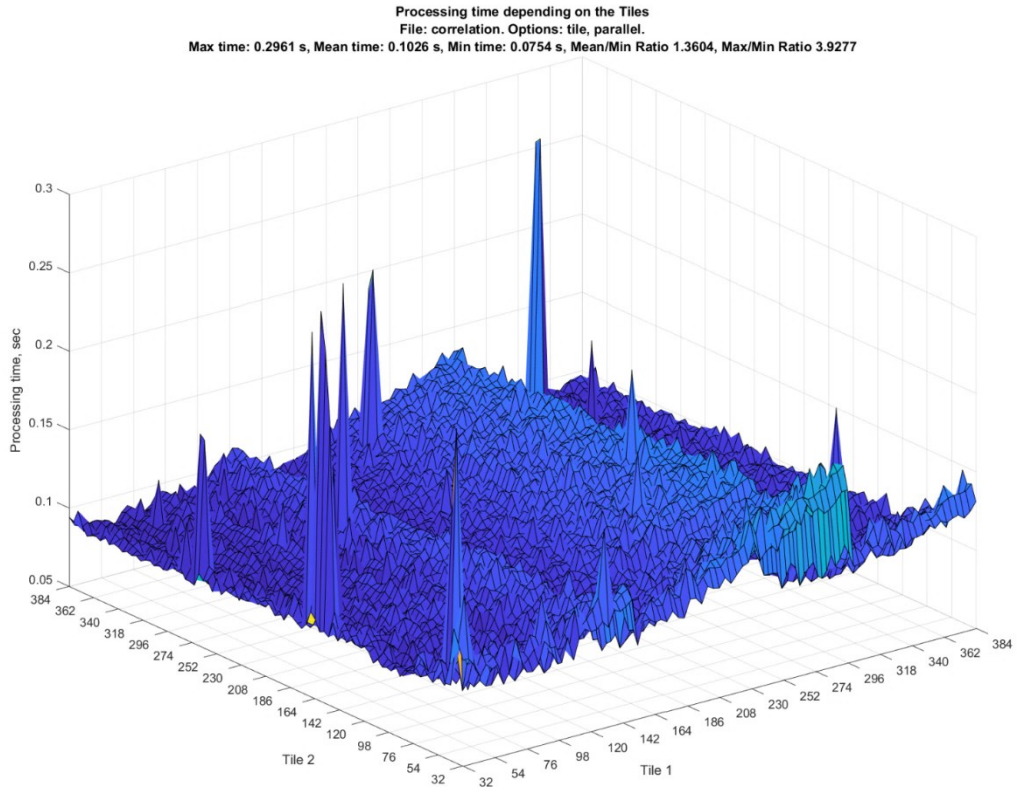

Рисунок 3.1 – Тестовий приклад для пошуку оптимальних параметрів розбиття

Пошук найкращих розмірів блоків розбиття виконувався дискретним методом рою часток для різної кількості часток і різної кількості ітерацій. Отримані дані наведено в таблиці 3.1.

Отримані дані засвідчують прискорення часу виконання при збільшенні кількості ітерацій та задіяних часток. В той же час, найкращі результати отримуються при значній кількості ітерацій та часток. Це означає, що в практичному використанні потребуватиметься багато вимірів, і як наслідок, часу, щоб отримати кращий результат пошуку. Також варто звернути увагу, що при подальшому підвищенні кількості ітерацій пошук кращого часу виконання не відбувається, тобто знайдені на перших ітераціях параметри демонструють значну ефективність і не прискорюються в подальшому.

Таблиця 3.1 – Результати пошуку мінімального часу обчислення при різній кількості ітерацій і кількості часток ДМРЧ.

Кількість часток	**Кількість ітерацій**	**Мінімум часу виконання програм, с**	**Кількість часток**	**Кількість ітерацій**	**Мінімум часу виконання програм, с**
4	4	0.083637	128	4	0.077609
	8	0.083637		8	0.077609
	16	0.079986		16	0.077217
	32	0.079523		32	0.077086
	64	0.079430		64	0.077086
	128	0.078543		128	0.077086
	256	0.078543		256	0.077086
	512	0.078543		512	0.077086
	1024	0.078543		1024	0.076744
8	4	0.083637	256	4	**0.076955**
	8	0.083637		8	0.076744
	16	0.081941		16	0.076744
	32	0.080030		32	0.076744
	64	0.079785		64	0.076744
	128	0.078543		128	0.076744
	256	0.078543		256	0.076744
	512	0.078543		512	0.076744
	1024	0.078543		1024	0.076744
16	4	0.078953	512	4	0.076955
	8	0.077609		8	0.076955
	16	0.077609		16	0.076955
	32	0.077609		32	0.076744
	64	0.077609		64	0.076744
	128	0.077609		128	0.076744
	256	0.077609		256	0.076744
	512	0.077609		512	0.076744
	1024	0.077609		1024	0.076744
32	4	0.079196	1024	4	0.076955
	8	0.078119		8	0.076955
	16	0.078119		16	0.076744
	32	0.077910		32	0.076744
	64	0.077609		64	0.076744
	128	0.077609		128	0.076744
	256	0.077086		256	0.076744
	512	0.077086		512	0.076744
	1024	0.077086		1024	0.076744
64	4	0.079041			
	8	0.079041			
	16	0.077609			
	32	**0.076950**			
	64	0.076744			
	128	0.076744			
	256	0.076744			
	512	0.076744			

3.3 Вплив параметрів дискретного методу рою часток на швидкість підбору кращих розмірів блоків розбиття

Оскільки метод рою часток є параметричним, заслуговує уваги дослідження щодо підбору параметрів метода з метою зниження ітерацій пошуку [126].

Згідно формули (3.1), що визначає ДМРЧ, існує декілька змінних параметрів методу. Насамперед, це коефіцієнт інерції швидкості, коефіцієнти індивідуальної та соціальної поведінки часток.

Використовуючи різні коефіцієнти методу, а саме: коефіцієнти індивідуальної і соціальної поведінок, інерцію швидкості можна корегувати поведінку часток, визначаючи їх збіжність до глобального мінімуму або пошук в околі власного локального мінімуму.

Також варто відзначити, що важливою є початкова позиція часток. В оригінальному МРЧ частки початкова позиція часток є випадкова в середині області пошуку. Однак, в різних роботах присвячених дослідженню МРЧ [111], [112] пропонуються інші підходи щодо початкових координат часток. Так, вони можуть розміщуватись по краям або рівномірно в середині визначеної області або мати інші схеми початкового розташування.

Оскільки до теперішнього часу ДМРЧ не використовувався для пошуку найкращих розмірів блоків розбиття, таке дослідження було виконано самостійно. Аналіз впливу початкового розміщення часток на кількість ітерацій показав, що ДМРЧ добре справляється з випадками, коли глобальний мінімум знаходиться глибоко в середині області пошуку. У випадках, коли глобальний мінімум знаходиться ближче до границі або на самій границі області пошуку, пошук цього мінімуму може не відбутися не зважаючи на кількість ітерацій. Таким чином, початкове розміщення часток було прийнято рівномірним на границі області пошуку з початковою ненульовою інерцією швидкості часток, що спрямована в середину області пошуку.

Визначення найкращих коефіцієнтів методу рою часток проводилося експериментально шляхом оцінки поведінки часток для різних значеннях

коефіцієнтів для 8 часток на 8 ітерацій, що дозволяє візуально оцінити поведінку часток. Отримані залежності наведені в додатку Г.

Приклади отриманих залежностей наведено на Рисунках 3.2-3.5.

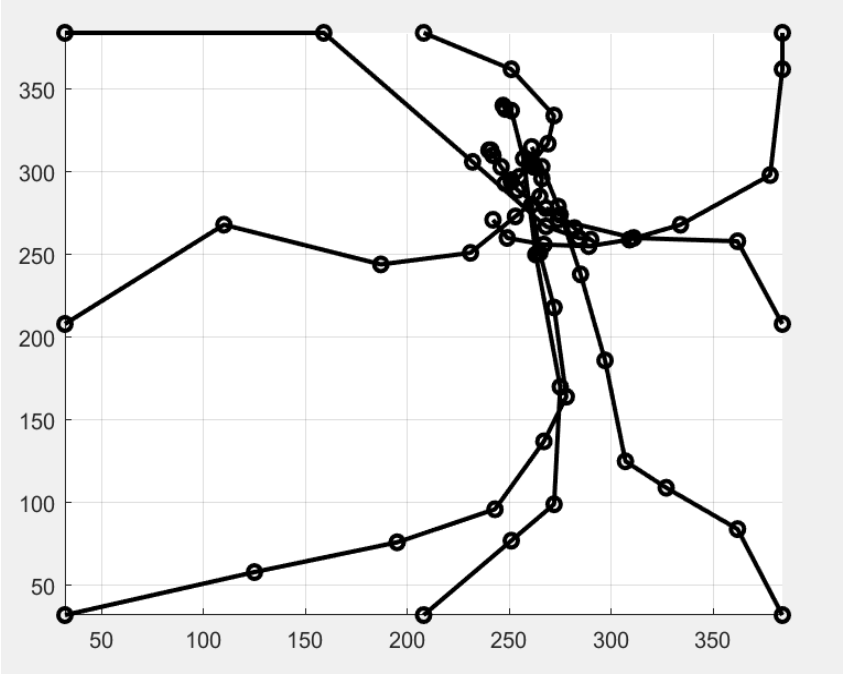

Рисунок 3.2 – Результати при w=0.5, $c_1 = 0$, $c_2 = 0.375$. Тільки початкове прискорення та соціальний фактор.

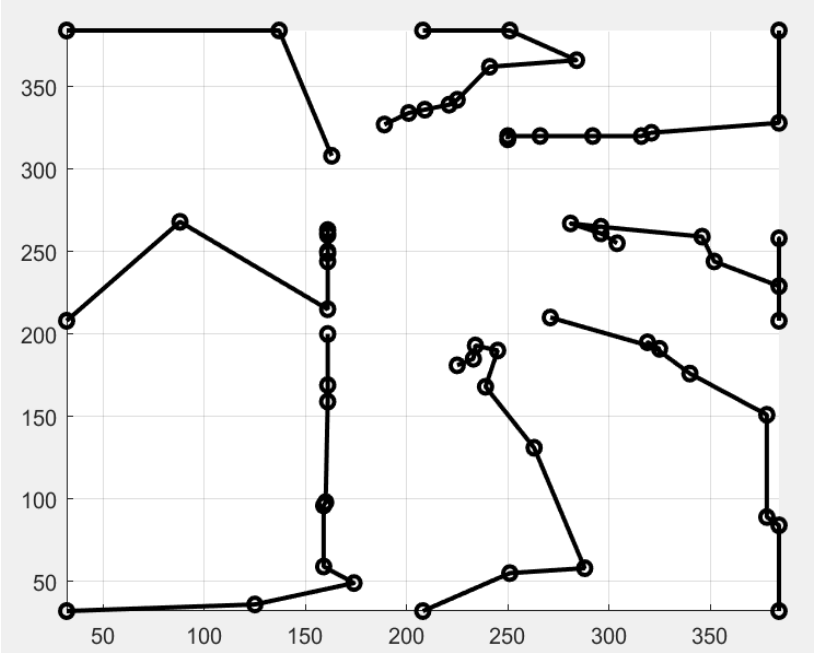

Рисунок 3.3 – Результати при w=0, $c_1 = 0.375$, $c_2 = 0.375$.

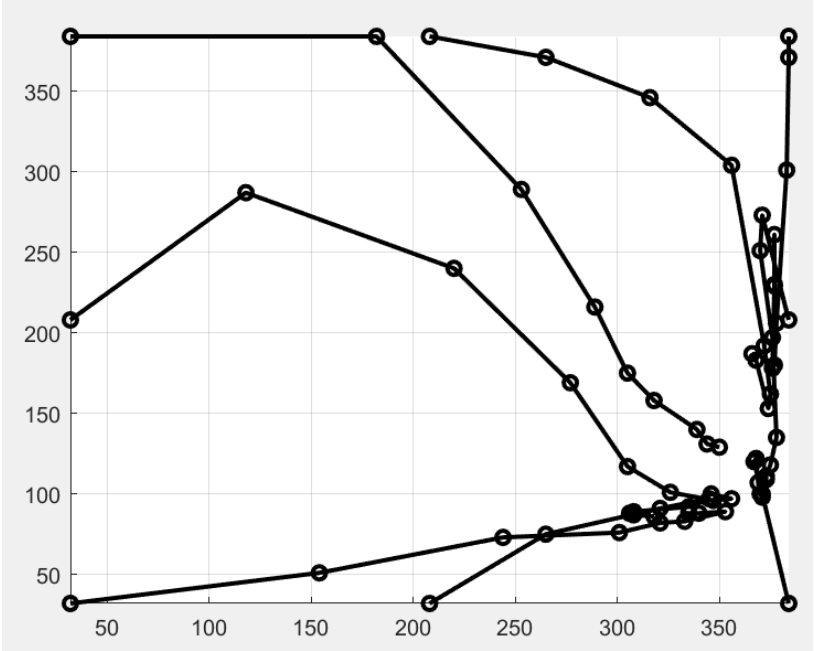

Рисунок 3.4 – Результати при w=0.3, c_1 = 0.5, c_2 = 0.49.

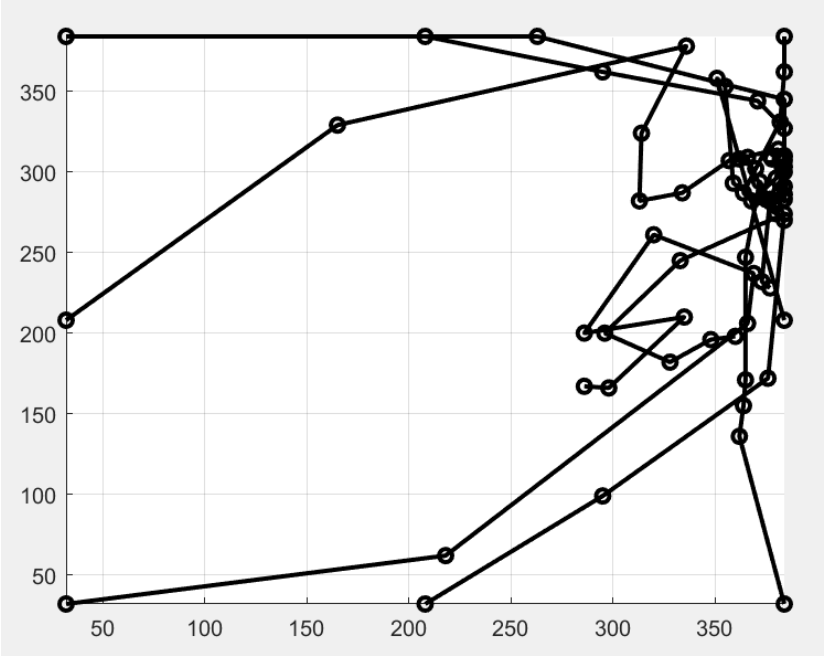

Рисунок 3.5 – Результати при w=0.5, c_1 = 0.75, c_2 = 0.75.

Як видно з наведених рисунків, коефіцієнти ДМРЧ мають великий вплив на характер поведінки часток. Загалом можна виділити два патерни поведінки часток – повільне дослідження навколо власного локального мінімуму або поступове або швидке наближення до першого локального мінімуму. Аналізуючи отримані характеристики, обрано такі коефіцієнти методу: w=0.5, c_1 = 0.75, c_2 = 0.75.

Варто зазначити, що обрані коефіцієнти мають велике значення, тому для інших тестових даних вибрані коефіцієнти можуть бути неоптимальними. Окремо варто відзначити випадкові коефіцієнти в формулі ДМРЧ, що мають значення від 0 до 1. Оскільки стартовий стан генератору випадкових чисел базується на системному часі ПК, ці коефіцієнти будуть різними при різних запусках програми навіть для тих самих даних. Щоби уникнути такої невизначеності початковий стан генератору випадкових чисел було явно задано.

3.4 Метод інтелектуального блочного розбиття

Запропонований метод інтелектуального блочного розбиття складається з декількох кроків. Нижче наведено послідовність кроків, що необхідно виконати при використанні запропонованого методу. Більш детально алгоритм роботи методу зображено на Рисунку 3.6.

1. Аналіз КП з метою визначення можливості використання методу розбиття на блоки;
2. Вибір одного або декількох обчислювальних циклів, що потребують оптимізації. Обмеження на склад обчислювальних циклів наведено нижче;
3. Вибір параметрів пошуку ДМРЧ – мінімальної та максимальної границь пошуку для розмірів блоків розбиття та вибір кількості часток;
4. Початкова ініціалізація методу ДМРЧ обраними параметрами;
5. Отримання від методу ДМРЧ розмірів блоків розбиття;
6. Застосування ДМРЧ для КП з розмірами блоків розбиття отриманих на минулому кроці;
7. Компіляція КП;
8. Виміри часу виконання скомпільованої КП;
9. Якщо число ітерацій перевищує задану кількість ітерацій або виконується критерій завершення оптимізації – завершення пошуку розмірів блоків розбиття. В іншому випадку – передача отриманих часів виконання в ДМРЧ і перехід на крок 5.

Як було зазначено, на КП накладаються деякі обмеження на обчислювальні цикли, а саме:

1. Обчислювальні цикли мають використовувати ключове слово for;
2. Всередині обчислювальних циклів не має бути викликів функцій або застосування макросів;
3. В середині циклів заборонено використання вказівників. Адресування даних має бути тільки через явні індекси масивів.

Алгоритм методу інтелектуального блочного розбиття, як зображено на Рисунку 3.6, використовує дискретний метод рою часток для пошуку кращих розмірів блоків розбиття шляхом ітеративного підбору блоків та оцінкою ефективності підібраних блоків шляхом компіляції КП та вимірювання часу виконання КП. Такий алгоритм достатньо гнучкий і дозволяє використовувати різноманітні варіанти методу розбиття на блоки як окремо, так і з розпаралелюванням одночасно.

В роботі перевірялась ефективність роботи методу на блоках розбиття прямокутної форми. Метод може бути також застосовано до методів, що використовують блоки розбиття іншої форми – трикутної, паралелограма, ромби тощо.

Завершення роботи алгоритму може досягатися двома шляхами – досягненням максимальної кількості ітерацій пошуку або виконанням критерія завершення пошуку. Критерій завершення пошуку в даному випадку – наперед заданий точний критерій або евристичний критерій, що може бути застосований оператором методу. Наприклад, деяка КП вимагає суттєвого прискорення виконання. З цією метою задана велика кількість часток методу і велика кількість ітерацій. Використовуючи метод, після невеликої кількості ітерацій досягнуто помітне прискорення КП. Припустимо, подальше використання методу на наступних ітераціях не призводить до пошуку кращих розмірів блоків. У такому випадку оператор може прийняти рішення про зупинення подальшого пошуку без досягнення максимальної кількості ітерацій.

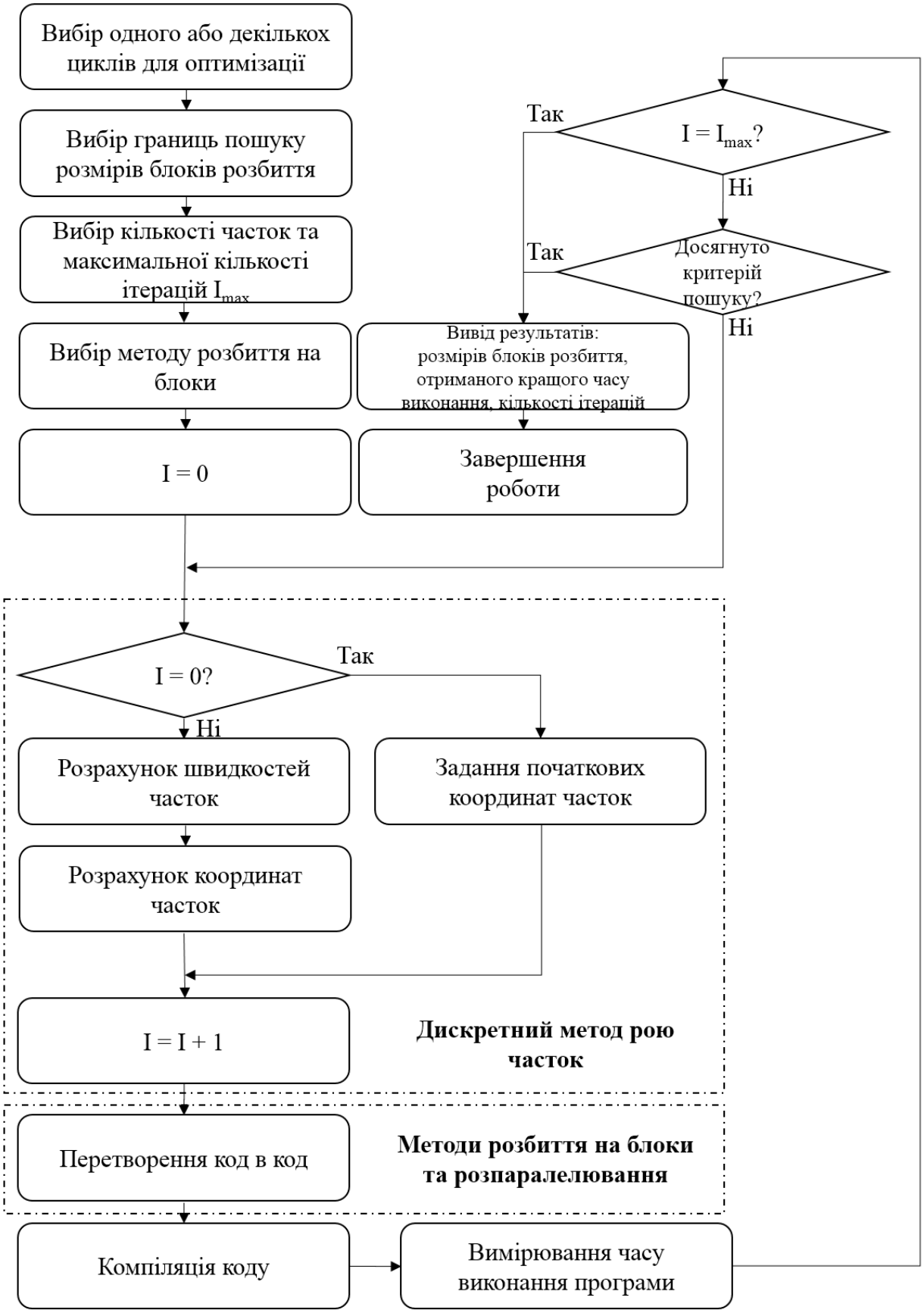

Рисунок 3.6 – Алгоритм методу інтелектуального блочного розбиття

Такий гнучкий критерій завершення пошуку необхідний тому, що складно визначити наперед точний критерій завершення. Наприклад, можна встановити критерієм завершення якесь конкретне значення прискорення, припустимо 10%. Проте якщо завдяки підбору кращих розмірів можна досягнути прискорення 20%, то таке прискорення не буде знайдено, тому що раніше спрацює критерій завершення пошуку.

Також обчислювальний цикл може не прискорюватись при будь-яких розмірах блоків при застосуванні ДМРЧ. В таких випадках також важливо мати можливість припинити подальший пошук достроково.

Реалізацію запропонованого методу інтелектуального блочного розбиття мовою програмування Matlab наведено у Додатку Ґ. В складі програмного пакету, що реалізує метод, виділено 4 функції: головна функція методу, функція початкового розміщення часток на границі допустимої області, функція дискретизації позицій часток та функція роботи ДМРЧ. Вказаний програмний пакет в інтерактивній формі запитує початкові параметри МІБР – кількість часток, ітерацій та допустимі границі розмірів блоків розбиття. Під час своєї роботи програма пропонує розміри блоків розбиття, які необхідно використати і для яких потрібно виміряти час виконання програми. Отриманий час виконання подається оператором в програму МІБР, яка на наступних ітераціях обчислить нові значення розмірів блоків розбиття.

Мова Matlab для реалізації МІБР була обрана завдяки простоті відлагодження та вбудованій можливості побудови графіків. Сам МІБР не прив’язаний до будь-якої мови програмування і може бути реалізований іншими мовами програмування високого рівня. Варто відзначити, що метод для своєї роботи не потребує значних обчислювальних потужностей, тому його програмна реалізація може бути використана на будь-якому сучасному ПК.

3.5 Висновки до Розділу 3

Розділ присвячено вирішенню задачі підбору кращого оптимізаційного методу для пошуку кращих розмірів блоків розбиття МРОЦНБ з метою зниження часу виконання КП. Запропоновано використання ДМРЧ для вирішення цієї задачі. Наведено експериментальні дані, що підтверджують доцільність використання методу для даної задачі.

Виконано підбір власних параметрів ДМРЧ з метою скорочення необхідних ітерацій для виконання пошуку.

На основі використання ДМРЧ для пошуку кращих розмірів блоків розбиття запропоновано метод інтелектуального блочного розбиття. Суть алгоритму зводиться до ітеративного підбору блоків розбиття для наперед заданої кількості часток та максимальної кількості ітерацій. Оцінка збільшення ефективності оптимізованих програм виконується шляхом вимірюванням часу виконання програм.

РОЗДІЛ 4. ВИКОРИСТАННЯ МЕТОДУ ІНТЕЛЕКТУАЛЬНОГО БЛОЧНОГО РОЗБИТТЯ

4.1 Вплив на енергоефективність обчислень

Аналізуючи збільшення енергоспоживання тестових програм, можна зробити висновок, що використання розпаралелювання призводить до збільшення енергоспоживання відносно однопотокового виконання тестових програм. Водночас, в такому випадку час виконання тестових програм значно зменшується. Для оцінки сумарної споживаної електричної енергії до та після застосування оптимізації, необхідно врахувати як зміну часу виконання, так і зміну енергоспоживання [27].

Для оцінки енергоефективності обчислень пропонується метод, що порівнює час виконання та енергоспоживання для початкової та для оптимізованої програми.

Сумарна електрична енергія, що необхідна для обчислення програми дорівнює інтегралу електричної потужності по часу виконання цієї програми:

$$E = \int_0^T p(t)dt. \tag{4.1}$$

Приймаючи до уваги постійне значення напруги живлення, значення електричної потужності в часі можна прийняти у вигляді:

$$p(t) = U \cdot i(t). \tag{4.2}$$

Враховуючи, що середнє значення сили струму може бути представлено згідно з вказаною формулою:

$$I_{mean} = \frac{1}{T}\int_0^T i(t)dt. \tag{4.3}$$

З наведених формул отримаємо загальну формулу розрахунку споживаної електричної енергії:

$$E = U \cdot I_{mean} \cdot T. \tag{4.4}$$

Для визначення кількісної оцінки підвищення енергетичної ефективності при використанні оптимізації введемо коефіцієнт енергетичної ефективності обчислень, як відношення початкової електричної енергії, що необхідна для розрахунку до електричної енергії, що необхідна до розрахунку оптимізованої версії:

$$k_E = \frac{E_{original}}{E_{optimized}} = \frac{U \cdot I_{mean\_original} \cdot T_{original}}{U \cdot I_{mean\_optimized} \cdot T_{optimized}} = \left(\frac{T_{original}}{T_{optimized}}\right) \cdot \left(\frac{I_{mean_{original}}}{I_{mean_{optimized}}}\right). \quad (4.5)$$

Використовуючи результати часу виконання тестових програм та їх енергоспоживання (Таблиці 2.1-2.4), використаємо формулу (4.5) для оцінки коефіцієнту енергоефективності обчислень для плати Raspberry Pi 3 та для ПК на базі Intel® Core™ i5-4670K. Отримані результати приведено в Таблиці 4.1 та 4.2 відповідно.

Як випливає з отриманих результатів, у випадку, коли методи розбиття на блоки пришвидшують виконання ТП, одночасно підвищується енергоефективність обчислень. У випадках, коли використовується розпаралелювання і споживана електрична потужність зростає, все одно енергоефективність обчислень також підвищується. Загалом, використання розпаралелювання з точки зору енергоефективності обчислень є більш ефективним ніж без використання розпаралелювання.

Запропоновану оцінку енергетичної ефективності обчислень доцільно використовувати для оцінки програм, що розпаралелені, з огляду на те, що енергоспоживання розпаралеленої програми може збільшуватися через використання усіх ядер мікропроцесора, а час роботи програми суттєво зменшуватися. Коефіцієнт енергетичної ефективності обчислень є відносною інтегральною оцінкою часу та енергоспоживання, що дозоляє порівнювати повну споживану електричну енергію, необхідну для розрахунків при використанні різних методів оптимізації та їх параметрів.

Таблиця 4.1 – Коефіцієнт енергоефективності обчислень для тестових програм для платі Raspberry Pi 3 при використанні різних методів розбиття на блоки та розпаралелюванні.

Тестова програма	**Опції оптимізації програми Pluto**							
	Tile	**Tile Parallel**	**Tile L2Tile Parallel**	**Innerpar**	**Innerpar Tile Parallel**	**Tile Multipar Parallel**	**Diamond-tile**	**Diamond-tile Parallel**
correlation	1,78	2,97	1,59	2,00	3,16	2,29	2,04	3,18
covariance	1,88	3,27	1,51	1,87	3,26	2,25	1,81	3,35
gemm	1,14	1,56	1,57	1,02	1,56	1,53	1,06	1,63
gemver	3,41	4,56	4,88	3,60	4,69	3,72	2,68	4,75
gesummv	1,03	1,57	1,36	1,17	1,55	1,12	1,00	1,74
symm	1,01	0,99	1,06	1,02	1,05	1,01	1,04	0,96
syr2k	2,11	3,85	1,09	1,82	3,51	2,51	1,95	3,83
syrk	1,26	1,89	0,67	1,27	1,84	1,13	1,17	1,77
trmm	1,31	3,26	0,91	1,43	3,28	2,06	1,30	3,93
2mm	1,12	1,89	0,70	0,88	1,86	1,43	1,10	1,89
3mm	1,18	1,83	0,66	0,99	1,87	1,46	1,17	1,87
atax	0,62	0,94	0,98	0,93	0,95	0,85	0,62	0,98
bicg	0,64	0,98	0,98	1,00	0,98	0,82	0,63	1,01
doitgen	1,15	1,18	0,51	1,30	1,18	1,14	1,07	1,19
mvt	1,32	2,75	2,77	0,84	2,75	1,87	1,26	2,81
cholesky	0,87	1,60	0,63	0,95	1,91	1,29	1,04	1,71
durbin	0,99	0,97	1,02	1,06	0,97	0,99	1,01	0,99
gramschmidt	1,32	1,90	0,74	1,00	1,86	1,54	1,01	1,89
lu	1,95	3,82	1,57	1,95	4,01	2,76	1,64	3,35
ludcmp	1,11	1,09	1,04	1,04	1,15	1,02	1,04	1,04
trisolv	0,86	0,81	1,26	1,00	1,53	0,72	0,86	0,83
deriche	0,98	1,04	0,95	1,01	0,99	0,93	0,98	1,01
floyd-warshall	0,66	0,68	0,29	1,00	0,78	0,56	0,67	0,68
nussinov	1,02	2,22	0,79	1,00	1,87	1,59	1,00	2,09
fdtd-2d	1,28	2,00	0,93	1,15	2,79	1,76	1,00	1,00
heat-3d	0,99	0,95	1,00	1,06	1,62	1,09	1,00	1,00
jacobi-2d	1,14	1,61	1,00	1,01	2,24	1,51	1,00	1,00
seidel-2d	1,04	1,92	1,00	0,99	2,38	1,44	1,06	1,81

Таблиця 4.2 – Коефіцієнт енергоефективності обчислень для тестових програм для ПК на базі Intel® Core™ i5-4670K при використанні різних методів розбиття на блоки та розпаралелюванні.

Тестова програма	**Опції оптимізації програми Pluto**								
	Tile	**Tile Parallel**	**Tile L2Tile**	**Innerpar**	**innerpar**	**Innerpar**	**Tile**	**Diamond-**	**Diamond-**
correlation	4,86	8,07	5,01	4,80	10,71	15,73	7,13	5,10	15,03
covariance	4,81	7,54	5,00	4,99	14,48	16,06	7,52	5,09	15,50
gemm	2,35	6,28	18,93	1,45	5,89	14,76	5,79	2,25	14,57
gemver	5,12	12,43	21,14	16,04	31,95	20,90	10,08	5,12	18,61
gesummv	0,88	2,16	3,10	1,77	6,51	3,91	1,84	0,90	3,62
symm	1,04	1,02	1,06	1,01	1,03	1,05	1,05	1,06	1,05
syr2k	1,41	3,68	1,81	1,44	8,39	7,46	2,97	1,32	7,43
syrk	1,00	2,12	1,33	0,96	2,91	2,87	1,56	0,99	3,83
trmm	3,40	11,10	4,61	5,47	1,70	27,26	10,97	3,49	25,93
2mm	1,91	3,92	2,37	1,64	9,24	7,90	4,34	1,97	7,94
3mm	1,59	3,28	2,01	1,34	7,41	6,78	3,70	1,68	6,79
atax	0,42	1,04	1,56	1,00	1,89	1,50	0,72	0,36	1,52
bicg	0,82	2,08	3,28	2,26	3,93	3,27	1,61	0,76	2,57
doitgen	4,73	2,84	4,42	6,42	0,30	5,65	6,20	5,39	5,50
mvt	3,67	8,53	17,28	1,16	4,64	15,23	8,19	3,69	13,35
cholesky	1,24	2,20	1,48	1,01	2,33	4,19	1,82	1,28	4,07
durbin	1,00	1,00	1,01	0,99	1,00	1,00	1,00	1,00	1,00
gramschmidt	1,79	2,41	1,74	1,04	3,80	4,07	2,84	1,75	3,97
lu	1,46	2,74	2,25	1,60	1,90	6,35	2,90	1,53	5,18
ludcmp	1,01	1,00	1,00	0,99	1,00	1,00	0,97	0,96	0,99
trisolv	0,73	1,43	1,93	1,22	1,96	2,47	1,09	0,70	1,64
deriche	1,28	1,57	2,00	1,00	2,43	2,83	1,32	1,19	1,75
floyd-warshall	0,81	1,21	1,33	0,99	0,54	2,22	1,27	0,84	2,18
nussinov	1,04	2,30	1,52	1,02	2,86	3,49	2,17	1,07	4,15
fdtd-2d	0,93	1,18	0,46	0,92	4,22	3,02	1,46	1,00	1,00
heat-3d	2,02	1,64	1,00	2,62	11,79	4,37	2,51	1,00	1,00
jacobi-2d	0,97	1,35	1,00	0,95	5,05	2,99	1,61	1,00	1,00
seidel-2d	1,19	2,22	1,00	1,02	4,75	4,65	2,14	1,00	1,00

4.2 Результати застосування підібраних коефіцієнтів для метода інтелектуального блочного розбиття

Для визначення ефективності обраних коефіцієнтів ДМРЧ було проведено ті самі тести для різної кількості часток та кількості ітерацій для тієї ж ТП. Отримані дані зведено в таблицю 4.3.

Таблиця 4.3 – Результати пошуку мінімального часу обчислення при різній кількості ітерацій і кількості часток ДМРЧ для нових коефіцієнтів методу.

Кількість часток	**Кількість ітерацій**	**Мінімум часу виконання програм, с**	**Кількість часток**	**Кількість ітерацій**	**Мінімум часу виконання програм, с**
4	4	0.082245	32	4	0.079200
	8	0.080601		8	0.077217
	16	0.079942		16	0.077086
	32	0.077362		32	0.077086
	64	**0.076950**		64	0.077086
	128	0.076950		128	0.077086
	256	0.076950		256	0.077086
	512	0.076950		512	0.077086
	1024	0.076950		1024	0.077086
8	**4**	**0.076955**	64	4	0.078654
	8	0.076955		8	0.078543
	16	0.076955		16	0.078543
	32	0.076955		32	0.077086
	64	0.076955		64	0.077086
	128	0.076955		128	0.077086
	256	0.076955		256	0.077086
	512	0.076955		512	0.077086
	1024	0.076955		1024	0.077086
16	4	0.079447	128	4	0.077202
	8	0.078953		8	0.076744
	16	0.077086		16	0.076744
	32	0.077086		32	0.076744
	64	0.077086		64	0.076744
	128	0.077086		128	0.076744
	256	0.077086		256	0.076744
	512	0.077086		512	0.076744
	1024	0.077086		1024	0.076744

Порівнюючи мінімум часу виконання програм, отриманий при підібраних коефіцієнтах ДМРЧ з мінімумом часу виконання програм, що був отриманий без підбору параметрів і був представлений в Таблиці 3.1, можна стверджувати, що

пошук кращих розмірів блоків розбиття став виконуватись набагато швидше. А саме: в даному випадку мінімум часу виконання був знайдений за 64 ітерації для 4 часток або 4 ітерації для 8 часток. Раніше, щоб отримати такі дані потребувалося виконати 64 ітерації для 32 часток або 256 ітерацій для 4 часток.

Таким чином, можна стверджувати, що використання підібраних параметрів ДМРЧ дозволило підвищити ефективність методу розбиття на блоки за меншу кількість ітерацій і часток пошуку. В практичному сенсі це означає менший час роботи методу інтелектуального блочного розбиття для досягнення тих самих результатів оптимізації.

4.3 Верифікація методу інтелектуального блочного розбиття

З метою перевірки ефективності роботи методу інтелектуального блочного розбиття його було використано на низці тестових програм, що виконувались в експериментах 1-4. Критерієм ефективності роботи методу є час виконання тестових програм.

Метод інтелектуального блочного розбиття застосовувався до двох методів із пакету Pluto – класичний метод розбиття на блоки (tile) та метод розбиття на блоки з виявленням внутрішнього паралелізму і розпаралелюванням (tile, innerpar, parallel).

Отримані дані нормалізовано до початкового часу виконання тестових програм і наведено в таблицях 4.4 та 4.5.

З метою кращого візуального аналізу отриманих результатів, їх також наведено у вигляді діаграм на Рисунках 4.1 та 4.2.

Таблиця 4.4 – Відносний час виконання тестових програм при використанні методу tile та методу інтелектуального блочного розбиття.

Тестова програма	**Відносний час виконання початкової КП, %**	**Відносний час виконання КП з використанням методу tile, %**	**Відносний час виконання КП з використанням методу інтелектуального блочного розбиття, %**
covariance	100,00	20,05	17,92
gesummv	100,00	98,82	86,23
syr2k	100,00	71,01	62,77
2mm	100,00	51,47	51,00
3mm	100,00	61,19	61,08
atax	100,00	252,50	209,52
bicg	100,00	121,35	90,30
mvt	100,00	26,48	23,18
cholesky	100,00	79,47	76,57
gramschmidt	100,00	56,54	41,00
lu	100,00	67,08	64,21

Таблиця 4.5 – Відносний час виконання тестових програм при використанні методів tile, innerpar, parallel та методу інтелектуального блочного розбиття.

Тестова програма	**Відносний час виконання початкової КП, %**	**Відносний час виконання КП з використанням методів tile, innerpar, parallel, %**	**Відносний час виконання КП з використанням методу інтелектуального блочного розбиття, %**
covariance	100,00	9,20	6,08
gesummv	100,00	33,54	30,14
syr2k	100,00	19,14	17,24
2mm	100,00	18,00	12,94
3mm	100,00	21,04	18,02
atax	100,00	80,53	66,43
bicg	100,00	39,00	29,27
mvt	100,00	8,71	7,53
cholesky	100,00	32,32	18,25
gramschmidt	100,00	31,95	23,38
lu	100,00	22,45	15,55

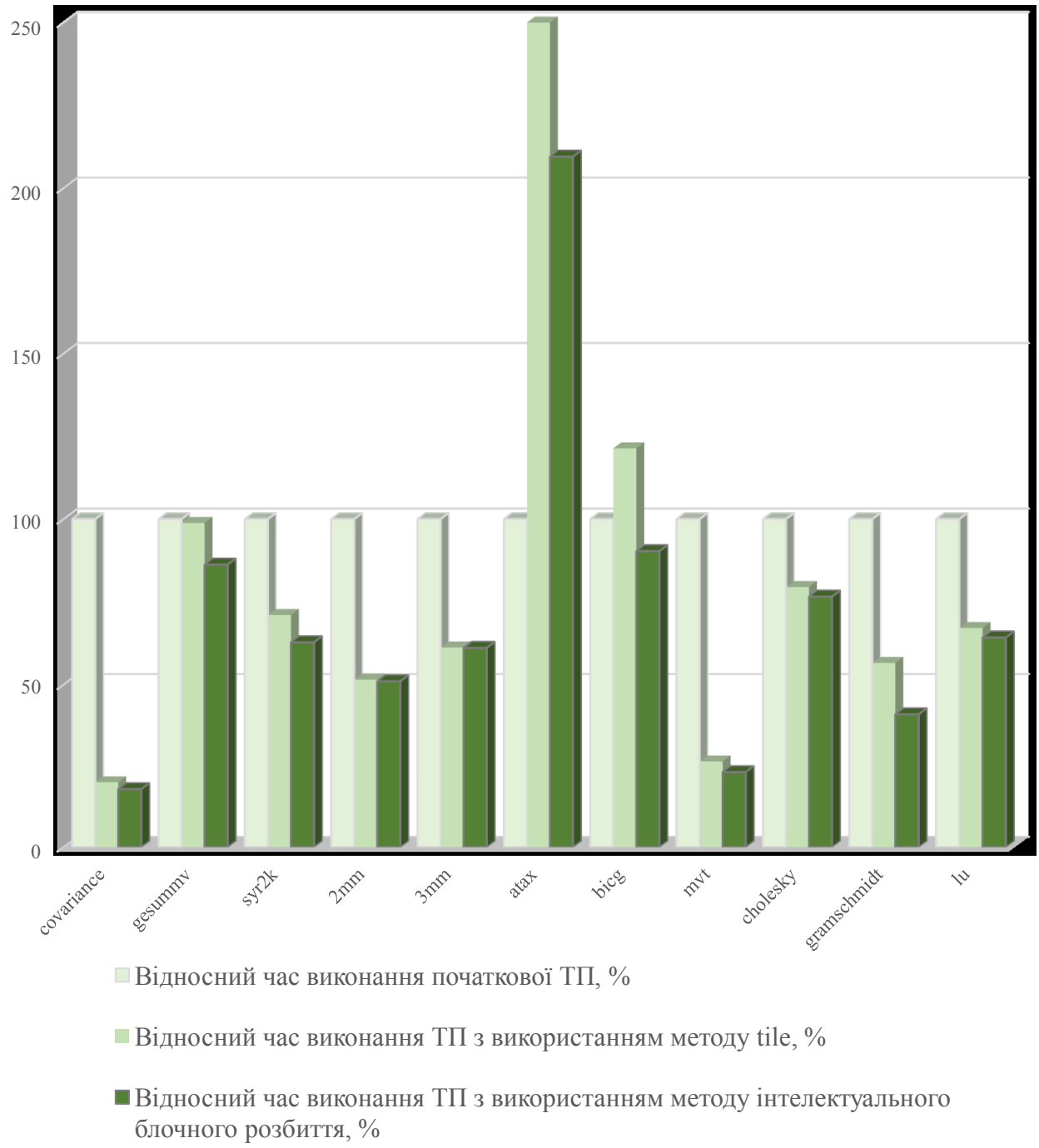

Рисунок 4.1 – Відносний час виконання тестових програм при використанні методу tile та методу інтелектуального блочного розбиття.

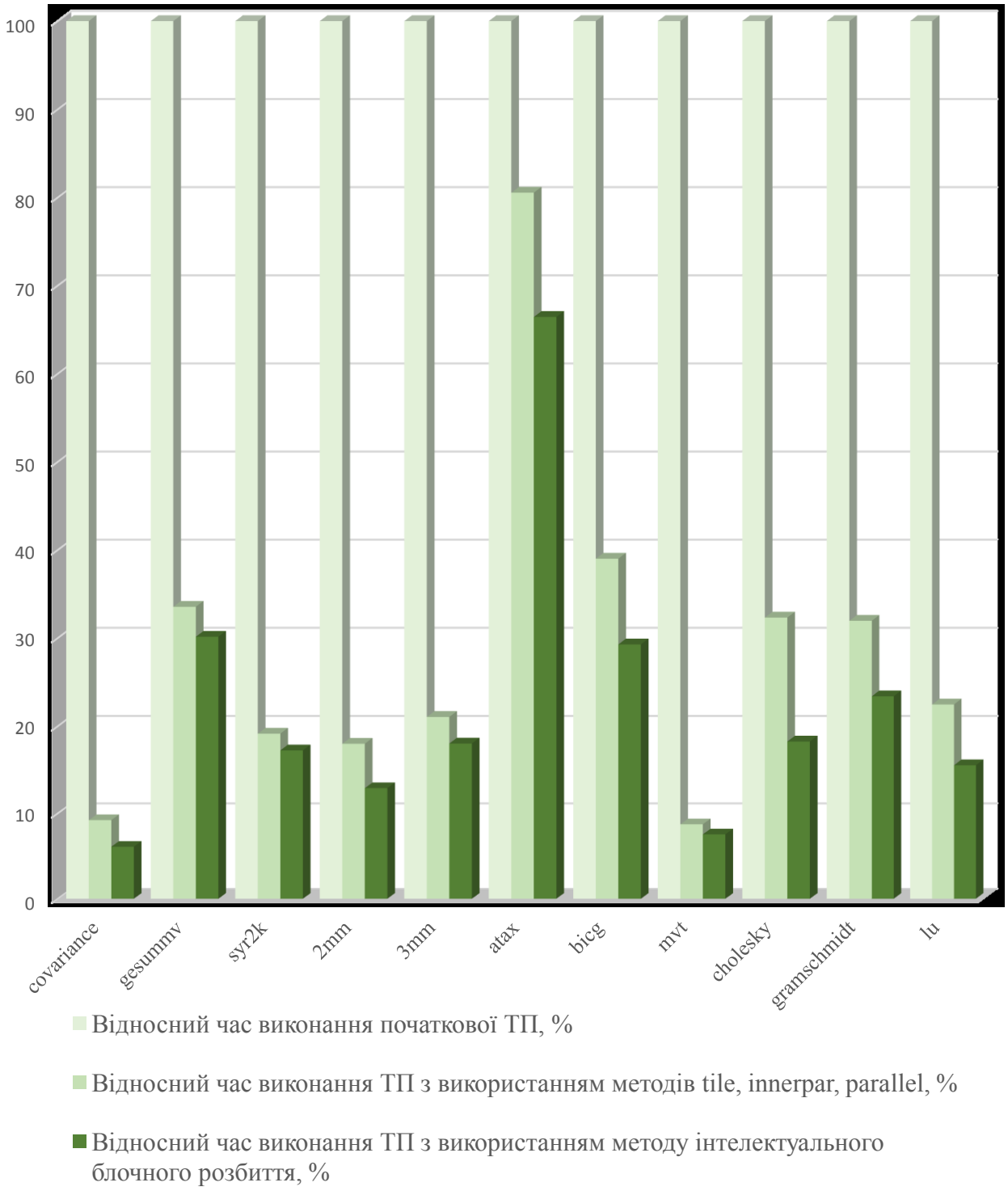

Рисунок 4.2 – Відносний час виконання тестових програм при використанні методів tile, innerpar, parallel та методу інтелектуального блочного розбиття

Як видно з наведених даних, метод інтелектуального блочного розбиття підвищує ефективність того методу, який використовує. Величина підвищення ефективності відрізняється в залежності від ТП. У випадках, коли використання методів розбиття на блоки призводило до сповільнення ТП, метод інтелектуального блочного розбиття також його прискорював. Для деяких випадків час виконання відносно початкового часу виконання ТП був менший, у деяких більший.

Результати роботи для методів розбиття на блоки без розпаралелювання та з розпаралелюванням дещо відрізняються. Загалом, додаткове прискорення виконання ТП методи розбиття на блоки без методів розпаралелювання досягає 1-18% та 4-33% використовуючи методи розбиття на блоки разом з розпаралелюванням.

4.4 Висновки до Розділу 4

В розділі запропоновано та виконано оцінювання доцільності оптимізації коду програм, що базується на часі виконання програм та енергоспоживанні. Вводиться коефіцієнт енергетичної ефективності обчислень, що дозволяє порівнювати загальну електричну енергію, що необхідна для обчислень для різних методів оптимізації та їх параметрів відносно одне іншого або відносно неоптимізованою програми. Застосування оцінювання енергетичної ефективності обчислень в першу чергу може бути корисним для мобільних обчислювальних систем, де сумарна кількість наявної електричної енергії є обмеженою.

В розділі наводяться експериментальні дані щодо використання запропонованого методу інтелектуального блочного розбиття та висновки щодо його ефективності.

Узагальнюючи отримані дані варто відзначити такі особливості методу інтелектуального блочного розбиття:

1. Використання методу інтелектуального блочного розбиття дає додаткове прискорення методам розбиття на блоки;

2. Використання методу інтелектуального блочного розбиття більш ефективне для методів розбиття на блоки, що використовуються одночасно з розпаралелюванням;

3. ТП, що мали сповільнення відносно початкового часу виконання, також дещо прискорюються;

4. Використання методу інтелектуального блочного розбиття потребує багато часу, оскільки метод є ітеративним і на кожній ітерації потрібно робити вимір часу виконання програми. Така особливість методу суттєво впливає на його застосування. Важливо ретельно обирати обчислювальні цикли для оптимізації, на які припадає найбільше часу виконання. Визначення таких циклів може бути виконано шляхом профілювання КП.

Зважаючи на вищезазначене можна стверджувати, що використання методу інтелектуального блочного розбиття виправдано як для методів розбиття на блоки окремо, так і для методів розбиття на блоки разом з використанням розпаралелювання.

ВИСНОВКИ

У дисертаційні роботі отримано нові результати у розв’язанні актуальної науково-практичної задачі з удосконалення методів оптимізації ПЗ. В роботі запропоновано метод інтелектуального блочного розбиття, який використовує дискретний метод рою часток з метою підбору найкращих розмірів блоків розбиття в оптимізаційному методі розбиття на блоки для прискорення виконання комп’ютерних програм. Ефективність застосування запропонованого методу доводиться отриманими експериментальними даними, які засвідчують прискорення швидкодії відносно класичного методу розбиття на блоки.

За результатами виконаної дисертаційної роботи можна сформулювати наступні висновки:

– Вперше запропоновано ітераційний процес розпаралелювання операторів циклів мікропроцесорних програм, який відрізняється від відомих методів визначенням оптимального рішення щодо розбиття ітераційного простору оператора циклу на окремі блоки.

– Вперше запропоновано оцінку доцільності оптимізації коду програм і отримано експериментальні дані, що засвідчують покращення ефективності обчислень (за часом або енергоспоживанням), в більшості випадків при застосуванні методу розбиття на блоки.

– Розроблено метод інтелектуального блочного розбиття, що виконує пошук оптимальних розмірів прямокутних блоків розбиття ітераційного простору операторів циклу мікропроцесорних програм на основі дискретного методу рою часток, який може бути застосовано до довільних обчислювальних циклів написаних мовами програмування С або С++.

– Вдосконалено метод оцінки часу виконання тестових програм при використанні різноманітних методів розбиття на блоки, який базується на автоматичному послідовному багаторазовому запуску тестових програм і фіксації отриманих результатів швидкодії, що можуть бути в подальшому проаналізовані.

– Вдосконалено дискретний метод рою часток шляхом визначення

коефіцієнтів методу, а саме: початкового коефіцієнту інерції, індивідуального та соціального коефіцієнтів, при яких пошук розмірів блоків розбиття в задачі прискорення швидкодії виконується швидше класичного методу.

– Вдосконалено використання дискретного методу рою часток шляхом визначення кращих початкових даних для розташування часток рою, що зменшує число ітерацій для знаходження оптимального рішення.

В роботі наводяться дані вимірювань, що свідчать не тільки про прискорення виконання більшості тестових програм при використанні методів розбиття на блоки, але й про підвищення енергоефективності обчислень. Цей факт свідчить про доцільність використання методу в «зелених» обчисленнях.

Отримані у рамках дисертаційної роботи результати підтверджують ефективність запропонованого методу та його програмної реалізації. Запропонований метод та його програмна реалізація можуть бути використані для оптимізації комп’ютерних програм написаних мовою програмування C або C++ будь-якого призначення, апаратної платформи та компілятора.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Abdallah F.B. Fast evaluation of power consumption of embedded systems using DIPLODOCUS / F.B. Abdallah and L. Apvrille // 39th Euromicro Conference on Software Engineering and Advanced Applications. – 2013. – P. 138–144. doi:10.1109/SEAA.2013.8

2. Abram H. Green software engineering: the curse of methodology / H. Abram // Proceedings: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), P. 46–55, 14–18 Mar 2016. doi:10.1109/SANER.2016.60

3. Aho A. Compilers, principles, techniques, and tools / A. Aho, M. Lam, R. Sethi and J. Ullman // Pearson Education, Inc., 2007.

4. Amini M. PIPS Is not (just) Polyhedral Software / M. Amini and et al // International Workshop on Polyhedral Compilation Techniques (IMPACT'11). – Chamonix, France. – April, 2011.

5. Banerjee P. The PARADIGM Compiler for Distributed-Memory Multicomputers / P. Banerjee et al. // IEEE Computer. – 1995. – Vol. 10, №28. – P. 37–47.

6. Banerjee U. Time and Parallel Processor Bounds for Fortran-like Loops / U. Banerjee et al. // IEEE Trans. on Computers. – 1979. – № 9. – P. 660-670.

7. Bastoul C. Code Generation in the Polyhedral Model Is Easier Than You Think / C. Bastou // Proceedings of the PACT'13 IEEE International Conference on Parallel Architecture and Compilation Techniques. – 2004. – P. 7-16.

8. Beletska A. Extracting Coarse-Grained Parallelism with the Affine Transformation Framework and its Limitations / A. Beletska, P. San Pietro // Electronic Modelling. – 2006. – №5.

9. Beni G. Swarm Intelligence in Cellular Robotic Systems / G. Beni, J. Wang // Proceed. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30, 1989.

10. Bielecki W. Extracting coarse-grained parallelism in computer simulation applications / W. Bielecki, M. Palkowski // ACS 2006, 13th International Multi-Conference. – Szczecin. – 2006. – Vol. II.

11. Bielecki W. Finding Synchronization-Free Parallelism for Non-uniform Loops / W. Bielecki // In Proceedings of the Computational Science – ICCS 2003, Lecture Notes in Computer Science. – Springer. – 2003. – Vol. 2658. – P. 925-934.

12. Bielecki W. Insight into tiles generated by means of a correction technique / W. Bielecki and P. Skotnicki // The Journal of Supercomputing, 2018.

13. Bielecki W. Synchronization-free automatic parallelization: Beyond affine iteration-space slicing / W. Bielecki et al. // LCPC 2009. Lecture Notes in Computer Science. – Berlin: Springer. – October, 2009. – Vol. 5898.

14. Bielecki W. Kompilator jezyka VHDL do syntezy ukladow logicznych. / W. Bielecki, S. Hyduke // Reprogramowalne uklady cyfrowe (RUC'99), – Szczecin. – 1999. – P. 183-190.

15. Bielecki W. Generating Boolean equations for synchronous-asynchronous logic described in VHDL language / W. Bielecki, P. Jaworski // Proceedings of Advanced computer systems (ACS'2000). – 2000. – Szczecin. – P. 397-400.

16. Bielecki W. Extracting synchronization-free threads in perfectly nested loops using the Omega project software / W. Bielecki, K. Siedlecki // SEPADS'05 Proceedings of the 4th WSEAS Internatioмnal Conference on Software Engineering, Parallel & Distributed Systems Salzburg, Austria. – February 13 – 15, 2005.

17. Blume W. Parallel programming with Polaris / W. Blume, R. Doallo, R. Eigenmann // Computer. – 1992. – Vol. 29, № 12. – P. 78–82.

18. Bondhugula U. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model / Uday Bondhugula et al. // International Conference on Compiler Construction (ETAPS CC). – Budapest, Hungary. – 2008.

19. Bondhugula U. A Practical Automatic Polyhedral Parallelizer and Locality Optimizer / Bondhugula Uday et al. // ACM SIGPLAN Programming Languages Design and Implementation (PLDI). – Tucson, Arizona. – June, 2008.

20. Bondhugula U. Effective Automatic Parallelization and Locality Optimization Using The Polyhedral model / Bondhugula Uday // The Ohio State University. – 2010. – P. 193.

21. Bondhugula U. Pluto: A practical and fully automatic polyhedral parallelizer and locality optimizer / Uday Boundhugula, J. Ramanujam, P. Sadayappan // Technical Report OSU-CISRC-10/07-TR70. – Columbus. – Lомuisiana State University. – 2007.

22. Boulet P. Loop parallelization algorithms: from parallelism extraction to code generation / P. Boulet et al. // Parallel Computing. – 1998. – Vol. 24, №3. – P. 421-444.

23. Brelsford K. Energy efficient computation: A silicon perspective / Kevin Brelsford, Serafin A. Perez Lopez, Santiago Fernandez-Gomez // Integration the VLSI Journal 47(1):1–11, 2014.

24. Carrol A. An analysis of power consumption in a smartphone / A. Carrol, G. Heiser // In Proceedings of the 2010 USENIX conference on USENIX annual technical conference (USENIXATC'10). – Berkeley, CA, USA: USENIX Association. – 2010.

25. Chandrakasan A. P. Low power CMOS digital design / A.P. Chandrakasan, S. Sheng, R.W. Brodersen // IEEE Journal of Solid State Circuits. – 1992. – Vol. 27. – P. 473-484.

26. Chemeris A.A. Software for Automatic Parallel Implementation of Sequential C Programs on Transputer Systems / V.N. Beletskii, A.A. Bagaterenko, A.A. Chemeris // Engineering Simulation. – 1995. – Vol. 12. – P. 174-189.

27. Chemeris A. Influence of Software Optimization on Energy Consumption of Embedded Systems / A. Chemeris, D. Lazorenko, S. Sushko // In book: V. Kharchenko et al. (eds.), Green IT Engineering: Components, Networks and Systems Implementation, Studies in Systems, Decision and Control 105, DOI 10.1007/978-3-319-55595-9_6, Springer International Publishing AG. – 2017. – P. 111-133.

28. Chemeris A. Organization of a parallel virtual machine / A. Chemeris, V. Bielecky, T. Popova // International Symposium on Parallel Architectures, Algorithms and Networks (Kanazawa , Japan, 14-16 Dec 1994). – 1994. – P. 421-426.

29. Chemeris A. The Creation of Program Graph for Automatic Parallelization / A. Chemeris // Applications of Computer Systems. (ACS'95) – Szciecin. – 1995. – P. 47-52.

30. Chemeris A. An Automatic Parallelization for Distributed Systems / A. Chemeris, V. Bielecky, A. Bagaterenco // Applications of Computer Systems. (ACS'95) – Szciecin. – 1995. – P. 53-60.

31. Chemeris A.A. Package for Automatic Parallelization of Serial C-Programs for Distributed Systems / A. Chemeris, V. Bielecky, A. Bagaterenco // IEEE Conference "Programming Models for Massively Parallel Computers", 9-12 Oct. 1995, Berlin, DOI: 10.1109/PMMPC.1995.504357. – 1995. – P. 184-188.

32. Chemeris A. Practical realization of asynchronous iterative algorithms for multiprocessor computers / A. Chemeris // ACS'99 : Advanced computer systems. – Szczecin. – 1999. – P. 228-232.

33. Chemeris A. Low-power issues for SoC / A. Chemeris, D. Lazorenko // In Proceedings: IEEE Tenth International Symposium on Consumer Electronics, ISCE '06. DOI: 10.1109/ISCE.2006.1689463. – 2006. – P. 573-575.

34. Chemeris A. Parallel computer emulator for digital devices modeling / A. Chemeris, S. Reznikova // In proc.: IEEE EWDTS'08, Lviv, October 9-12, – 2008. – P. 383-386.

35. Chemeris A. Loop Nests Parallelization for Digital System Synthesis / A. Chemeris, J. Gorunova, D. Lazorenko // Proceedings of IEEE East-West Design & Test Symposium (EWDTS'2012), Kharkov, Ukraine, September 14–17. – 2012. – P. 118-121.

36. Chemeris A. Asynchronous Analogs of Iterative Methods and Their Efficiency for Parallel Computers / A. Chemeris, M. Savchenko // In Collection of scientific papers "Parallel and Distributed Computing Systems" (PDCS 2013). – Kharkiv. – 2013. – P. 276-280.

37. Chemeris A. Data-flow multiprocessor with deterministic architecture / A. Chemeris, A. Novatsky, J. Glushko, O. Pugachov // Proceedings of IEEE East-West

Design & Test Symposium (EWDTS'2013), Rostov-on-Don, Russia, September 27 – 30. – 2013. – P. 138-141.

38. Chemeris A. The Dependence of Microprocessor System Energy Consumption on Software Optimization / S. Sushko, A. Chemeris // 2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO), April 18-20, 2017, ISBN: 978-1-5386-1700-7. – Kyiv. – P. 451-454.

39. Christian P. Low-Power Processors and Systems on Chips, / P. Christian // CRC Press, Boca Raton (2005)

40. Clauss Phillippe Parametric Analysis of Polyhedral Iteration Spaces / Phillippe Clauss, Vincent Loechner // Journal of VLSI Signal Processing, – 1998. – №19 (2). – P. 1-16.

41. Clerc M. Particle Swarm Optimization / M. Clerc // London, UK: ISTE, 2006.

42. Danckaert K. Platform independent data transfer and storage exploration illustrated on a parallel cavity detection algorithm / K. Danckaert, F. Catthoor, H. Man // Proc. PDPTA (CSREA Conf. on Parallel and Distributed Processing Techniques and Applications). – Las Vegas, Nevada. – 1999.

43. Darte A. Affine-by-Statement Scheduling of Uniform Loop Nests over Parametric Domains / A. Darte, Y. Robert // Journal of Parallel and Distributed Computing. – 1995. – Vol. 29, №1. – P. 43-59.

44. Darte A. Loop Parallelization Algorithms / A. Darte, Y. Robert, F. Vivien // LNCS on Compiler Optimizations for Scalable Parallel Systems: Languages, Compilation Techniques, and Run Time Systems / – Springer Verlag. – 2001. – Vol. 1808.

45. Darte Alain Optimal Fine and Medium Grain Parallelism Detection in Polyhedral Reduced Dependence Graphs / Alain Darte, Frederic Vivien // International Journal of Parallel Programming. – 1997. – Vol. 25, №6, – P. 447-496.

46. Darte A. Combining retiming and scheduling techniques for loop parallelization and loop tiling. / A. Darte, G. Silber, F. Vivien. // Parallel Processing Letters. – 1997. – №7. – P. 379–392.

47. Ellis C. Controlling energy demands in mobile computing systems / C. Ellis // San Rafael : Morgan & Claypool. – 2007.

48. Engelbrecht A. Fundamentals of Computational Swarm Intelligence / Engelbrecht A. P. // Chichester, UK, John Wiley & Sons, 2005.

49. Feautrier P. Parametric integer programming / P. Feautrier // RAIRO Recherche Op´erationnelle. – 1988. – Vol. 22, №3. – P. 243–268.

50. Feautrier P. Some efficient solutions to the affine scheduling problem, part i, one dimensional time. / P. Feautrier // International Journal of Parallel Programming. – 1992. – Vol. 21. – P. 313-348.

51. Feautrier P. Some efficient solutions to the affine scheduling problem, part ii, multidimensional time / P. Feautrier // International Journal of Parallel Programming . – 1992. – Vol. 21. – P. 389- 420.

52. Feautrier Paul The Polyhedron Model / Paul Feautrier, Christian Lengauer // In book: Encyclopedia of Parallel Computing. – Springer-Verlag. – 2011.

53. Foster Ian Designing and Building Parallel Programs [Електронний ресурс]. – 2016. – Режим доступу: http://www.mcs.anl.gov/~itf/dbpp/.

54. Fraboulet A. Loop Alignment for Memory Accesses Optimization / A. Fraboulet, G. Huard, A. Mignotte // Twelfth International Symposium on System Synthesis. – Piscataway, New Jersey. – 1999.

55. Glökler T. Design of energy-efficient application specific instruction set processors / T. Glökler // Dordrecht : Kluwer academic publishers. – 2004.

56. Grosser T. Polly – Performing polyhedral optimizations on a low-level intermediate representation / Tobias Grosser, Armin Groesslinger, Christian Lengauer // Parallel Processing Letters. – 2012.

57. Gostelow K. P. A view of data flow / K. P. Gostelow, R. E. Thomas // Proceedings of the 1989 NCC, Montvale, NJ, AFIPS. – Montvale, NJ. – 1989. – P. 629-636.

58. Gunter F. Managing the Impact of Increasing Microprocessor Power Consumption. / F. Gunter et al. // Intel Technology Journal. – 2001. – Vol. 5, №.1.

59. Havinga P.J.M. Low Power Systems Design Techniques for Mobile Computers / P.J.M. Havinga, G.J.M. Smit // Enschede, ISSN 1381-3625 : Centre for Telematics and Information Technology University of Twente. – 1997.

60. Hindle A. Green Software Engineering: The Curse of Methodology / Hindle A. // Proceedings: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 14-18 March 2016, P. 46–55, DOI: 10.1109/SANER.2016.60.

61. Horowitz M. Low-power digital design / M. Horowitz, T. Indermaur, R. Gonzalez // IEEE Symposium on Low Power Electronics (10-12 October 1994). – San Diego. – 1994.

62. Huang T.-C. Data dependence analysis for array references / T.-C. Huang, C.-M. Yang // Journal of Systems and Software. – 2000. – Vol. 52. – P. 55–65.

63. Kelly W. New User Interface for Petit and Other Extensions. / W. Kelly et al. // User Guide. CS Dept, University of Maryland. – 1996.

64. Kelly W. The Omega Library, Version 1.1.0., Interface Guide. [Електронний ресурс]. – 1996. – Режим доступу: http://www.csumd.edu/projects/omega

65. Kennedy Ken Optimizing Compilers for Modern Architectures – A dependence based approach / Ken Kennedy, Allen Randy // San Francisco, San Diego, New York : Morgan Kaufmann Publishers. – 2001.

66. Khan S. U. Green computing and communications / Samee Ullah Khan, Lizhe Wang, Laurence T. Yang, Feng Xia // Springer Science+Business Media, LLC 2011.

67. Kharchenko V. Green IT Engineering: Concepts, Movdels, Complex Systems Architectures, Studies in Systems, Decision and Control, / V. Kharchenko, Y. Kondratenko and J. Kacprzyk(eds.), // Springer, Vol. 74, Berlin (2017). doi:10.1007/978-3-319-44162-7)

68. Kodary K. Loop fusion for memory space optimization / K. Kodary, A. Mignotte // The 14th International Symposium on System Synthesis, 2001, Proceedings. – P. 95-100.

69. Lam M. S. The cache performance and optimizations of blocked algorithms / M. S. Lam, E. E. Rothberg, and M. E. Wolf // Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, p. 63–74, April 1991.

70. Lee S. Cetus – An Extensible Compiler Infrastructure for Source-to-Source Transformation / S. Lee, T. A. Johnson and R. Eigenma // Languages and Compilers for Parallel Computing, 16th Intl. Workshop. – 2003. – Vol. 2958. – P. 539-553.

71. Li Z. An efficient data dependence analysis for parallelzing compilers / Z. Li, P.-C. Yew and C.-Q. Zhu // IEEE Transaction on Parallel and Distributed Systems. – 1990. – Vol. 1(1). – P. 26–34.

72. Lim A. W. Communication-free parallelization via affine transformations / A. W. Lim, M. S. Lam // Languages and Compilers for Parallel Computing: 7th International Workshop Ithaca. – Berlin: Springer Berlin Heidelberg. – 1995. – P. 92-106.

73. J. McConnell, Analysis of algorithms / J. McConnell // 2nd ed. Jones & Bartlett Learning, 2007.

74. Mohammed A. M. Green Computing Beyond the traditional ways / Mohammed Anwar Mohammed, Danial Abdulkareem Muhammed, Jaza Mahmood Abdullah // University of Sulaimani Vol. 3 2015.

75. Min S.J. Portable compilers for openmp / S.J. Min, S. Kim // OpenMP Shared-Memory Parallel Programming, Lecture Notes in Computer Science . – 2001. – Vol. 2104, №7. – P. 11-19.

76. Petersen P. Static and Dynamic Evaluation of Data Dependence Analysis Techniques / P. Petersen, D. Padua // IEEE Trans. on Computers. – 1996. – Vol. 7, №11. – P. 1121-1132.

77. Piguet C. Low-power CMOS circuits: technology, logic design and CAD tools / C. Piguet // Boca Raton : CRC Press. – 2005. – P. 440.

78. Piguet C. Low-power processors and systems on chips / C. Piguet// Boca Raton: CRC Press. – 2005.

79. Pillai Anju S. Factors Causing Power Consumption in an Embedded Processor – A Study / Anju S. Pillai and T. B. Isha // International Journal of Application or Innovation in Engineering & Management. – 2013. – Vol. 2, №7. – P. 300-306.

80. Pouchet L.N. PolyBench/C the Polyhedral Benchmark suite [Електронний ресурс]. – Режим доступу: http://web.cse.ohio-state.edu/~pouchet/software/polybench/#description.

81. Pouchet Louis-Noël the Polyhedral Benchmark suite [Електронний ресурс] // PolyBench/C. – Режим доступу: http://web.cs.ucla.edu/~pouchet/software/polybench/.

82. Pugh W. Iteration Space Slicing and Its Application to Communication / W. Pugh, E. Rosser // Proceedings of the International Conference on Supercomputing. – 1997. – P. 221-228.

83. Pugh W. An Exact Method for Analysis of Value-based Array Data Dependences / W. Pugh, D. Wonnacott // Maryland, USA : Univ. of Maryland. CS-TR-3196. – 1993.

84. Pugh W. The Omega Test: a fast and practical integer programming algorithm for dependence analysis / W. Pugh // Communications of the ACM. – 1992. – Vol. 8, №35 – P. 4-13.

85. Roy K. Low Power CMOS VLSI Circuit Design. / K. Roy, S.D. Prasad // New York: John Wiley and Sons, Inc. – 2000.

86. Shinan W. Software power analysis and optimization for power-aware multicore systems: / Shinan W. // Shinan Wang – Detroit, 2014. – 177 p.

87. Sproch J. Low power design without compromise / J. Sproch et al. // International Symposium on Low Power Electronics and Design . – Monterey. – 1997.

88. The CLooG Code Generator in the Polyhedral Model's Home [Електронний ресурс]. – 2016. – Режим доступу: http://www.cloog.org.

89. The LooPo Project – Loop parallelization in the polytope model [Електронний ресурс]. – University of Passau. – Режим доступу: http://www.fmi.uni-passau.de/loopo.

90. Vasilache N. Polyhedral code generation in the real world. / N. Vasilache, C. Bastoul, A. Cohen // In Proceedings of the International Conference on Compiler Construction (ETAPS CC'06). – 2006. – P. 185-201.

91. Vasilache N. Automatic Correction of Loop Transformations / N. Vasilache, A. Cohen, L.-N. Pouchet // in Parallel Architectures and Compilation Techniques – Conference Proceedings, PACT, – Brasov, Romania, – 2007.

92. Veendrick H.J.M. Nanometer CMOS ICs: From Basics to ASICs. – New York: Springer, 2017. —638 p.

93. Veendrick H.J.M. Deep-Submicron CMOS ICs. From Basics to ASICs. / H.J.M. Veendrick // Dordrecht: Kluwer academic publishers. – 2000.

94. Wilson R.P. SUIF: An infrastructure for research on parallelizing and optimizing compilers / R.P. Wilson, R.S. French, C.S. Wilson // SIGPLAN Not. – 1994. – Vol. 29, №12. – P. 31–37.

95. Wolf M. E. A loop transformation theory and an algorithm to maximize parallelism / M. E. Wolf, M. S. Lam // Transactions on Parallel and Distributed Systems. – 1994. – Vol. 2, №4. – P. 452-470.

96. Wolf M. E. Improving locality and parallelism in nested loops / M.E. Wolf // Ph.D. Dissertation CSL-TR-92-538. – 1992.

97. Żygadło M. Green computing and energy storage systems / Monika Żygadło, Jerzy Kotowski, Jacek Oko // 10th Conference on Interdisciplinary Problems in Environmental Protection and Engineering EKO-DOK 2018, E3S Web of Conferences 44, 00202 (2018)

98. Ахо А. Компиляторы: принципы, технологии и инструментарий / Альфред Ахо, Моника Лам, Рави Сети, Джеффри Ульман // М.: «Вильямс», 2008. — 1184 с. — ISBN 978-5-8459-1349-4.

99. Борисова Н.В. Ефективне управління ресурсами вбудованих систем для обчислювальної техніки реального часу / Н.В. Борисова, Л.В. Шабанова-Кушнаренко // Системи обробки інформації. – 2018. №1(152). – с. 87-93.

100. Бурцев В. С. Новые принципы организации вычислительных процессов высокого параллелизма / В С. Бурцев // Труды Первой Всероссийской научной

конференции «Методы и средства обработки информации». – Москва. – 2003. – С. 17-32.

101. Вальковский В.А. Параллельное выполнение циклов. Метод параллелепипедов / В.А. Вальковский // Кибернетика. – 1982. – №2. – С. 51-62.

102. Вальковский В.А. Параллельное выполнение циклов. Метод пирамид / В.А. Вальковский // Кибернетика. – 1983. – №5. – С. 51-55.

103. Воеводин В.В. Параллельные вычисления / Воеводин В.В., Воеводин Вл.В // СПб: БХВ-Петербург. – 2002. – 608 С.

104. Дружиніна О.О. Підвищення ефективності функціонування веб-серверів з використанням технології прогнозування часових рядів на основі нейромереж / О.О. Дружиніна, Р.Н. Квєтний // Інформаційні технології та комп'ютерна інженерія – 2013. – №1. – С. 15-21.

105. Дорошенко А. Ю. Автоматизоване проектування та розпаралелювання програм для гетерогенних платформ із використанням алгебро-алгоритмічного інструментарію / А. Ю. Дорошенко, О. Г. Бекетов, М. М. Бондаренко, О. А. Яценко // Проблеми програмування. – 2020. – № 2-3. – С. 103–114.

106. Евстигнеев В.А. Анализ циклов: выбор кандидатов на распараллеливание / В.А. Евстигнеев, И.Л. Мирзуитова // Новосибирск : Ин-т систем информатики им. А.П. Ершова СО РАН. – 1999.

107. Іваненко П.А. Методи автоматизації створення автотюнерів для паралельних програм. Автореферат дисертації на здобуття наукового ступеня кандидата фізико-математичних наук, Спеціальність 01.05.03 – математичне та програмне забезпечення обчислювальних машин і систем. (на правах рукопису) – Київ. – 2018. – 25с.

108. Карпенко А.П. Обзор методов роя частиц для задачи глобальной оптимизации (Particle Swarm Optimization) / Карпенко А.П., Селиверстов Е.Ю. // Научное издание МГТУ им. Н.Э. Баумана “НАУКА И ОБРАЗОВАНИЕ”, Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". ISSN 1994-0408. №3, 2009.

109. Клименко І.А. Методи та засоби підвищення ефективності обробки інформації в реконфігурованих комп'ютерних системах на базі ПЛІС: / Клименко І.А. // Київ, 2017, 377 с.

110. Ковалев И.В. Использование метода роя частиц для формирования состава мультиверсионного программного обеспечения / Ковалев И.В., Соловьев Е.В., Ковалев Д.И., Бахмарева К.К., Демиш А.В. // Приборы и системы. управление, контроль, диагностика, Moscow, Научтехлитиздат , ISSN: 2073-0004 – 2013. – №3. – С. 1-6.

111. Курейчик В.М. Алгоритмы эволюционного роевого интеллекта в решении задачи разбиения графа / В.М. Курейчик, А.А. Кажаров. // Taganrog. Ed. ТРТУ, 2012.

112. Курейчик В.М. Использование роевого интеллекта в решении NP-трудных задач / В.М. Курейчик, А.А. Кажаров // Известия ЮФУ. Технические науки. – 2011. – №7(120). – С. 30-37.

113. Курейчик В.М. Генетические алгоритмы / В.М. Курейчик // Известия Южного федерального университета. Технические науки – 1998. – Т.8. – №2. – С. 4-7.

114. Лазоренко Д. И. Метод высокоуровневых трансформаций исходного кода описания цифровых систем с целью снижения их энергопотребления / Д.И. Лазоренко // Збірник наукових праць Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова. – 2007. – №38. С. 41-53.

115. Лебедев Б. Планирование на основе роевого интеллекта и генетической эволюции. / Лебедев Б., Лебедев В. // Коломна, 2009.

116. Лебедев Б.К. Разбиение на основе роевого интеллекта и генетической эволюции / Лебедев Б.К. Лебедев В.Б. // Труды V Международной научно-практической конференции Интегрированные модели и мягкие вычисления в искусственном интеллекте, Kolomna М., Физматлит, 2009.

117. Луцкий Г.М. Повышение эффективности кластеров на основе Infiniband / Г.М. Луцкий, И.С. Райзин // Вісник університету «Україна». Інформатика, обчислювальна техніка та кібернетика. – 2011. – №8. – С. 133.

118. Майника Э. Алгоритмы оптимизации на сетях и графах / Э. Майника // Москва : Мир. – 1981.

119. МакКоннелл Дж. Основы современных алгоритмов / МакКоннелл Дж. // Москва Техносфера, 2004.

120. Матренин П. В. Системное описание алгоритмов роевого интеллекта / П.В. Матренин, В.Г. Секаев // Программная инженерия, Теоретический и прикладной научно-технический журнал, ISSN 2220-3397 – 2013. – №12. – С. 39–45.

121. Падуа Д. Оптимизация в компиляторах для суперкомпьютеров. Векторизация программ: теория, методы, реализация / Д. Падуа, М. Вольф // Москва : Мир. – 1991.

122. Петренко В. В. Внутреннее представление ОРС 3 / В.В. Петренко // Труды Всероссийской научной конференции «Научный сервис в сети Интернет: технологии распределенных вычислений». – Новороссийск. – 2005. – С. 106 -108.

123. Самигулина Г.А Обзор современных методов роевого интеллекта для компьютерного молекулярного дизайна лекарственных препаратов / Г.А Самигулина, Ж.А. Масимканова // Проблемы информатики – 2016. – № 2 (31) – С. 50-61.

124. Свами М. Графы, сети и алгоритмы / М. Свами, К. Тхуласираман // Москва : Мир. – 1984. – 455 С.

125. Семенов В.Ю. Розв’язання систем нелінійних алгебраїчних рівнянь та задач мінімізації функцій: елементи теорії та застосування / Семенов В.Ю. // Автореф. дис. на здобуття наук. ступеня доктора фізико-математичних наук : [спец.] 01.05.02 – математичне моделювання та обчислювальні методи. – НАН України Інститут Кібернетики ім. В.М. Глушкова, Київ – 2019. – 35с.

126. Сушко С.В. Вплив розмірів блоків розбиття операторів циклів на час виконання комп’ютерних программ / С.В. Сушко, О.А. Чемерис // Моделювання та інформаційні технології. – 2018 – №82 – С. 110-117.

127. Сиволовський І. М. Огляд розпаралелюючих систем / І.М. Сиволовський // Восточно-Европейский журнал передовых технологий. – 2012. – Vol. 56, №2/10 – С. 10-13.

128. Филиппова А.С. Матричный алгоритм роя частиц для поддержки принятия решений при составлении расписания обслуживающих бригад / А.С. Филиппова, Э.И. Дяминова, Е.В. Андреева, Е.Д. Лаптенок // Труды Шестой всероссийской научной конференции "Информационные технологии интеллектуальной поддержки принятия решений", Уфа-Севастополь. – 2018. – С. 125-128.

129. Чемерис А.А. К вопросу распараллеливания последовательных программ / А.А. Чемерис, А.М. Пекар // В кн: Збірник наукових праць ІПМЕ НАН України. – Вип.11. – 2001. – С. 127-137.

130. Чемерис А.А. Использование многопотоковых программ в алгоритмах сортировки / А.А. Чемерис, А.М. Пекар // В кн: Збірник наукових праць ІПМЕ НАН України. – Вип.20. – 2003. – С. 112-117.

131. Чемерис А.А. Возможности уменьшения энергопотребления современных цифровых систем / А.А. Чемерис, Д.И. Лазоренко // Збірник наукових праць Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова. – 2005. – Вип. 30. – С. 27-33.

132. Чемерис А.А. Оптимизация программного обеспечения для снижения энергопотребления современных цифровых систем / А.А. Чемерис, Д.И. Лазоренко // в кн. «Моделювання та інформаційні технології», сб. наук. праць. – Київ. – 2006. – Вип. 37. – С. 16-19.

133. Чемерис А.А. Методы распараллеливания циклов / А.А. Чемерис, Ю.А. Полуян // В кн.: Збірник наукових праць Інституту проблем моделювання в енергетиці ім. Г.Є.Пухова. – 2009. – Вип. 53. – С. 81-89.

134. Чемерис А.А. Об одном методе распараллеливания тесногнездовых циклов / А.А. Чемерис, Ю.А. Горунова, А.Д. Смирнов // В кн.: Наукові праці ДонНТУ, Серія “Інформатика, кібернетика та обчислювальна техніка”, ISSN 1996-1588. – 2011. – Вип. 14(188). – С. 50-53.

135. Чемерис А.А. Трансформация программ с аффинным доступом к данным / А.А. Чемерис, А.Д. Смирнов // В кн.: Збірник наукових праць Інституту проблем моделювання в енергетиці, – 2011. – Вип. 59. – С. 20-29.

136. Чемерис А.А. Исследование возможности снижения энергопотребления вычислительного устройства путем использования режимов производительности процессора / А.А. Чемерис, Д.И. Лазоренко, С.Ю. Пронзелева // Радіоелектронні і комп'ютерні системи, Харків, «ХАІ». – 2013. – №5(64). – С. 183-185.

137. Чемерис А.А. Влияние программного обеспечения на энергопотребление параллельных вычислительных систем. / А.А. Чемерис // «Системи обробки інформації». – 2014. – Вип. 7 (123). – С.106-108.

138. Чемерис А.А. Определение зависимостей в компьютерных программах / А.А. Чемерис // В зб. наук. праць «Моделювання та інформаційні технології», ISSN 2309-76476. – 2015. – Вип. 74.– С. 3-9.

139. Чемерис А.А. Комплекс программных средств автоматического распараллеливания последовательных Си-программ для транспьютерных систем / В.Н. Белецкий, А.А. Чемерис, А.А. Багатеренко // Электронное моделирование. – 1994. – №2. – С. 8-16.

140. Чемерис А.А. Снижение энергопотребления цифровых устройств с помощью операции объединения циклов / А.А. Чемерис, Д.И. Лазоренко, В.В. Тарапата // Электронное моделирование. – 2010. – Т. 32, № 6. – С. 45-58.

141. Чемерис О.А. Теорія і методи трансформації циклічних операторів програм. Автореферат дисертації на здобуття наукового ступеня доктора технічних наук, Спеціальність 05.13.05 – комп'ютерні системи та компоненти. (на правах рукопису) – Київ, 36с.

142. Чемерис А.А. Трансформации кода программного обеспечения для снижения энергопотребления современных цифровых систем // А.А. Чемерис, Д.И. Лазоренко // В кн.: Тезисы Научно-технической конференции молодых ученых и специалистов «МОДЕЛИРОВАНИЕ», 13 января 2006, ИПМЭ НАНУ. – Киев. – 2006. – С. 13-14.

143. Чемерис А.А. Метод распараллеливания циклов на основе аффинных преобразований / А.А. Чемерис, Ю.А. Горунова // В кн.: Моделювання та інформаційні технології. Спеціалізований вип. Праці міжнародної наукової конференції «МОДЕЛЮВАННЯ-2010». 12-14 мая 2010. – 2010. – С. 287-293.

144. Чемерис А.А. Снижение энергопотребления вычислительных устройств с помощью использования режимов центрального процессора / А.А. Чемерис, Д.И. Лазоренко // Сборник трудов конференции «МОДЕЛИРОВАНИЕ-2012», Киев, 16-18 мая 2012. – Киев. – 2012. – ISBN 978-966-7690-11-3. – С. 252-254.

145. Чемерис А.А. Исследование возможности снижения энергопотребления вычислительного устройства путем использования режимов производительности процессора / А.А. Чемерис, Д.И. Лазоренко, С.Ю. Пронзелева // 3rd International Workshop Critical Infrastructure Safety and Security (CrISS 2013), Ukraine, Sevastopol, 23-26 May. – Севастополь. – 2013. – С. 245-247.

146. Чемерис А.А. Влияние оптимизации программного обеспечения на энергопотребление вычислительных систем / А.А. Чемерис, Д.В. Стась // Збірник тез VII Міжнародної науково-технічної конференції «Комп'ютерні системи та мережні технологи» (CSNT-2014). м. Київ. 17-19 квітня 2014 р НАУ – Київ. – 2014. – С. 153.

147. Чемерис А.А. Уменьшение энергопотребления компьютеров за счет оптимизации программного обеспечения / А.А. Чемерис, Д.И. Лазоренко / Материалы XXI междунар. конф. АВТОМАТИКА-2014, г.Киев, 23-27 сентября 2014. – К: Изд-во НТУУ «КПИ». – 2014. – С. 232-233.

148. Чемерис О.А. Енергоспоживання мікропроцесорних систем / О.А. Чемерис // Збірник тез доповідей IX міжнародної наукової конференції «Комп'ютерні системи і мережні технології» CSNT-2016, 21-23 квітня 2016 р. – 2016. – С. 75-76.

149. Чемерис А.А. Исследование эффективности выполнения программ, оптимизированных на основе полиэдральной модели / А.А. Чемерис, С. В. Сушко

// Сборник трудов 5й международной конференции SIMULATION-2016. – Киев. – 2016. – С. 241-244.

150. Чемерис А.А. Сравнение эффективности автоматической оптимизации на основе полиэдральной модели / А. А. Чемерис, С. В. Сушко // Summer InfoCom Advanced Solutions 2016, II Міжнародна науково-практична конференція, 1-3 червня 2016. – Режим доступу до ресурса: http://www.iconfs.net/infocom2016/sravnenye-effektyvnosty-avtomatycheskoj-optymyzatsyy-na-osnove-polyedralnoj-modely

151. Чемерис А.А. Оценка снижения времени выполнения тестовых программ при применении автоматической оптимизации на основе полиэдральной модели на Raspberry Pi 3 / А.А. Чемерис, С.В. Сушко // Winter InfoCom Advanced Solutions 2016, III Міжнародна науково-практична конференція, 1-2 грудня 2016. – Режим доступу до ресурса: http://www.iconfs.net/w.infocom2016/otsenka-snyzhenyya-vremeny-vypolnenyya-testovykh-programm-pry-prymenenyy-avtomatycheskoj-optymyzatsyy-na-osnove-polyedralnoj-modely-na-raspberry-pi-3.

152. Штейнберг Б.Я. Математические методы распараллеливания рекуррентных программных циклов на суперкомпьютеры с параллельной памятью / Б.Я. Штейнберг // Ростов-на-Дону : Издательство Ростовского университета. – 2004. – 192 С.

153. Штейнберг Б.Я. Состояние и возможности Открытой распараллеливающей системы / Б.Я. Штейнберг и др. // Шестая международная конференция "Перспективы систем информатики". Рабочий семинар "Наукоемкое программное обеспечение", 28-29 июня 2006 г. – Новосибирск, Академгородок, Россия. – 2006.

154. Штейнберг Б.Я. Открытая распараллеливающая система / Б.Я. Штейнберг // Труды международной конференции «Параллельные вычисления и задачи управления» PACO’2001, ИПУ РАН. – Москва. – 2001. – Р. 214-220.

155. Шульженко А.М. Решетчатый граф и использующие его преобразования программ в Открытой распараллеливающей системе /

А.М. Шульженко // Труды Всероссийской научной конференции «Научный сервис в сети Интернет: технологии распределенных вычислений». – Новороссийск. – 2005. – P. 82-85.

ДОДАТОК А

СПИСОК ПУБЛІКАЦІЙ ЗДОБУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

Наукові праці, в яких опубліковані основні наукові результати дисертації

1. Chemeris A. Influence of Software Optimization on Energy Consumption of Embedded Systems / Chemeris A., Lazorenko D., Sushko S. // Green IT Engineering: Components, Networks and Systems Implementation. Studies in Systems / Kharchenko V., Kondratenko Y., Kacprzyk J. (eds) – Cham.: Springer, 2017. – Vol. 105. – С. 111–133.

2. Чемерис А.А. Исследование быстродействия и энергопотребления при автоматической оптимизации методами разбиения на блоки и распараллеливания для вычислений на платформе X64 / А.А. Чемерис, С.В. Сушко // Моделювання та інформаційні технології. – Київ, 2017. – Вип. 80. – С. 52-60.

3. Сушко С.В. Вплив розмірів блоків розбиття операторів циклів на час виконання комп'ютерних програм / С.В. Сушко, О.А. Чемерис // Моделювання та інформаційні технології. – Київ, 2018. – Вип. 82. – С. 110-117.

4. Sushko S. Dependency between Tiles' Sizes and Program Execution Time / Sushko S., Chemerys A. // Measurement Automation Monitoring. – Gliwice, 2018. – Vol. 64, No.2, P. 28-30, Wydawnictwo PAK, ISSN 2450-2855.

5. Chemerys A. Analysis and Optimization of the Sizes of the Iteration Space Tiles During the Parallelization of Program Loop Operators / Chemerys A., Sushko S. // Advances in Cyber-Physical Systems. – Київ, 2018. – Випуск 3, Номер 1, С. 1-6.

6. Сушко С.В. Визначення енергоефективності обчислень на різних обчислювальних системах / С.В. Сушко, О.А. Чемерис // Моделювання та інформаційні технології. – Київ, 2018. – Вип. 85. – С. 34-39.

Наукові праці, які засвідчують апробацію матеріалів дисертації

1. Чемерис А.А. Исследование эффективности выполнения программ, оптимизированных на основе полиэдральной модели / А.А. Чемерис, С.В. Сушко //

Сборник трудов 5й международной конференции SIMULATION-2016. – Киев, 2016. – С. 241-244.

2. Чемерис А.А. Сравнение эффективности автоматической оптимизации на основе полиэдральной модели / А.А. Чемерис, С. В. Сушко // Summer InfoCom 2016: Матеріали II Міжнародної науково-практичної конференції, м. Київ, 1-3 червня 2016 р. – К.:Вид-во "Інжиніринг", 2016. – С. 74-76.

3. Чемерис А.А. Оценка снижения времени выполнения тестовых программ при применении автоматической оптимизации на основе полиэдральной модели на Raspberry Pi 3 / А.А. Чемерис, С.В. Сушко // Winter InfoCom 2016: Матеріали III Міжнародної науково-практичної конференції, м. Київ, 1-2 грудня 2016 р. – К.:Вид-во «Інжиніринг», 2016. – С. 63-65.

4. Chemeris A. The Dependence of Microprocessor System Energy Consumption on Software Optimization / A. Chemeris, S. Sushko // 2017 IEEE 37th International Conference on Electronics and Nanotechnology (ELNANO), April 18-20, 2017, Kyiv, ISBN: 978-1-5386-1700-7, P. 451-454.

5. Сушко С.В. Вплив розмірів блоків розбиття на час виконання програм. / С.В. Сушко // Науково-технічна конференція молодих вчених і спеціалістів ІПМЕ ім. Г. Є. Пухова НАН України, – Київ, 12 травня 2018, – С. 34-37.

6. Sushko S. Increasing An Energy Efficiency Of Computational System By Using Automatic Software Optimization / S. Sushko, A. Chemeris // 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT-2018), May 24-27 2018, Kyiv, ISBN 978-1-5386-5902-1, P. 613-617.

7. Сушко С. Вплив розмірів блоків розбиття операторів циклів на час виконання комп'ютерних програм / Сушко С. Чемерис О. // Збірник праць 6-ї міжнародної конференції Моделювання-2018, 12-14 вересня 2018, Київ, С. 234-238.

8. Чемерис О.А. Методи штучного інтелекту при оптимізації роботи мікропроцесорних систем. / О.А. Чемерис, С.В. Сушко // XII Міжнародна Науково-Практична Конференція Комп'ютерні Системи та Мережні Технології 28-30 березня 2019, Київ, С. 127-129.

9. Chemeris A. Usage of Discrete Particle Swarm Optimization Method for the Searching of Optimal Tile Size / A. Chemeris, S. Sushko // 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T-2019), October 8-11, 2019, Kyiv, P. 202-206. ISBN: 978-1-7281-4183-1.

10. Сушко С.В. Універсальні оптимізаційні методи програмного забезпечення / Сушко С.В. // Науково-практична конференція Безпека енергетики в епоху цифрової трансформації, 20 грудня 2019, Київ, С. 89-91.

ДОДАТОК Б

Залежність часу виконання тестових програм від розмірів блоків розбиття при застосуванні методу розбиття на блоки

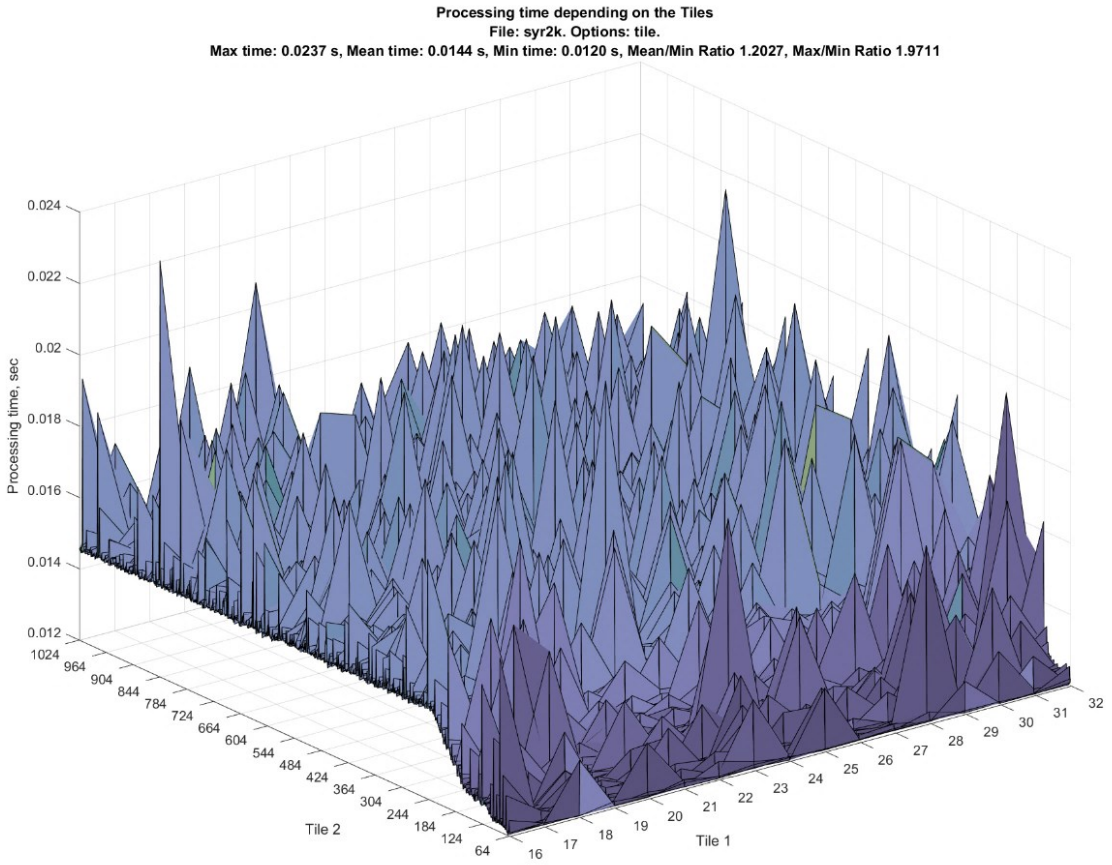

Рисунок Б.1 – Залежність часу виконання від розмірів блоків розбиття для тестової програми syr2k

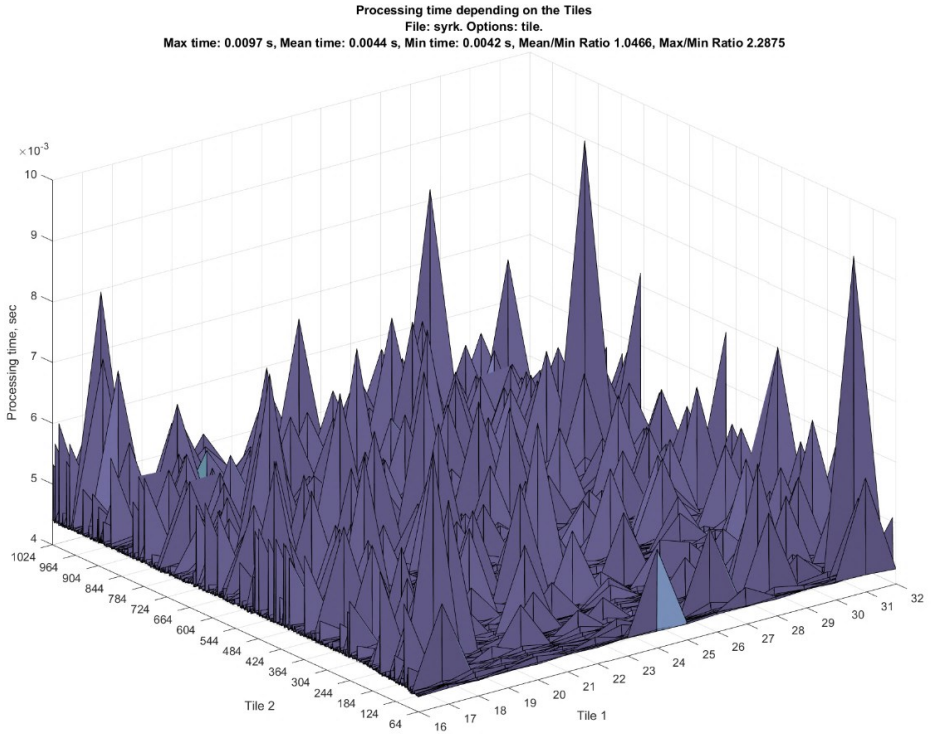

Рисунок Б.2 – Залежність часу виконання від розмірів блоків розбиття для тестової програми syrk

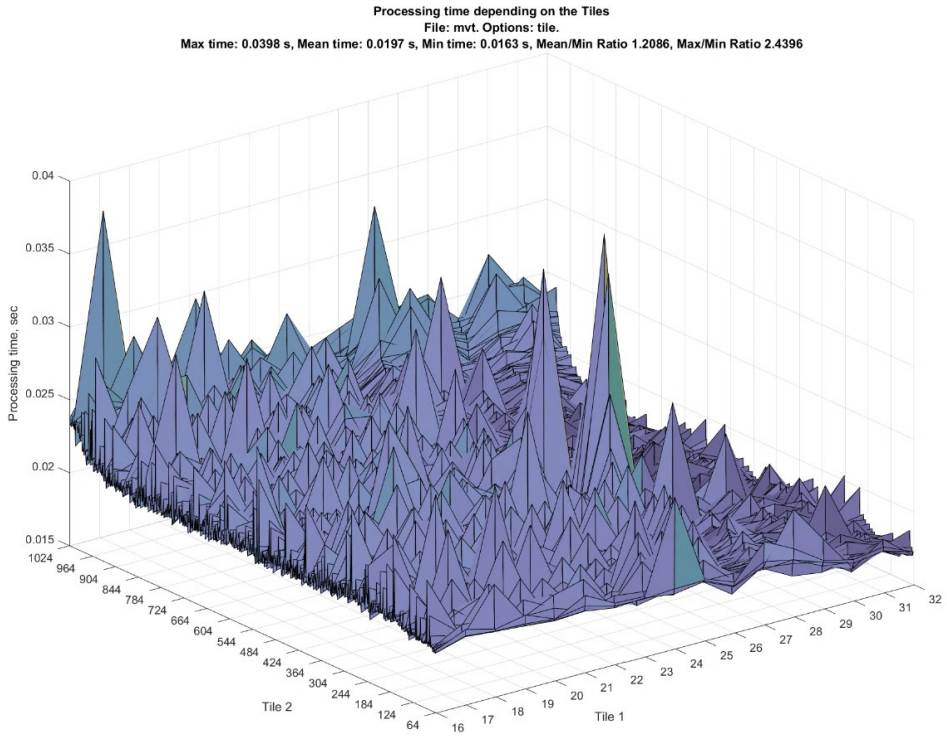

Рисунок Б.3 – Залежність часу виконання від розмірів блоків розбиття для тестової програми mvt

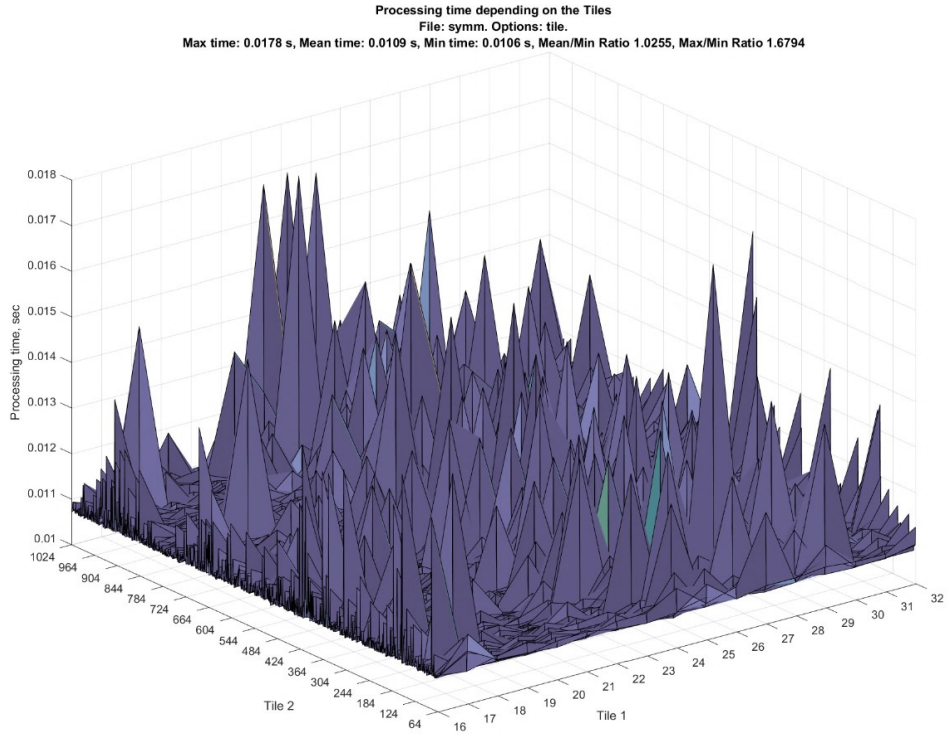

Рисунок Б.4 – Залежність часу виконання від розмірів блоків розбиття для тестової програми symm

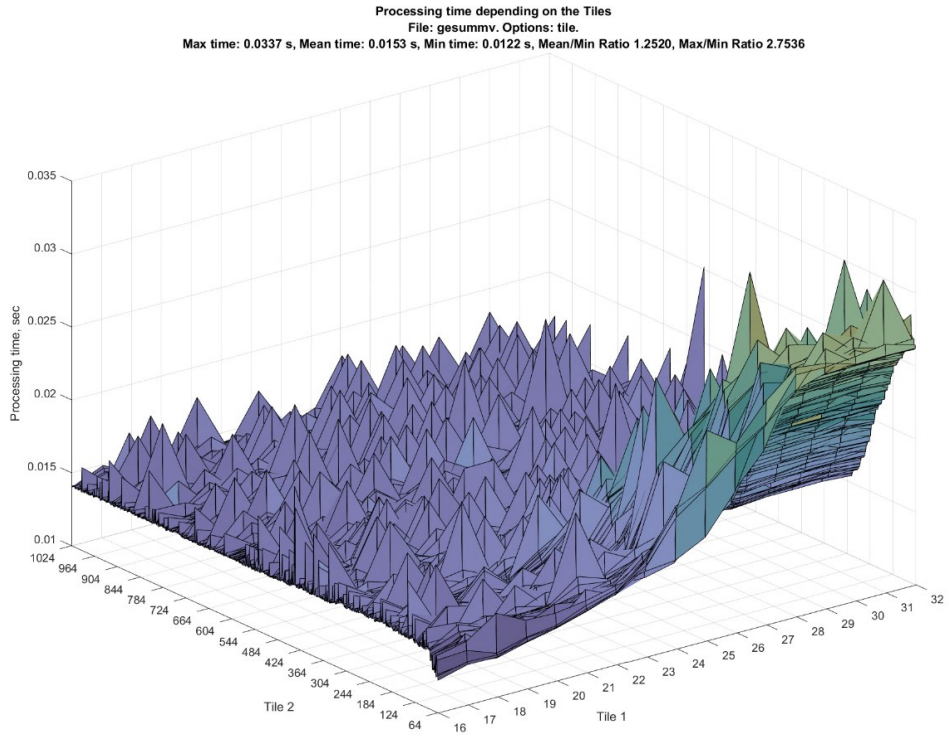

Рисунок Б.5 – Залежність часу виконання від розмірів блоків розбиття для тестової програми gesummv

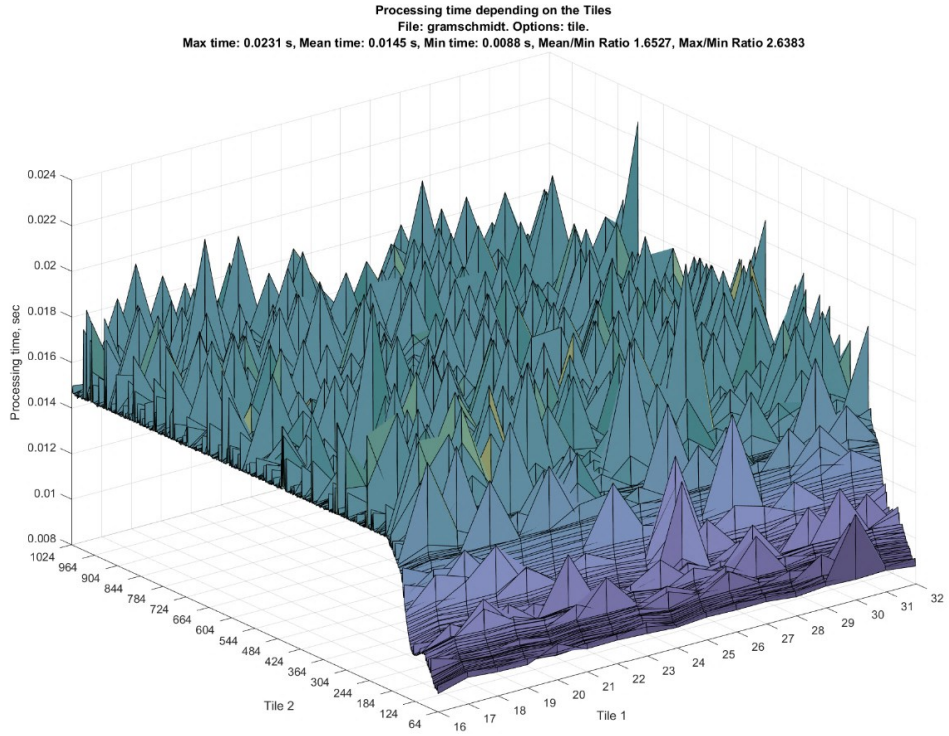

Рисунок Б.6 – Залежність часу виконання від розмірів блоків розбиття для тестової програми gramschmidt

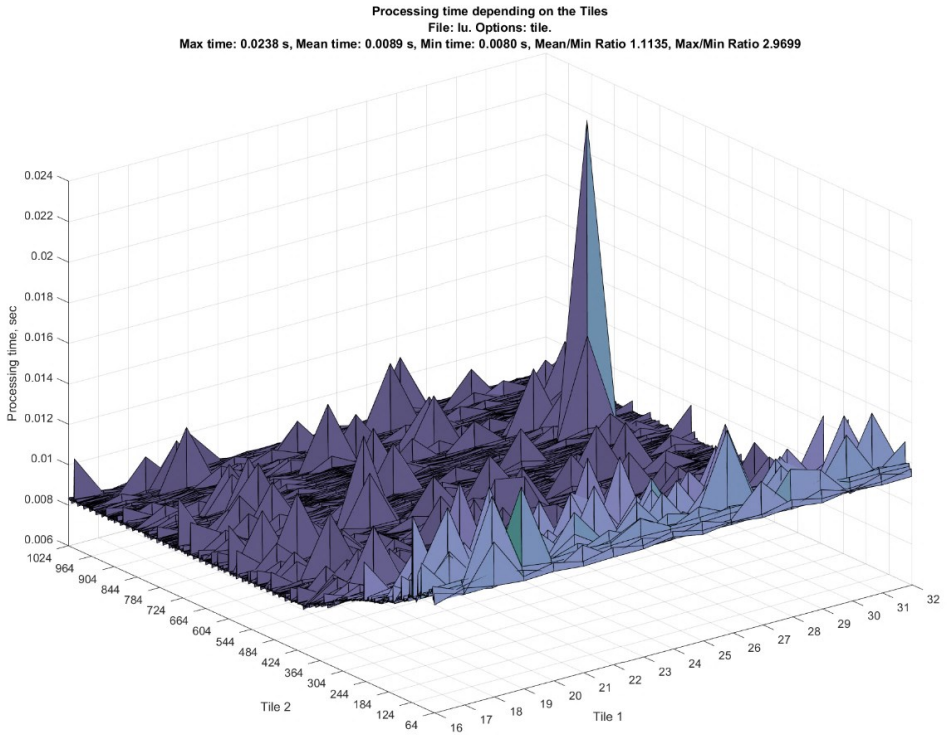

Рисунок Б.7 – Залежність часу виконання від розмірів блоків розбиття для тестової програми lu

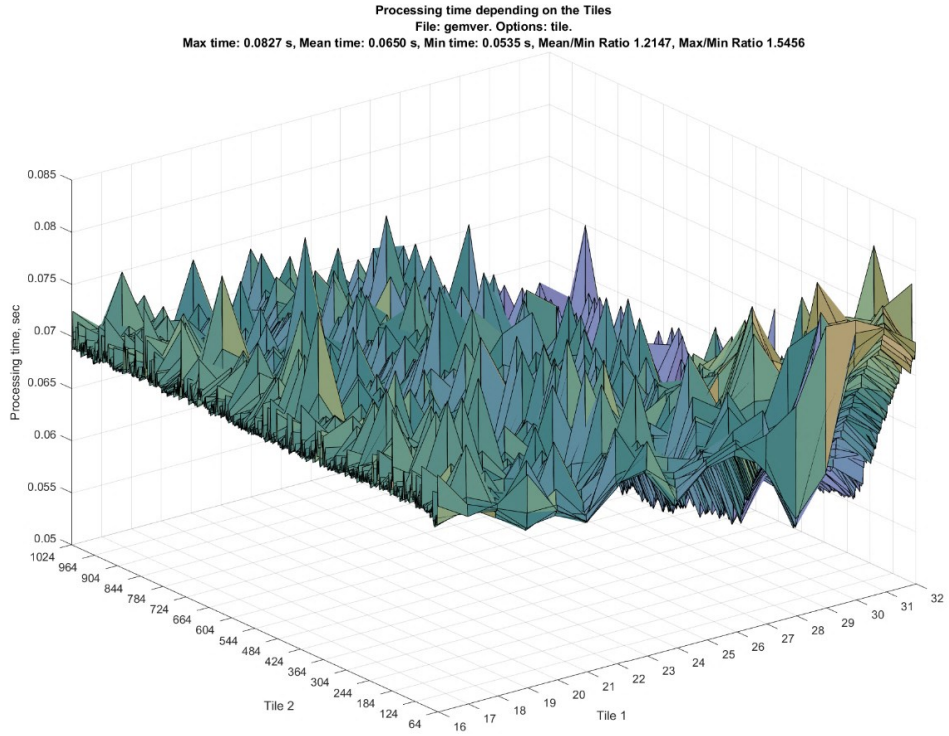

Рисунок Б.8 – Залежність часу виконання від розмірів блоків розбиття для тестової програми gemver

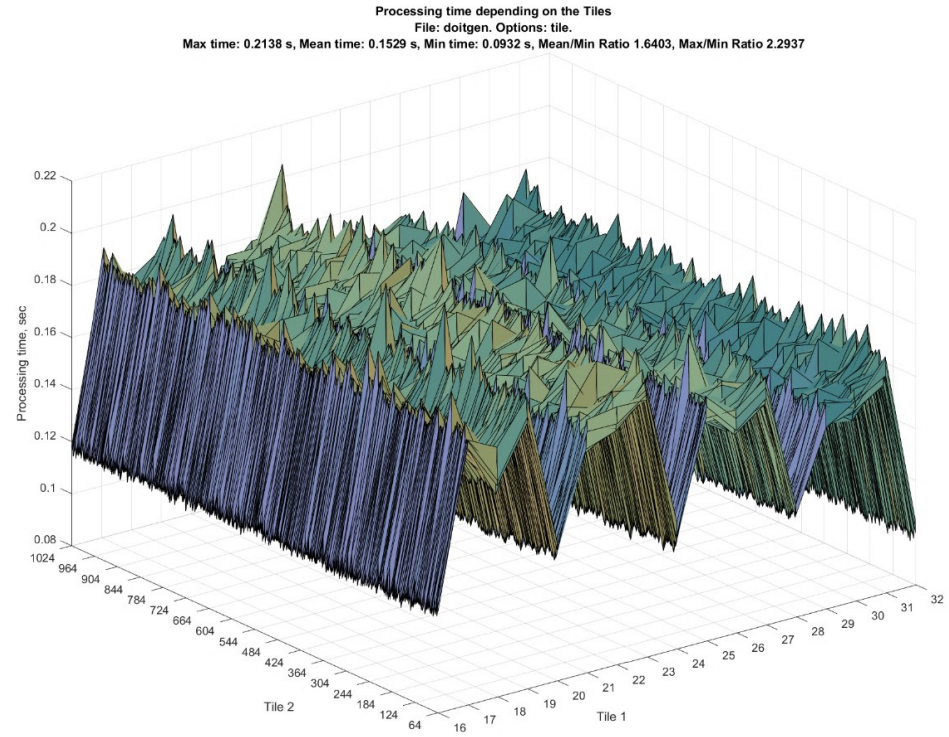

Рисунок Б.9 – Залежність часу виконання від розмірів блоків розбиття для тестової програми doitgen

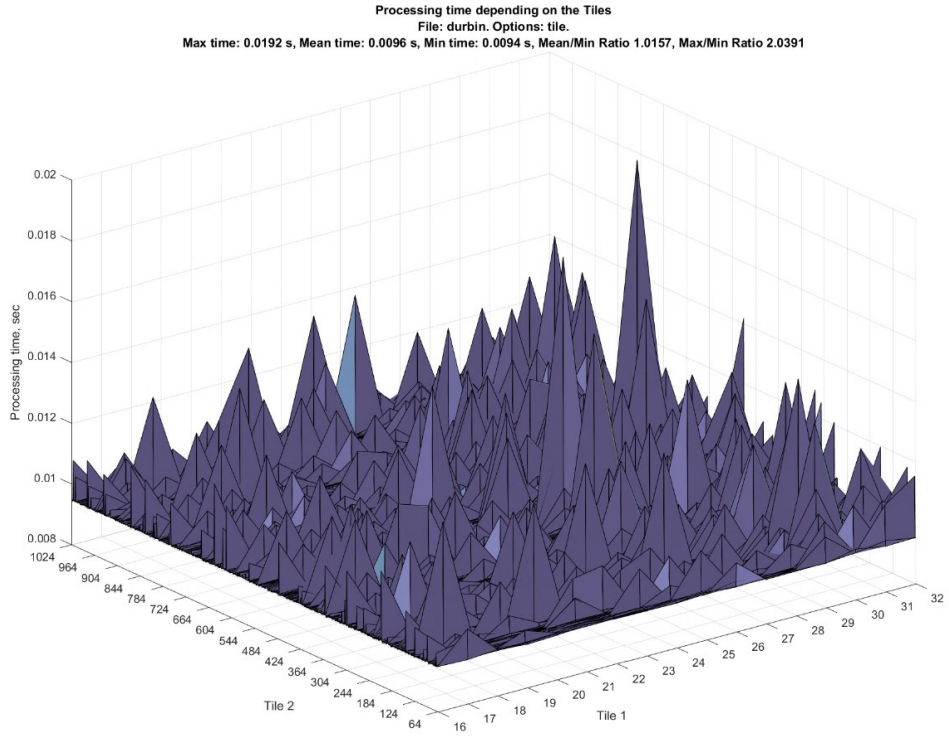

Рисунок Б.10 – Залежність часу виконання від розмірів блоків розбиття для тестової програми durbin

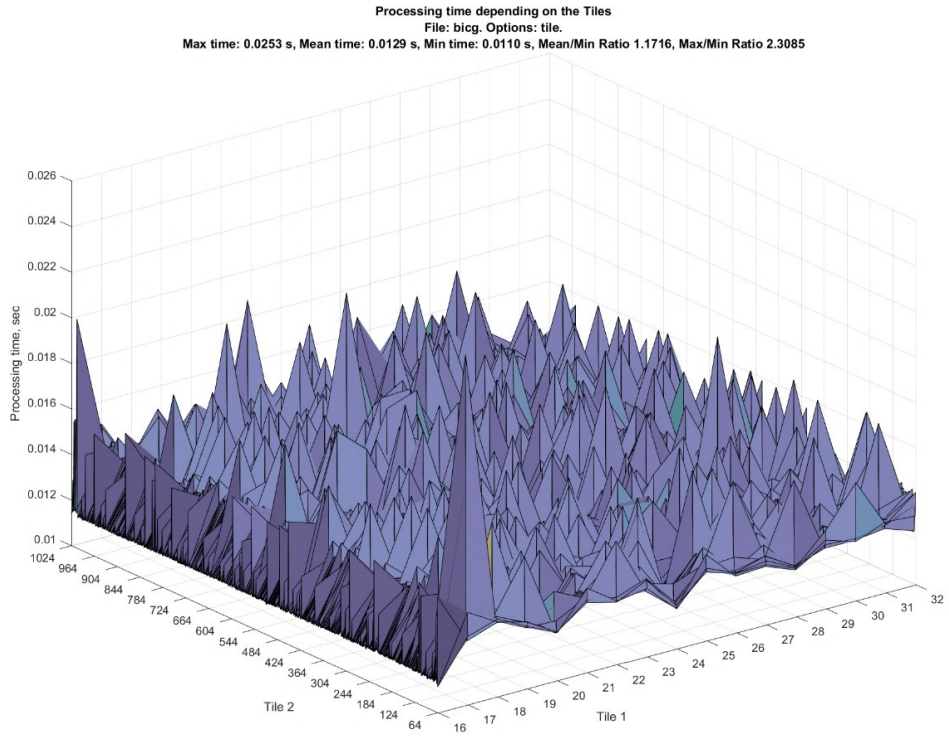

Рисунок Б.11 – Залежність часу виконання від розмірів блоків розбиття для тестової програми bicg

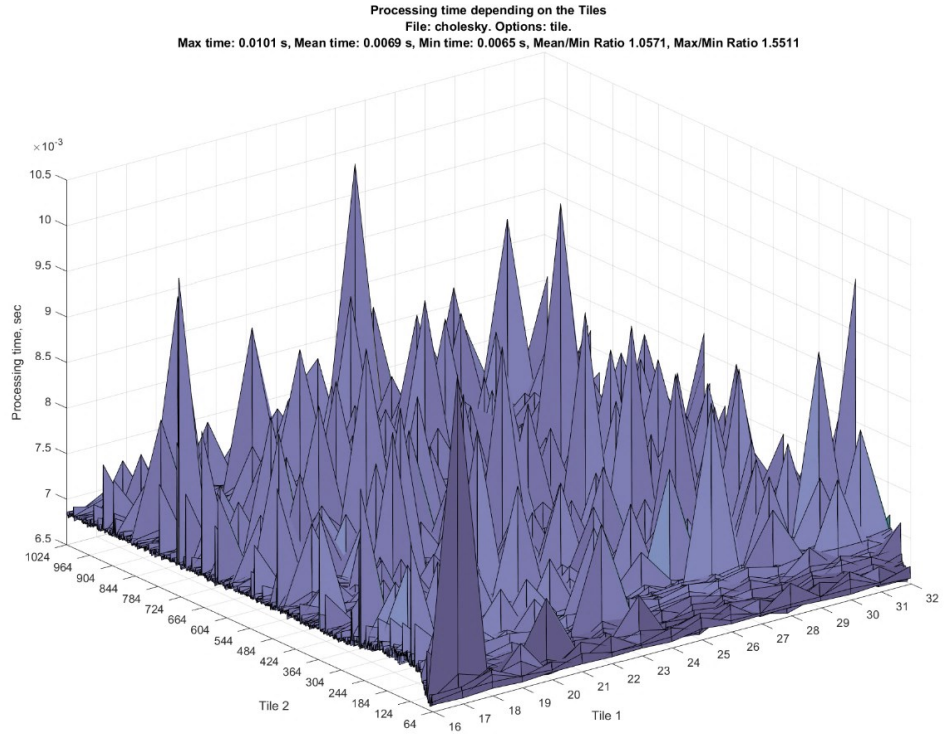

Рисунок Б.13 – Залежність часу виконання від розмірів блоків розбиття для тестової програми cholesky

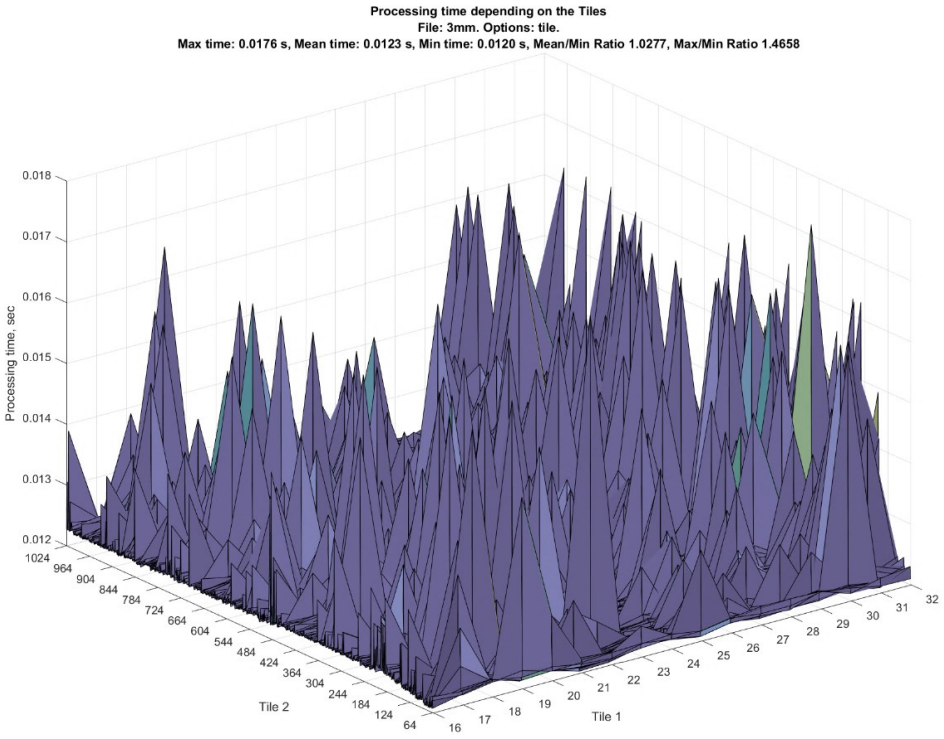

Рисунок Б.14 – Залежність часу виконання від розмірів блоків розбиття для тестової програми 3mm

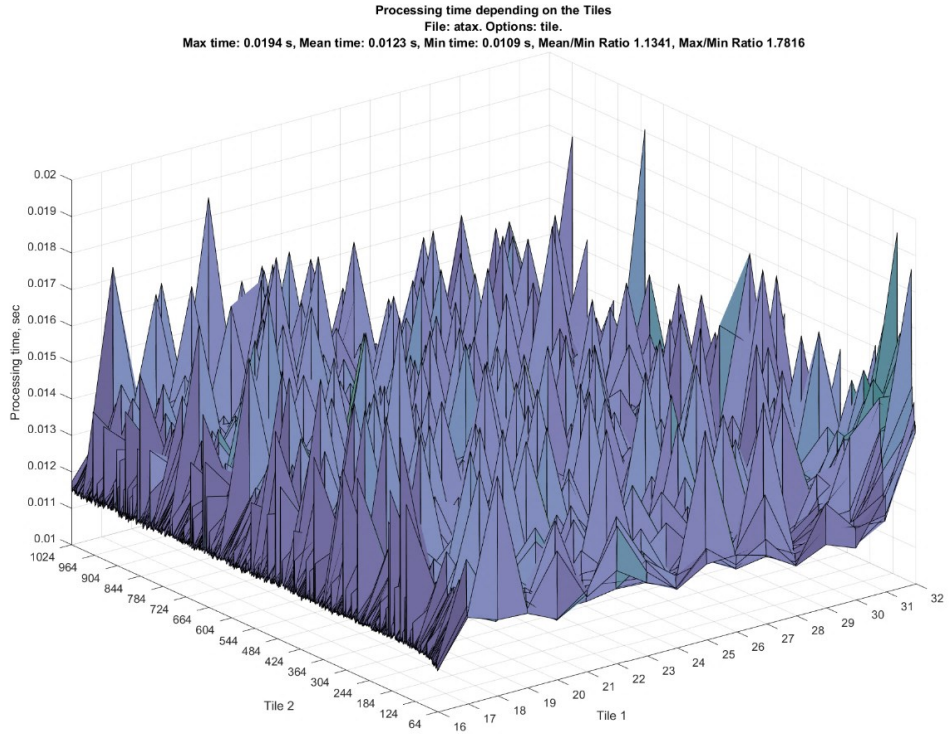

Рисунок Б.15 – Залежність часу виконання від розмірів блоків розбиття для тестової програми atax

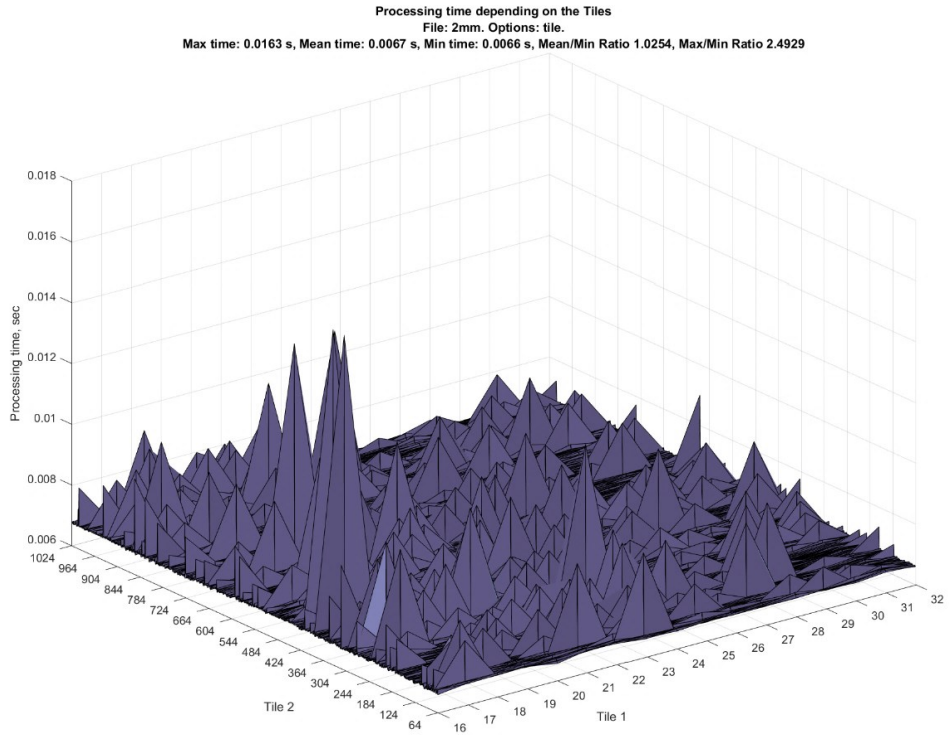

Рисунок Б.16 – Залежність часу виконання від розмірів блоків розбиття для тестової програми 2mm

ДОДАТОК В

Залежність часу виконання тестових програм від розмірів блоків розбиття при застосуванні методу розбиття на блоки та розпаралелюванні

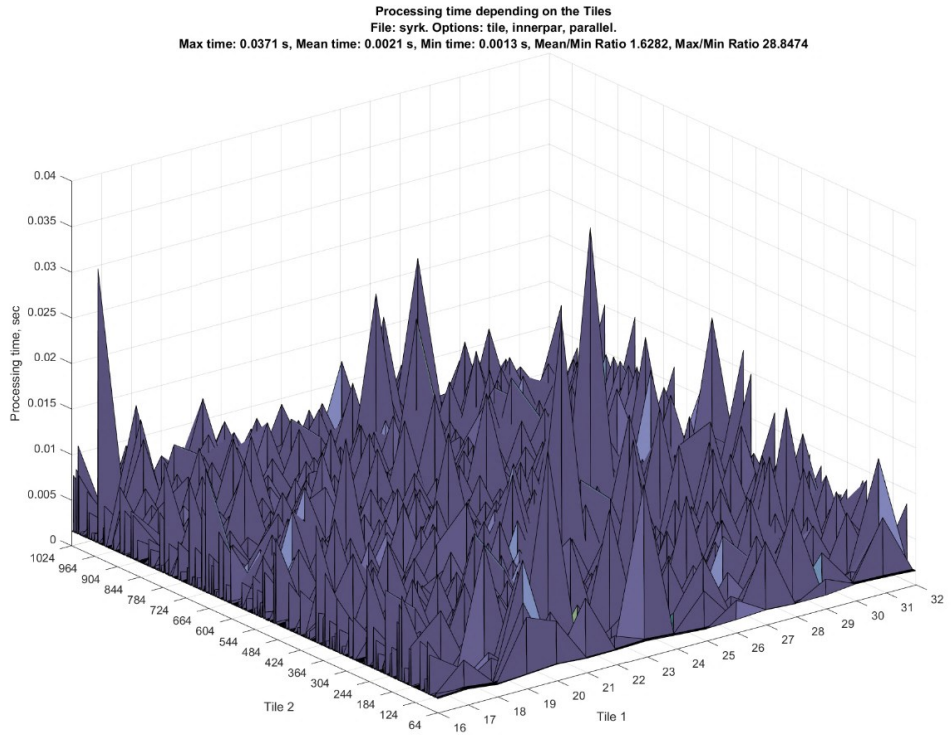

Рисунок В.1 – Залежність часу виконання від розмірів блоків розбиття для тестової програми syrk

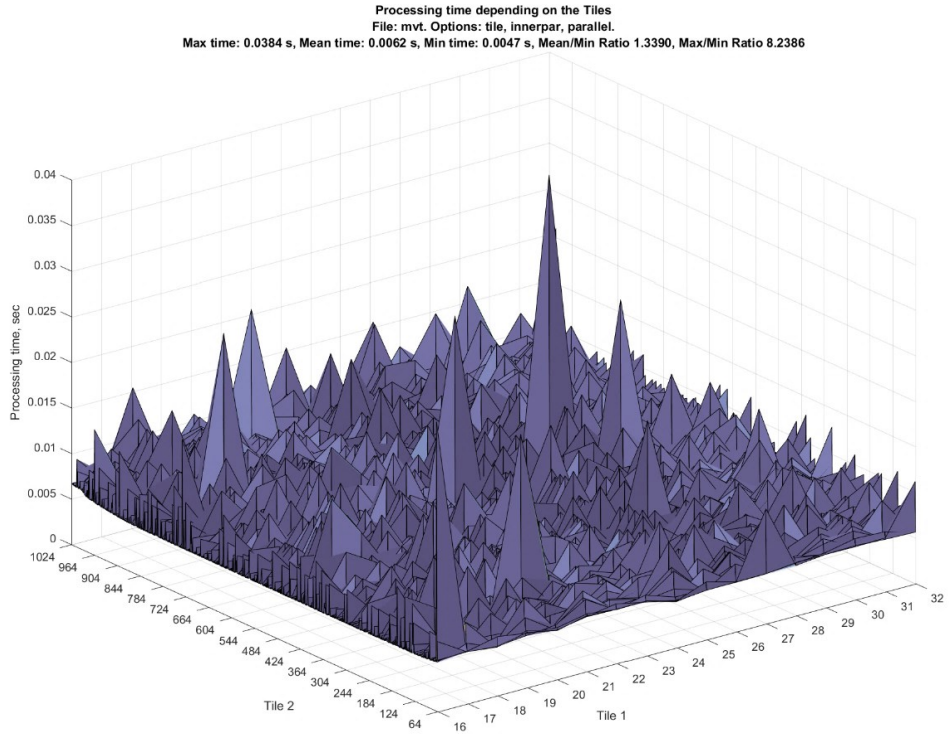

Рисунок В.2 – Залежність часу виконання від розмірів блоків розбиття для тестової програми mvt

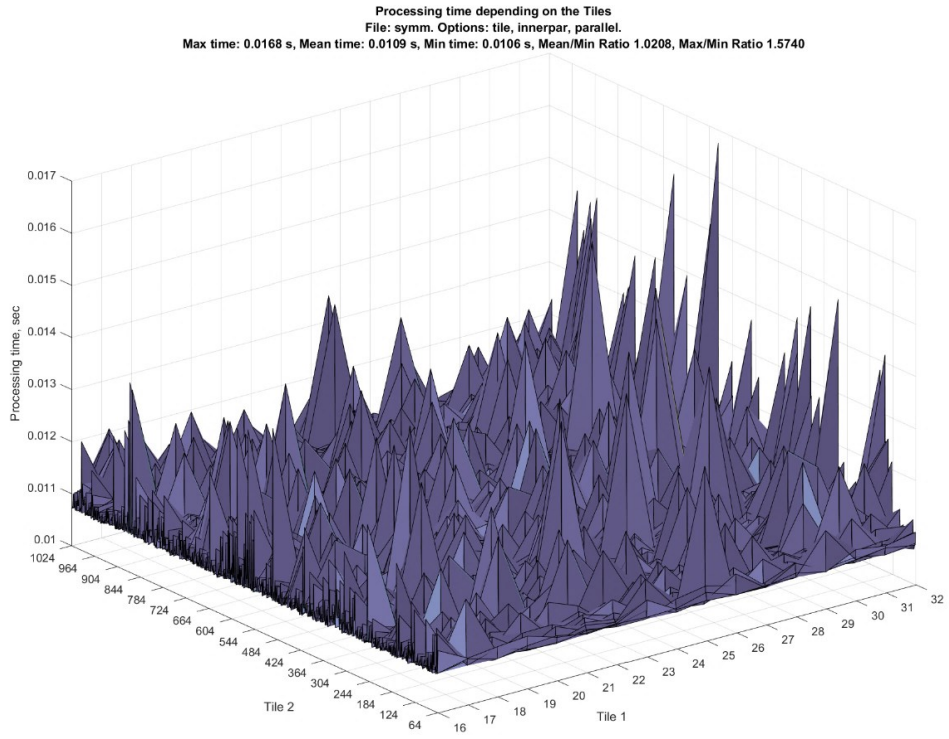

Рисунок В.3 – Залежність часу виконання від розмірів блоків розбиття для тестової програми symm

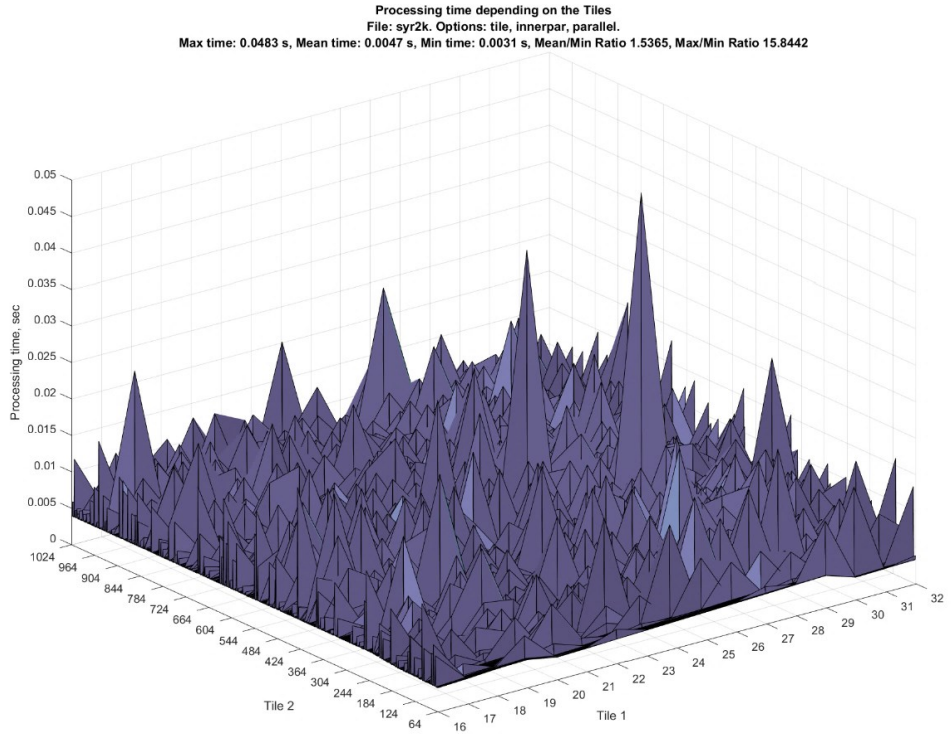

Рисунок В.4 – Залежність часу виконання від розмірів блоків розбиття для тестової програми syr2k

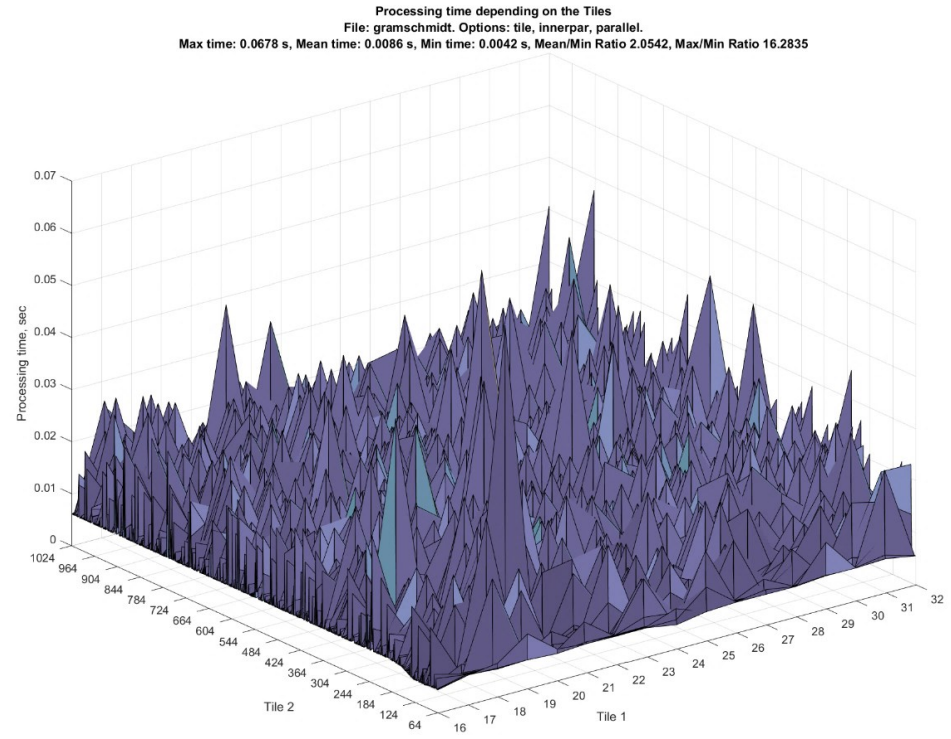

Рисунок В.5 – Залежність часу виконання від розмірів блоків розбиття для тестової програми gramschmidt

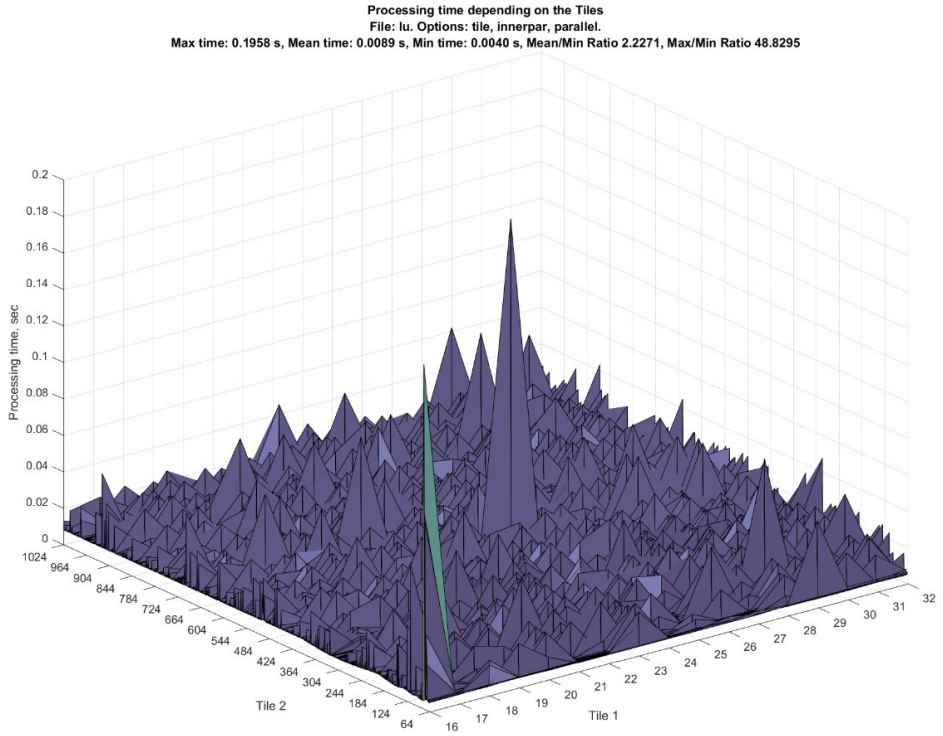

Рисунок В.6 – Залежність часу виконання від розмірів блоків розбиття для тестової програми lu

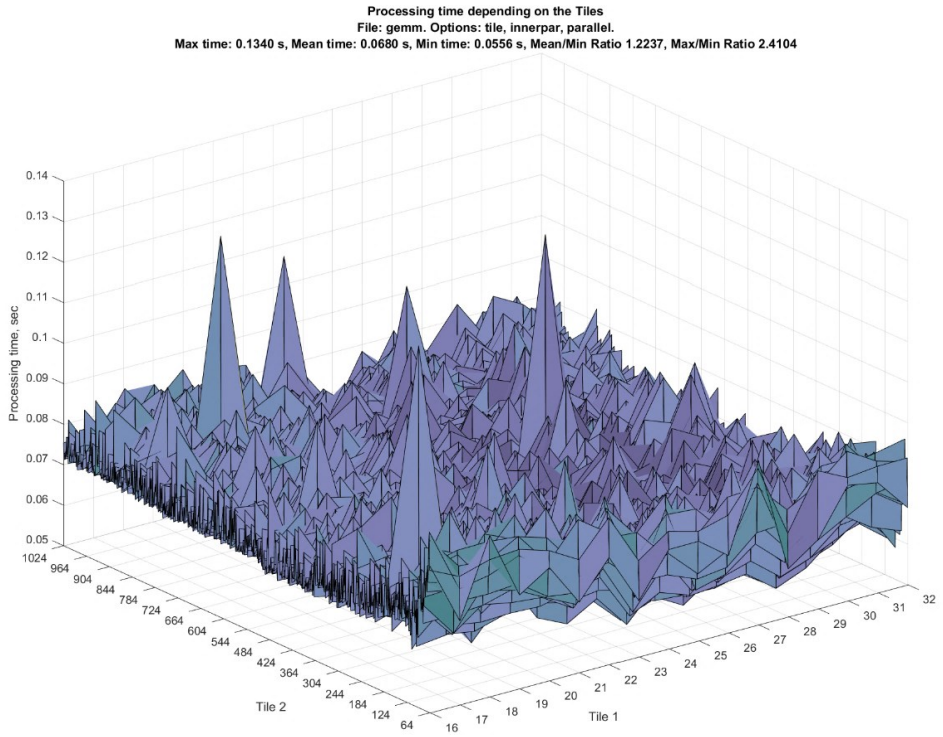

Рисунок В.7 – Залежність часу виконання від розмірів блоків розбиття для тестової програми gemm

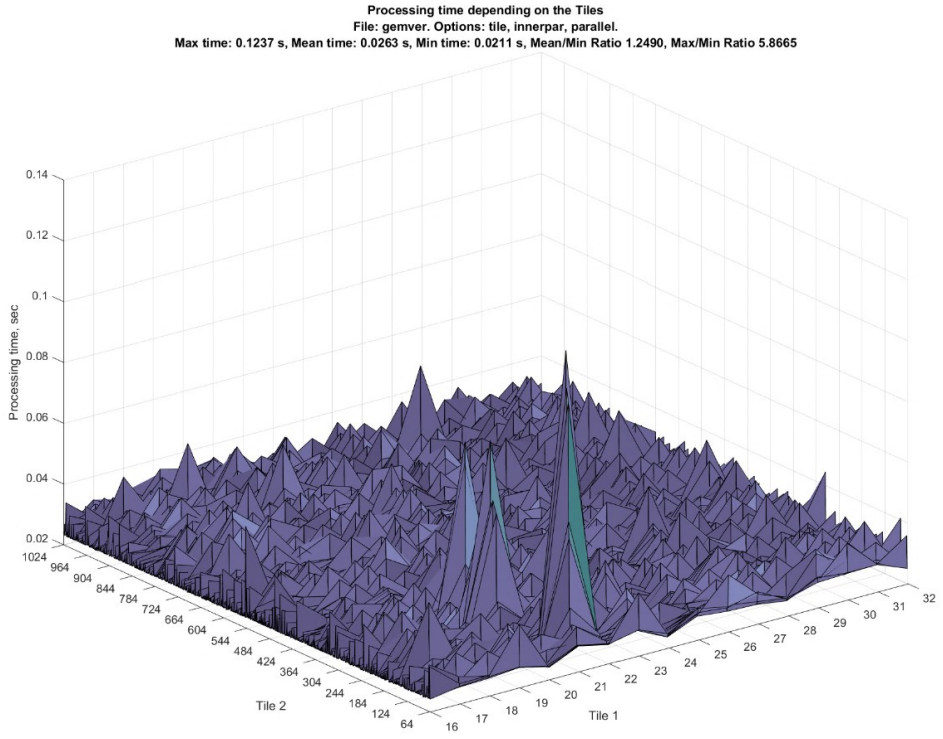

Рисунок В.8 – Залежність часу виконання від розмірів блоків розбиття для тестової програми gemver

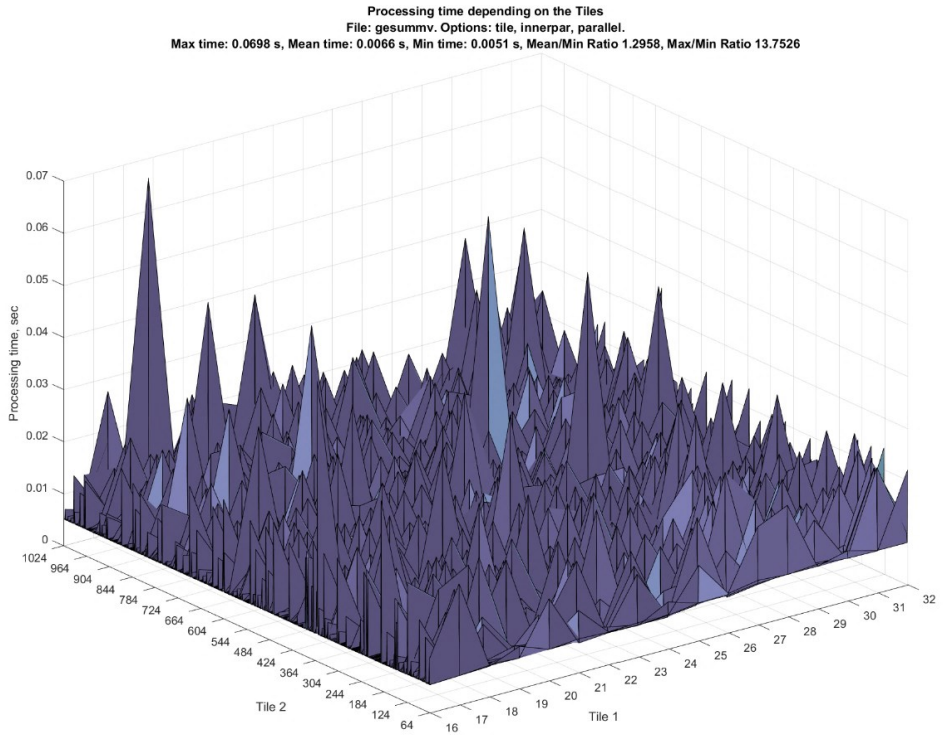

Рисунок В.9 – Залежність часу виконання від розмірів блоків розбиття для тестової програми gesummv

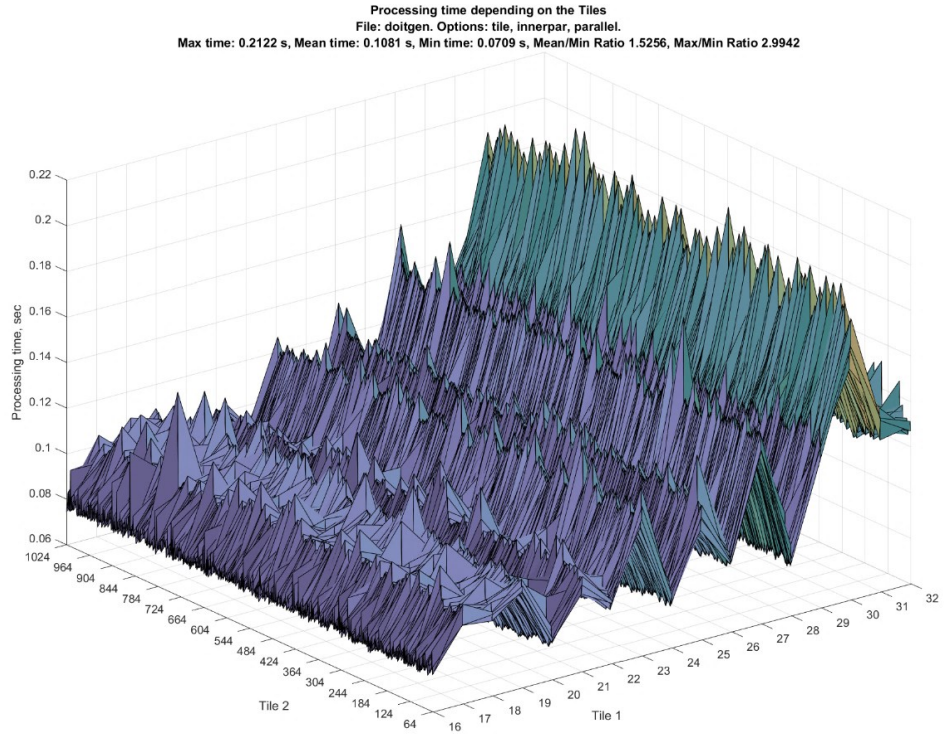

Рисунок В.10 – Залежність часу виконання від розмірів блоків розбиття для тестової програми doitgen

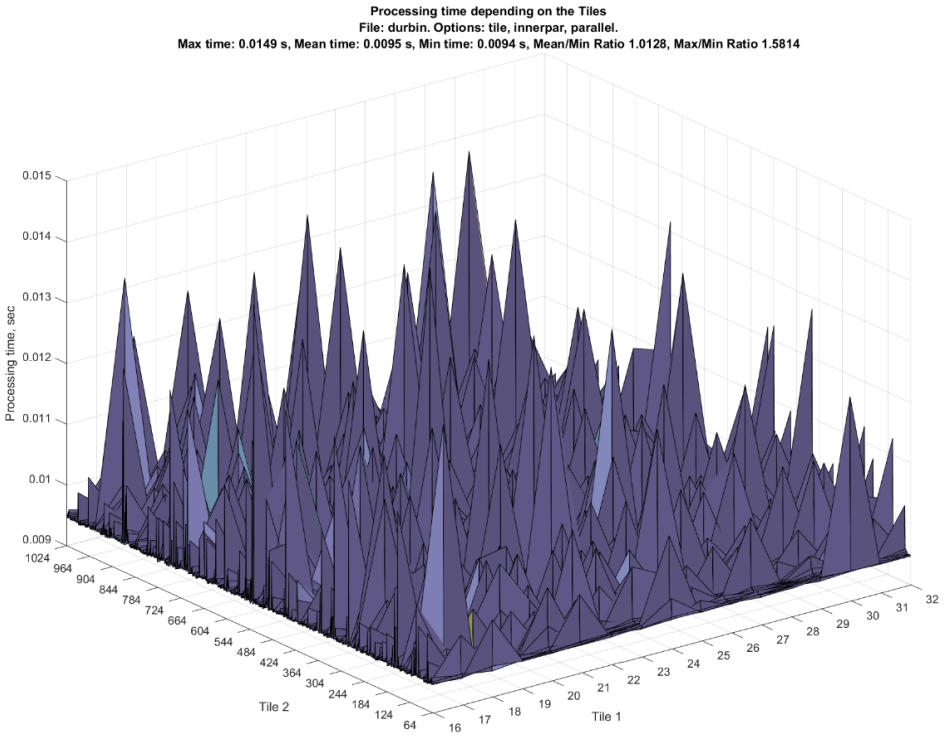

Рисунок В.11 – Залежність часу виконання від розмірів блоків розбиття для тестової програми durbin

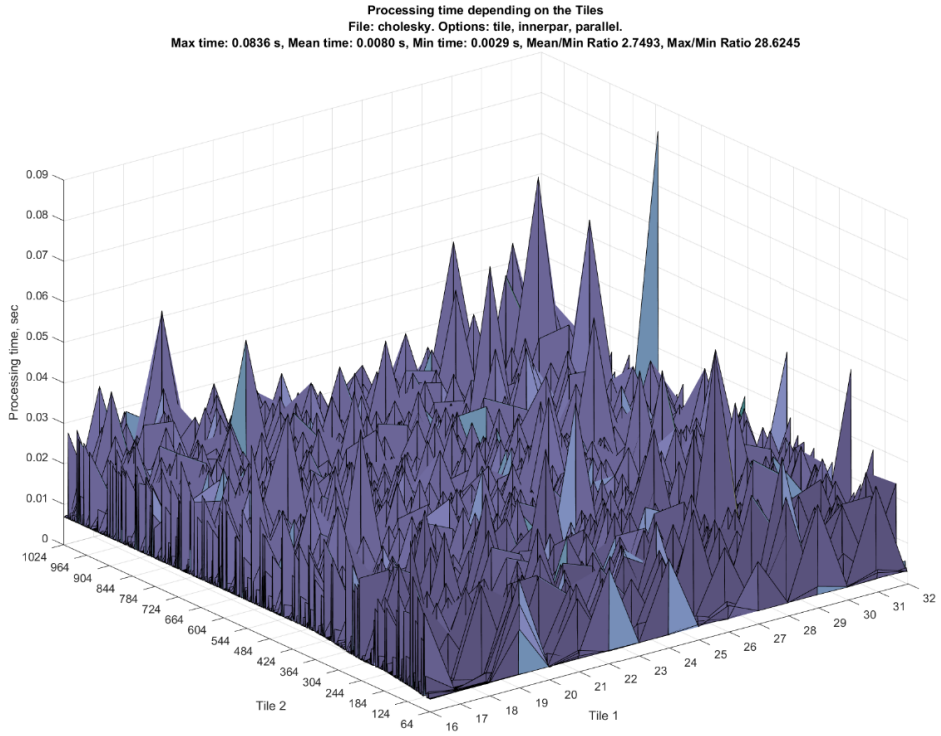

Рисунок В.12 – Залежність часу виконання від розмірів блоків розбиття для тестової програми cholesky

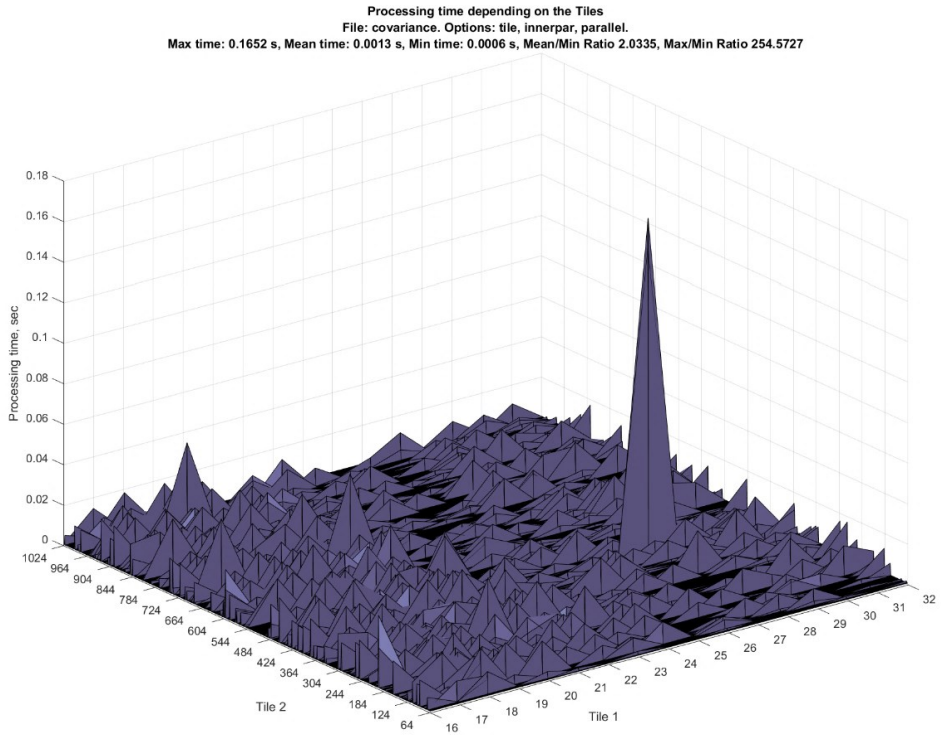

Рисунок В.13 – Залежність часу виконання від розмірів блоків розбиття для тестової програми covariance

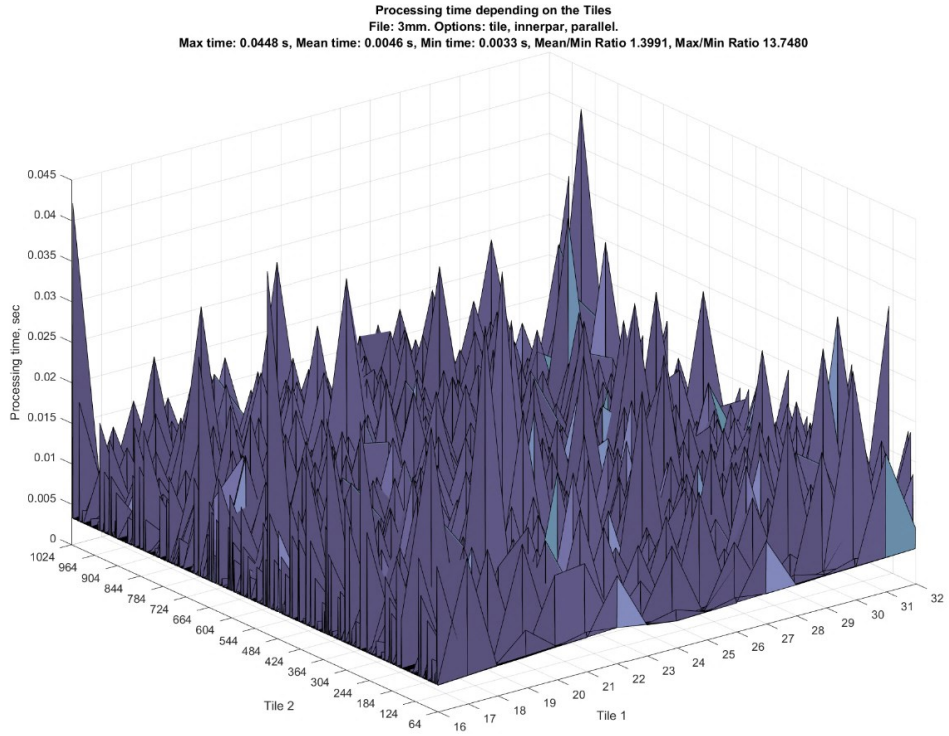

Рисунок В.14 – Залежність часу виконання від розмірів блоків розбиття для тестової програми 3mm

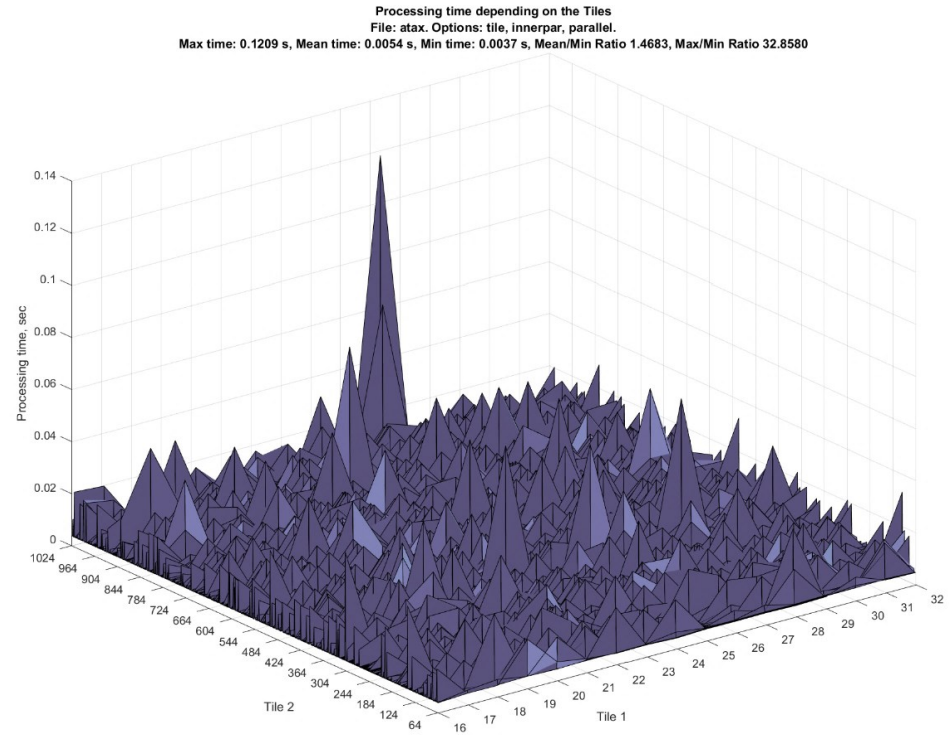

Рисунок В.15 – Залежність часу виконання від розмірів блоків розбиття для тестової програми atax

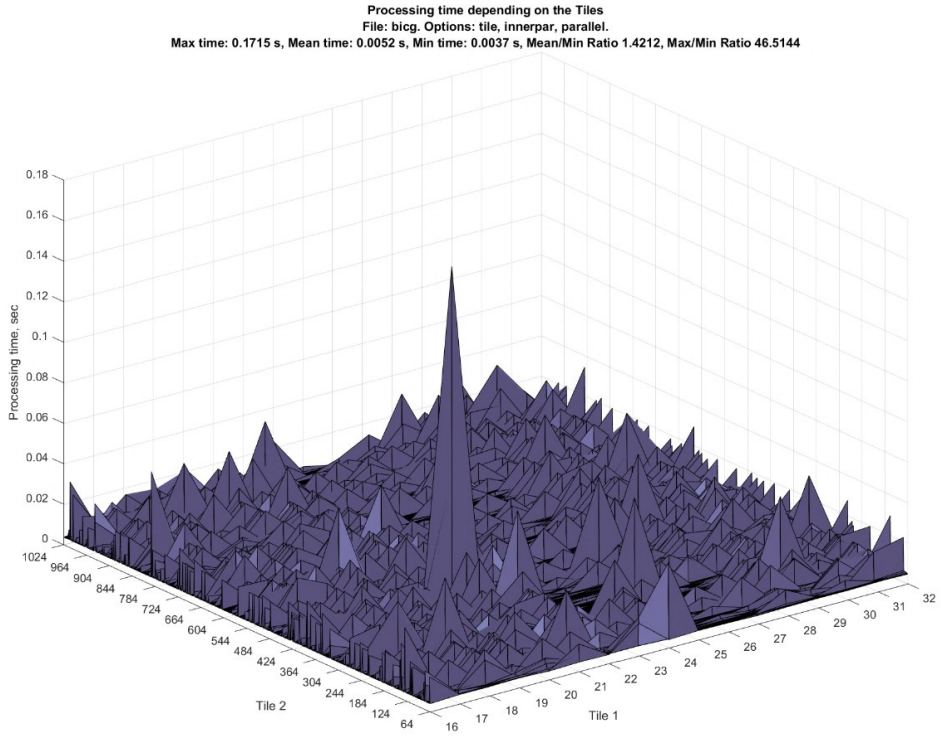

Рисунок В.16 – Залежність часу виконання від розмірів блоків розбиття для тестової програми bicg

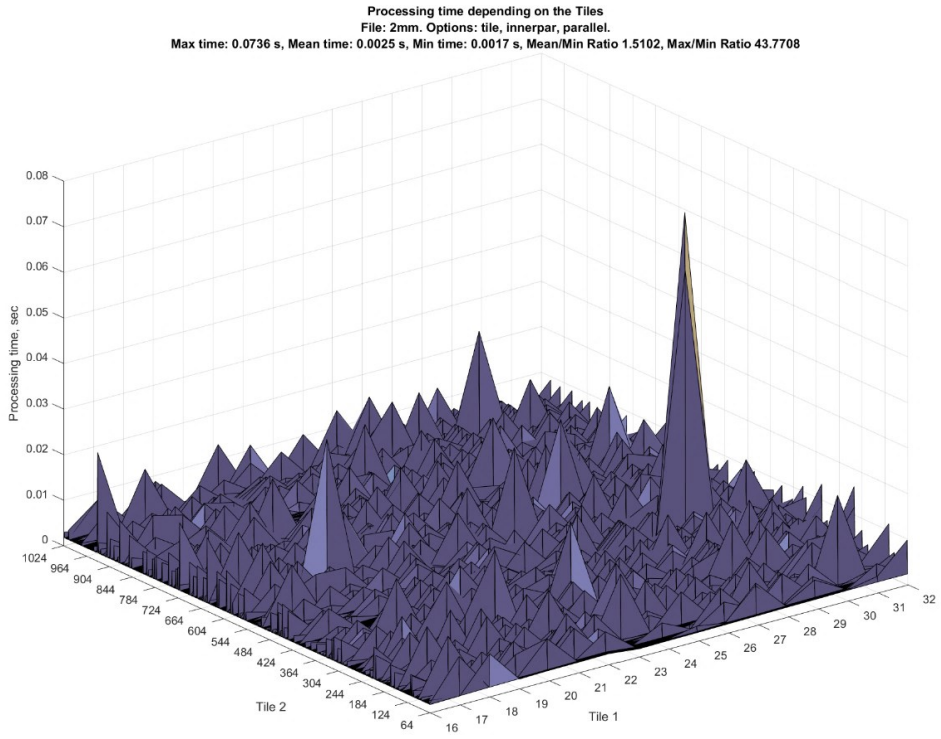

Рисунок В.17 – Залежність часу виконання від розмірів блоків розбиття для тестової програми 2mm

ДОДАТОК Г

Результати дослідження впливу параметрів ДМРЧ на характер поведінки часток

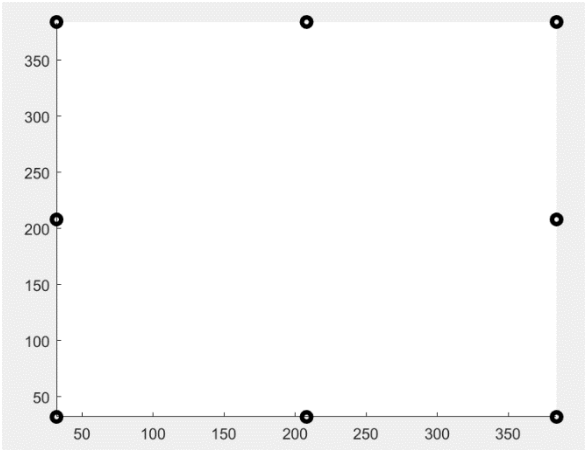

Рисунок Г.1 – Результати при w = 0, c1 = 0, c2 = 0. Частки не рухаються так як усі коефіцієнти дорівнюють нулю

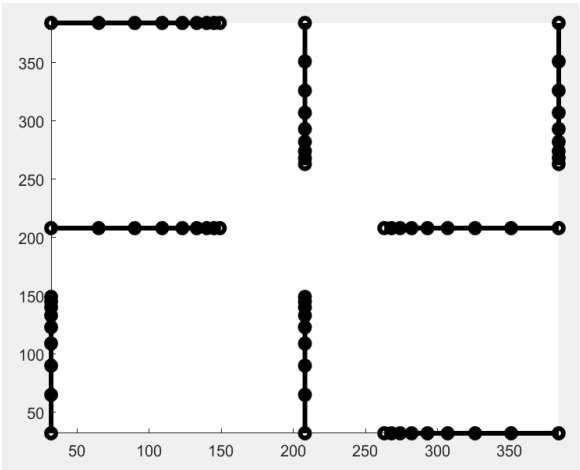

Рисунок Г.2 – Результати при w=0.75, c1 = 0, c2 = 0. Частки починають рух всередину області, але з часом швидкість руху сповільнюється оскільки немає не індивідуального ні соціального складника

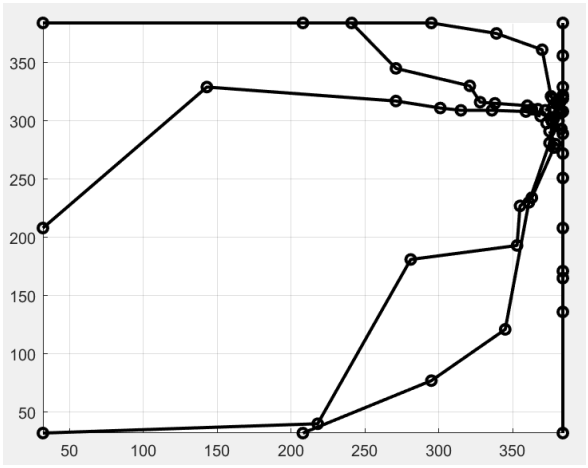

Рисунок Г.3 – Результати при w=0, c1 = 0, c2 = 0.75. На частки впливає тільки соціальний коефіцієнт

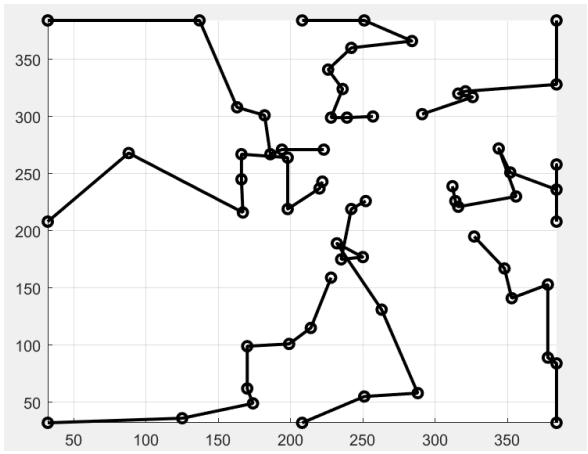

Рисунок Г.4 – Результати при w=0, c1 = 0, c2 = 0.375. На частки впливає тільки соціальний коефіцієнт, але він в 2 рази менший ніж на минулому зображенні

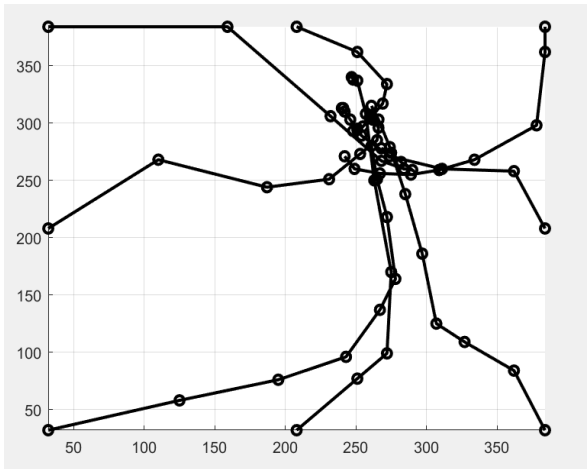

Рисунок Г.5 – Результати при w=0.5, c1 = 0, c2 = 0.375. Тільки початкове прискорення та соціальний фактор

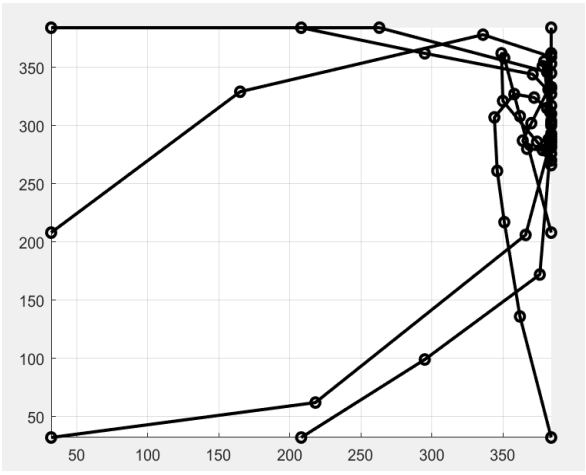

Рисунок Г.6 – Результати при w=0.5, c1 = 0, c2 = 0.75. Відносно минулого рисунку соціальний фактор подвоєно

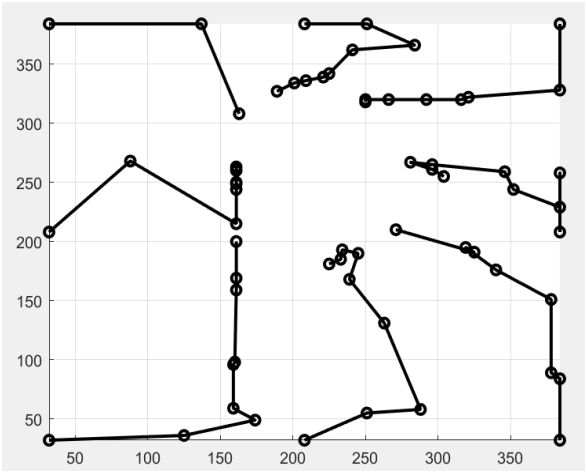

Рисунок Г.7 – Результати при w=0, c1 = 0.375, c2 = 0.375

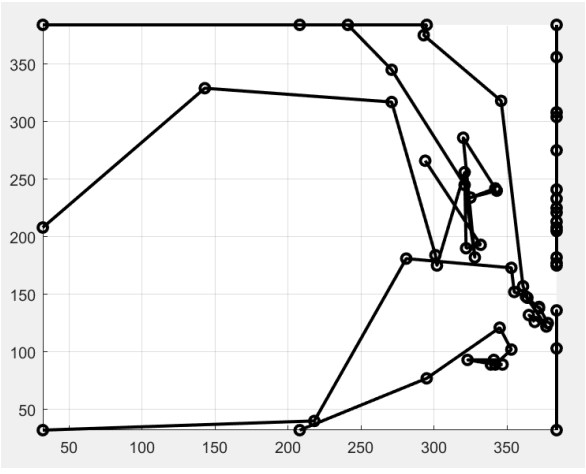

Рисунок Г.8 – Результати при w=0, c1 = 0.75, c2 = 0.75. Соціальний і індивідуальний коефіцієнти подвоєно.

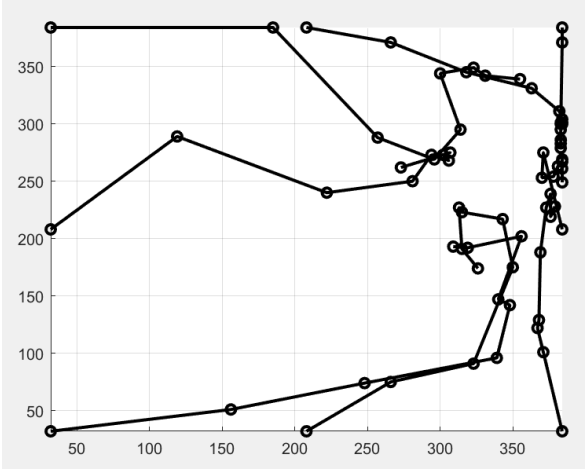

Рисунок Г.9 – Результати при w=0.3, c1 = 0.5, c2 = 0.5

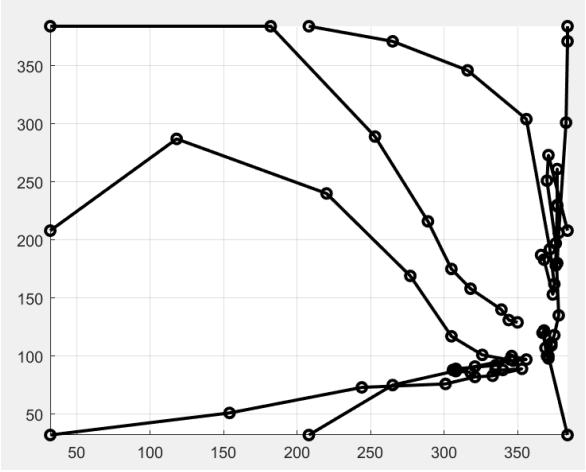

Рисунок Г.10 – Результати при w=0.3, c1 = 0.5, c2 = 0.49. Відносно минулого рисунка соціальний коефіцієнт зменшено на 0.01

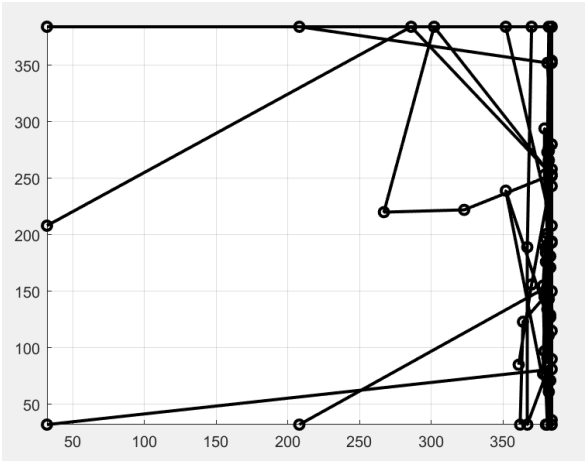

Рисунок Г.11 – Результати при w=0.7298, c1 = 1.49618, c2 = 1.49618. Застосовано значення коефіцієнтів, згідно рекомендацій для іншого призначення

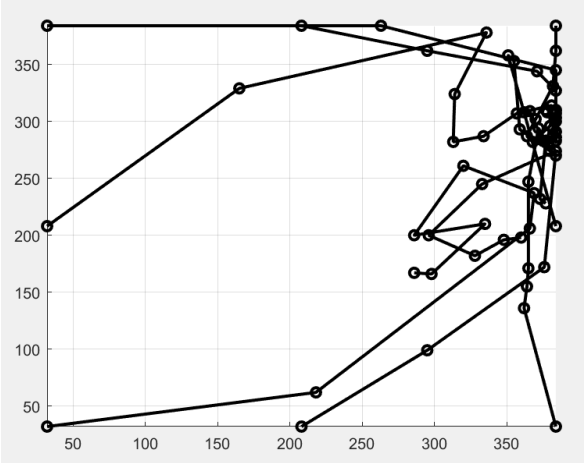

Рисунок Г.12 Результати при w=0.5, c1 = 0.75, c2 = 0.75

ДОДАТОК Ґ

Програмні коди, що реалізують запропонований метод інтелектуального блочного розбиття мовою Matlab

Програмний код головної функції запропонованого методу, файл SmartTiling.m

```
function [] = SmartTiling()

    format compact
    format shortg

    clc;
    close all;
    disp([' '])
    disp(['Метод інтелектуального блочного розбиття, Сушко С.В. (c), 2021'])
    disp(['Smart Tiling Method, Sushko S. (c), 2021'])
    disp([' '])

    % Початкова ініціалізація коефіцієнтів методу:
    config.w            = 0.5;  % Inertial coefficient
    config.self_coeff   = 0.75; % SelfAdjustmentWeight
    config.social_coeff = 0.75; % SocialAdjustmentWeight
    config.verbose      = 0;    % Verbose computations

    prompt = 'Please enter number of particles (4, 8, 16, ...): ';
    config.particles  = input(prompt);
    prompt = 'Please enter maximum number of iterations: ';
    config.max_iterations = input(prompt);
    prompt = 'Please enter low bound for tile size 1 (8, 16, 32, ...): ';
```

```
    config.tile1_low  = input(prompt);
    prompt = 'Please enter high bound for tile size 1 (64, 128, 256, ...): ';
    config.tile1_high = input(prompt);
    prompt = 'Please enter low bound for tile size 2 (8, 16, 32, ...): ';
    config.tile2_low  = input(prompt);
    prompt = 'Please enter high bound for tile size 2 (64, 128, 256, ...): ';
    config.tile2_high = input(prompt);
    for i = 1:config.max_iterations
        config.iteration = i;
        config = DPSOinterpEdges(config);
    end
end
```

Програмний код функції початкового розміщення часток на границі області пошуку, файл edge_range_2D.m

```
function [ v, x ] = edge_range_2D( x1lb, x1ub, x2lb, x2ub, N )

% Generates random values from range
x = zeros(2, N); % coordinates
v = zeros(2, N); % velocities

N_4 = N/4;

% Arragment of particles on the edges
x(1, 1:N_4)          = x1lb;
x(1, N_4 + 1:2*N_4)   = x1lb + (x1ub – x1lb).*(1:N_4)/N_4;
x(1, 2*N_4 + 1:3*N_4) = x1ub;
x(1, 3*N_4 + 1:N)     = x1lb + (x1ub – x1lb).*(N_4-1:-1:0)/N_4;

x(2, 1:N_4)          = x2lb + (x2ub – x2lb).*(1:N_4)/N_4;
```

```
x(2, N_4 + 1:2*N_4)   = x2ub;
x(2, 2*N_4 + 1:3*N_4) = x2lb + (x2ub – x2lb).*(N_4-1:-1:0)/N_4;
x(2, 3*N_4 + 1:N)     = x2lb;

% Setup of velocities
v(1, 1:N_4)           = (x1ub – x1lb)./8;
v(1, N_4 + 1:2*N_4)   = 0;
v(1, 2*N_4 + 1:3*N_4) = -(x1ub – x1lb)./8;
v(1, 3*N_4 + 1:N)     = 0;

v(2, 1:N_4)           = 0;
v(2, N_4 + 1:2*N_4)   = -(x2ub – x2lb)./8;
v(2, 2*N_4 + 1:3*N_4) = 0;
v(2, 3*N_4 + 1:N)     = (x2ub – x2lb)./8;

end
```

Програмний код функції квантування та обмеження позицій часток, файл saturate_inputs.m

```
function [ quant_x ] = saturate_inputs( x, xvalues, yvalues )
% Saturate and quantize values in the range

for i=1:length(x)
   x1diff(i, :) = abs(x(1, i) – xvalues);
   x2diff(i, :) = abs(x(2, i) – yvalues);
end

[v1, ind1] = min(x1diff(:,:)');
[v2, ind2] = min(x2diff(:,:)');
```

```
quant_x(1,:) = xvalues(ind1);
quant_x(2,:) = yvalues(ind2);

end
```

Програмний код функції початкового розміщення часток на границі області пошуку, файл DPSOinterpEdges.m

```
function [config] = DPSOinterpEdges(config)

% Partical Swarm Optimization Algorithm parameters
self_coeff   = config.self_coeff;   % SelfAdjustmentWeight
social_coeff = config.social_coeff; % SocialAdjustmentWeight
particles    = config.particles;
iteration    = config.iteration;
verbose      = config.verbose;

xmin = config.tile1_low; % low bound
xmax = config.tile1_high; % hi bound

ymin = config.tile2_low;  % low bound
ymax = config.tile2_high; % hi bound

if (iteration == 1)
   % Initial setup
   rng('default');

   disp(['Iteration 1']);

   % w = config.w; % Inertial coefficient
   config.v  = zeros(2, particles); % Velocities
```

```
config.x  = zeros(2, particles); % Positions
config.p  = zeros(2, particles); % Local minimum tiles' values
config.fp = Inf*ones(1, particles); % Time of local minimums tiles' values
config.g  = Inf;                % Global minimum tiles
config.fg = Inf;                % Time of minimun tiles

% Start points initialization on the edges
% Quantization in the interpolated grid
[config.v, config.x] = edge_range_2D( xmin, xmax, ymin, ymax, particles );

% Local minimums initialization
if (verbose > 0)
   disp(['Initial random x: ']);
   config.x
end

for i = 1:particles
   prompt = ['Tile sizes to check are (' num2str(config.x(1, i)) ', ' num2str(config.x(2,
i)) '). Please enter time of computation with these tile sizes in seconds: '];
   processing_time = input(prompt);
   if (processing_time < config.fp(i))
      config.fp(i) = processing_time;
      config.p(:, i) = config.x(:, i);
   end
   if (processing_time < config.fg)
      config.fg = processing_time;
      config.g  = config.x(:, i);
   end
end
```

```
    info_text = ['Global minimum so far is in (' num2str(config.g(1)) ', '
num2str(config.g(2)) '), time is: ' num2str(config.fg) ' sec'];
    disp(info_text);
else

    if (verbose > 0)
        disp([newline 'Iteration: ' num2str(iteration)]);
    end

    w  = config.w;   % Inertial coefficient
    v  = config.v;   % Velocities
    x  = config.x;   % Positions
    p  = config.p;   % Local minimum tiles' values
    fp = config.fp;  % Time of local minimums tiles' values
    g  = config.g;   % Global minimum tiles
    fg = config.fg;  % Time of minimun tiles

    % Creation of the random social and individual coefficients
    u1 = rand(2, particles);
    u2 = rand(2, particles);

    % Computation of the velocity change
    v = (w * v) + self_coeff * u1.*(p – x) + social_coeff * u2.*(g – x);

    if (verbose > 0)
        disp(['Velocity: ']);
        v
    end

    % Updating of the coordinates
```

```
x = x + v;

if (verbose > 0)
   disp(['Modified x: ']);
   x
end

% Saturation of the coordinates in the allowed range
% Quantization in the interpolated grid
quant_x = saturate_inputs( x, xmin:xmax, ymin:ymax);

% Update velocities with respect to the quantized inputs
config.v = v + quant_x – x;

if (verbose > 0)
   disp(['Velocity quantized: ']);
   config.v
end

% Update the input with the quantized inputs
config.x = quant_x;

if (verbose > 0)
   disp(['Saturated x: ']);
   config.x
end

for i = 1:particles
   prompt = ['Tile sizes to check are (' num2str(config.x(1, i)) ', ' num2str(config.x(2,
i)) '). Please enter time of computation with these tile sizes in seconds: '];
```

```
        processing_time = input(prompt);
        if (processing_time < config.fp(i))
            config.fp(i) = processing_time;
            config.p(:, i) = config.x(:, i);
        end
        if (processing_time < config.fg)
            config.fg = processing_time;
            config.g  = config.x(:, i);
        end
    end
    info_text = ['Global minimum so far is in (' num2str(config.g(1)) ', '
num2str(config.g(2)) '), time is: ' num2str(config.fg) ' sec'];
    disp(info_text);
end
end
```

ДОДАТОК Д

Акт впровадження

ДОВІДКА

про впровадження результатів дисертаційного дослідження
Сушка Сергія Володимировича
«Методи оптимального розпаралелювання програм мікропроцесорних систем для підвищення їх ефективності» за спеціальністю 05.13.05 – комп'ютерні системи та компоненти

Довідку надано здобувачеві наукового ступеня кандидата технічних наук Сушку Сергію Володимировичу, яка посвідчує, що результати його досліджень, які проводились протягом 2015-2020 років, знайшли практичне впровадження та використовуються в роботі ТОВ «Дельта СПЕ». Результати роботи впроваджено в програмний комплекс обробки даних супутникових каналів зв'язку з метою прискорення швидкодії.

Запропонований метод інтелектуального блочного розбиття та його практична реалізація дозволяють прискорити виконання комп'ютерних програм. Використання методу доцільне в обчислювальних системах, в яких час виконання комп'ютерних програм є обмеженим.

Загалом наукова робота Сушка С. В. та практичні результати його дослідження отримали позитивну оцінку колективу ТОВ «Дельта СПЕ».

Доктор фізико-математичних наук,
Завідуючий науково-дослідним відділом алгоритмів
ТОВ «Дельта СПЕ» Семенов В.Ю.

West Pomeranian University of Technology in Szczecin

al. Piastów 17, 70-310 Szczecin

TO WHOM IT MAY CONCERN:

This is to certify that optimization results provided by Sergii Sushko in his investigation titled “Methods of optimal parallelization of programs of microprocessor systems to increase of efficiency” are new scientific and practical results for design parallel computer programs and accelerating existing ones. His Smart Tiling Method helps to find quickly an optimal solution for a tile size and to decrease program parallelization time.

I am the leader of the TRACO project that is the part of research work of the Department of Software Engineering in West Pomeranian University of Technology. The goal of the project is to develop the TRACO compiler that parallelizes and optimizes the source code automatically.

In our plans is to engage the Smart Tiling Method into the TRACO project that allows us to enhance its possibilities aimed at automatic program parallelization.

Sincerely,