НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Кваліфікаційна наукова
праця на правах рукопису

Антонів Володимир Ярославович

УДК 004.424.4+004.424.5

ДИСЕРТАЦІЯ

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПАРАЛЕЛЬНОГО СОРТУВАННЯ ТА ПОШУКУ ДАНИХ

05.13.06 – 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 _____ Антонів В. Я.

Науковий керівник – Цмоць Іван Григорович, доктор технічних наук, професор

Львів – 2021

Львів – 2021

АНОТАЦІЯ

Антонів Володимир Ярославович. Інформаційні технології паралельного сортування та пошуку даних. – Кваліфікаційна наукова праця на правах рукопису.

Дисертація на здобуття наукового ступеня кандидата технічних наук за спеціальністю 05.13.06 – інформаційні технології. – Національний університет «Львівська політехніка», Міністерство освіти і науки України, Львів, 2021.

Дисертаційне дослідження присвячено розробленню нових та удосконаленню існуючих методів, моделей та засобів інформаційних технологій паралельного сортування та пошуку даних у реальному часі з високою ефективністю використання обладнання.

У *вступі* обґрунтовано актуальність теми дисертаційного дослідження; описано зв'язок роботи з науковими програмами, планами, темами; сформульовано мету та основні завдання дисертаційної роботи; представлено методи дослідження та визначено наукову новизну та відображено практичне значення одержаних результатів дослідження; презентовано списки опублікованих праць за тематикою дисертаційної роботи та конференцій, на котрих було апробовано основні результати дисертаційної роботи.

У *першому розділі* дисертаційної роботи проаналізовано галузі застосування, методи, моделі та засоби інформаційних технологій сортування та пошуку даних. Визначено, що в значній частині галузей застосування інформаційних технологій сортування та пошуку даних вимагається сортування та пошук максимальних і мінімальних значень у реальному часі при інтенсивному паралельному надходженні потоків даних.

Проаналізовано багаторівневі автоматизовані системи управління технологічними процесами, енергоефективністю тощо, на нижніх рівнях яких виконують збирання та попередню обробку технологічних даних. Основними операціями попередньої обробки є сортування та пошук даних, які необхідно виконувати в реальному часі. Результати сортування та пошуку даних передаються

на вищі рівні, де здійснюють їхнє накопичення та попереднє інтелектуальне опрацювання.

Проаналізовано та оцінено складність методів і алгоритмів сортування та пошуку даних та визначено, що їхня реалізація вимагає великої кількості операцій попарного порівняння та перестановки даних і у більшості випадків має квадратичну залежність. Показано, що більшість методів і алгоритмів сортування відрізняються вибором даних для попарного порівняння. Визначено, що для паралельної реалізації алгоритмів сортування та пошуку даних вони повинні бути добре структурованими з детермінованим переміщенням даних, ґрунтуватися на однотипних операціях із регулярними та локальними зв'язками.

Проаналізовано сучасну елементну базу та засоби розробки, які використовуються для створення програмно-апаратних засобів інформаційних технологій сортування та пошук даних у реальному часі. Запропоновано для виконання сортування та пошуку даних використовувати графічні процесори (GPU – Graphics Processing Unit), які є процесорами класу SIMD (Single Instruction Multiple Data) та програмовані логічні інтегральні схеми (ПЛІС). Особливістю процесорів класу SIMD є те, що в них одну операцію використовують одночасно для опрацювання множини незалежних даних. Для розроблення програмного забезпечення сортування великих масивів даних, частина якого працює на CPU (центральний процесор), а частина на GPU, доцільно використати кросплатформову систему компіляції та виконання програм CUDA. Показано, що основними перевагами використання ПЛІС є: доступність, висока швидкодія, надійність, універсальність, різноманітність під час вибору напруг живлення і параметрів сигналів вводу/виводу, низьке енергоспоживання, наявність засобів автоматизованого проєктування, які зменшують час проєктування, налаштування та відлагодження апаратних засобів. Суттєвими особливостями останніх поколінь ПЛІС є забезпечення можливості часткової реконфігурації апаратних засобів у процесі роботи. Показано, що для апаратної реалізації алгоритмів сортування та пошуку даних однією із вимог є мінімізація кількості виводів інтерфейсу. Проаналізовано мови опису апаратури (VHDL, HDL, Verilog HDL, AHDL), які використовують для

проєктування та моделювання структур апаратних засобів сортування та пошуку даних. Мова опису апаратури дає можливість автоматично аналізувати, імітувати та тестувати розроблені пристрої сортування та пошуку даних.

На основі проведеного аналізу визначено перелік задач і досліджень, які необхідно виконати для вирішення наукового завдання, поставленого у дисертаційній роботі.

У *другому розділі* сформульовано вимоги до засобів інформаційних технологій паралельного сортування та пошуку даних, основною з яких є забезпечення реального часу. Визначено, що алгоритми паралельного сортування та пошуку даних у реальному часі повинні бути: структурованими з детермінованим переміщенням даних; ґрунтуватися на однотипних операціях попарного порівняння та перестановки даних з регулярними та локальними зв'язками; орієнтованими на реалізацію на множині взаємозв'язаних процесорних ядр; використовувати конвеєризацію та просторовий паралелізм.

Вдосконалено та орієнтовано методи сортування чисел вставкою та злиттям на паралельне та паралельно-потокове сортування одновимірних масивів. Також в другому розділі було розроблено функціональні моделі алгоритмів паралельного сортування потоків даних у реальному часі. Вхідними даними таких моделей є інтенсивності надходження даних, розміри масиву, методи сортування та засоби реалізації. Функціональні моделі алгоритмів сортування завдяки знаходженню просторово-часових співвідношень забезпечують отримання паралельного конкретизованого графа для виконання сортування потоків даних у реальному часі. Процес розроблення функціональної моделі паралельного сортування масиву розбивається на такі чотири етапи: декомпозиція алгоритму сортування масиву; проектування комунікацій між функціональними операторами; укрупнення функціональних операторів; планування процесу сортування даних. Розроблені функціональні моделі паралельного сортування даних методом злиття та вставки, які внаслідок використання механізму управління паралелізмом забезпечують отримання конкретизованого графу алгоритму, орієнтованого на сортування у реальному часі на заданих засобах.

У *третьому розділі* вдосконалено та орієнтовано методи вставки і злиття на паралельно-вертикальне сортування даних у реальному часі.

Розроблено метод паралельно-вертикального пошуку максимальних і мінімальних чисел у масивах, який внаслідок паралельного опрацювання *i*-го розрядного зрізу масиву чисел і паралельного формування слів управління забезпечує зменшення часу пошуку, який визначається в основному розрядністю чисел та метод паралельно-вертикального сортування даних, який завдяки підрахунку одиниць у *i*-му вхідному розрядному зрізі та паралельному формуванню *i*-го розрядного зрізу відсортованого масиву чисел забезпечує зменшення часу сортування. Також у третьому розділі для кожного розробленого методу були створені функціональні моделі.

У *четвертому розділі* розроблено інформаційну технологію паралельного сортування даних, яка ґрунтується на інтегрованому підході, який охопив: дослідження, розроблення методів і алгоритмів паралельного сортування масивів даних; розроблення функціональних моделей алгоритмів паралельного сортування потоків даних у реальному часі; сучасну елементну базу (GPU, ПЛІС) та засоби автоматизованого проектування; нові архітектурні рішення, орієнтовані на технології надвеликих інтегральних схем. Реалізована інформаційна технологія паралельного сортування даних ґрунтується на розроблених і вдосконалених методах сортування даних, функціональних моделях і враховує інтенсивність надходження даних, розміри масивів даних, особливості засобів реалізації та забезпечує сортування даних у реальному часі з високою ефективністю використання обладнання. Також було розроблено інформаційну технологію паралельного пошуку даних, основними компонентами якої є: засоби збирання та передавання даних розрядними зрізами; розроблені методи паралельно-вертикального пошуку максимальних і мінімальних чисел; функціональні графові моделі пошуку у реальному часі максимальних і мінімальних чисел у масивах; засоби автоматизованого проєктування апаратного і програмного забезпечення; програмні та апаратні засоби пошуку даних у реальному часі з високою ефективністю використання обладнання.

Для розроблення апаратного і програмного забезпечення інформаційних технологій паралельного сортування і пошуку даних пропонується використовувати засоби автоматизованого проєктування: MAX+PLUS II або Quartus II, мови опису апаратури AHDL, VHDL, VERILOG HDL та кросплатформову систему компіляції та виконання програм CUDA. Для реалізації апаратних засобів пропонується використовувати ПЛІС фірми Altera, а програмні засоби розроблено для графічних процесорів фірми Nvidia. Використання розроблених засобів інформаційних технологій паралельного сортування та пошуку даних забезпечує виконання сортування і пошуку даних у реальному часі з високою ефективністю використання обладнання.

Ключові слова: інформаційна технологія; потоковий граф; паралельне сортування; метод злиття; метод вставки; алгоритми сортування; алгоритми пошуку; паралельний пошук; попарне порівняння; масив даних; графічний процесор, режим реального часу.

ABSTRACT

Volodymyr Antoniv. Information technologies of parallel sorting and data searching. – Qualifying scientific work on the rights of manuscripts

Thesis for obtaining the scientific degree of the candidate of technical sciences in the specialty 05.13.06 – information technologies. – Lviv Polytechnic National University, Ministry of Education and Science of Ukraine, Lviv, 2021.

Thesis is devoted to solving actual scientific problem – development new and improvement the existing methods, models and software and hardware of information technologies for parallel sorting and data searching in real-time with high efficiency of equipment.

In the *introduction* the relevance of the topic of the dissertation research has substantiated; the relationship with scientific programs, plans, topics has described; the purpose and tasks of the dissertation are formulated; methods of research have presented, and scientific novelty has determined when solving the set tasks. Also, the practical significance of the obtained research results is reflected; lists of published papers on the topic of dissertation work and conferences, where the main results of the dissertation were tested, are presented.

In the *first section* of the thesis were analyzed the fields of application, methods, models, and tools of information technology for data sorting and searching. It was found that a significant part of the fields of application of information technologies of data sorting and searching requires sorting and searching for the maximum and minimum values in real time with intensive parallel data flows.

Multilevel automated control systems of technological processes, energy efficiency, etc. are analyzed, at the lower levels of which the collection and pre-processing of technological data is performed. The main pre-processing operations are sorting and searching for data that needs to be performed in real time. The results of sorting and searching data are transmitted to higher levels, where they are accumulated and pre-processed.

The complexity of methods and algorithms for data sorting and searching is analyzed and evaluated, and it is found that their implementation requires many operations of pairwise matching and rearrangement of data and in most cases has a quadratic dependence. It is shown that most of the methods and algorithms of sorting differ in the choice of data for pairwise comparison. It was found that for parallel implementation of sorting and data search algorithms, they must be well structured with deterministic data shifting, run on the same type of operations with regular and local links.

The modern element base and development tools used for the creation of software and hardware of information technology sorting and searching in real-time are analyzed. It is required to use GPUs (Graphics Processing Unit), which are SIMD (Single Instruction Multiple Data) class processors and programmable logic devices (PLD) for data sorting and searching. A special feature of SIMD processors is that they use a single operation to process multiple independent data at the same time. For the development of software for sorting large amounts of data, part of which works on the CPU and part on the GPU, it is advisable to use the cross-platform system of compilation and execution programs CUDA. It is shown that the main advantages of using PLD are availability, high speed, reliability, versatility, and variety of options when selecting power supply voltages and input/output signal parameters, low energy consumption, availability of automated design features that reduce the time of design, configuration, and maintenance of hardware. The essential features of the latest generations of PLD are the possibility of partial reconfiguration of hardware during operation. It is shown that for hardware implementation of sorting and data searching algorithms one of the requirements is to minimize the number of interface outputs. Hardware description languages (VHDL, HDL, Verilog HDL, AHDL), which are used to design and model the structures of hardware sorting and data retrieval, are analyzed. The equipment description language makes it possible to automatically analyze, simulate and test the developed data sorting and retrieval devices.

Based on the conducted analysis the list of tasks and research that need to be carried out to solve the scientific task set in the dissertation work was determined.

In the *second section*, I formulated the requirements to information technology for parallel data sorting and searching, the main of which is to ensure real time. It is found that

the algorithms for parallel sorting and data searching in real-time must be: well-structured with deterministic data shifting; be based on the same type of pairwise comparison and permutation operations with regular and local connections; focused on the implementation of multiple interconnected processor cores; use pipelining and spatial parallelism.

The methods of sorting numbers by insertion and merging on parallel and parallel-stream sorting of one-dimensional arrays are improved and oriented. Also in the second section, functional models of algorithms for parallel sorting of real-time data streams were developed. The input data of such models are data intensities, array sizes, sorting methods and means of implementation. Functional models of sorting algorithms by finding spatio-temporal relations provide a parallel specified graph for sorting data streams in real time. The process of developing a functional model of parallel array sorting is divided into the following four stages: decomposition of the array sorting algorithm; design of communications between functional operators; consolidation of functional operators; planning the data sorting process. Functional models of parallel data sorting by merging and inserting methods have been developed, which, due to the use of the parallelism control mechanism, provide a concretized graph of the algorithm focused on real-time sorting on given means.

In the *third section*, the insertion and merging methods for parallel-vertical sorting of data in real time are improved.

Developed the method of parallel-vertical search of maximum and minimum numbers in arrays, which due to parallel evaluation of the *i*-th digit section of the array of numbers and parallel formation of control words provides reduction of search time, which is determined in the main by the ordering of the numbers. Developed the method of parallel-vertical sorting of data, which due to the count of the units in the *i*-th input bit array and parallel formation of the *i*-th bit array of the sorted array of numbers ensures reduced time of sorting. Also, in the third section functional models were created for each developed method.

In the *fourth section* was developed the information technology for parallel data sorting, which is based on the integrative approach, which includes: research, development of methods and algorithms for parallel sorting of data sets; development of functional

models of algorithms for parallel sorting of data streams in real time; modern element base (GPU, FPGA) and computer-aided design tools; new architectural solutions focused on the technology of ultra-large integrated circuits. Implemented information technology of parallel data sorting is based on developed and improved data sorting methods, functional models and takes into account data intensity, data array sizes, features of implementation tools and provides real-time data sorting with high efficiency of equipment use. Information technology of parallel data search was also developed, the main components of which are: means of data collection and transmission by bit slices; developed methods of parallel-vertical search of maximum and minimum numbers; functional graph models of real-time search of maximum and minimum numbers in arrays; means of automated design of hardware and software; software and hardware real-time data retrieval with high efficiency of equipment use.

For development of hardware and software of information technologies of parallel sorting and search of data it is offered to use means of automated designing: MAX + PLUS II or Quartus II, languages of the description of the AHDL, VHDL, VERILOG HDL equipment and cross-platform system of compilation and execution of CUDA programs. Altera FPGAs are proposed for hardware implementation, and software is designed for Nvidia GPUs. The use of developed information technology tools for parallel sorting and data searching provides real-time sorting and searching of data with high efficiency of equipment use.

Key words: information technology; flow graph; parallel sorting; merge method; insertion method; sorting algorithms; search algorithms; parallel searching; pairwise comparison; data array; graphics processor, real time mode.

СПИСОК ПУБЛІКАЦІЙ ЗДОВУВАЧА ЗА ТЕМОЮ ДИСЕРТАЦІЇ

1. Цмоць І. Г., Антонів В. Я. Вертикально-паралельний метод сортування масивів чисел // Збірник наукових праць “Моделювання та інформаційні технології”. Інститут проблем моделювання в енергетиці ім. Г. Є. Пухова. 2016, Випуск 77. С. 186 – 192.

2. Цмоць І. Г., Антонів В. Я., Рабик В. Г. Метод вертикально-паралельного обчислення максимальних і мінімальних чисел у масивах // Збірник наукових праць “Моделювання та інформаційні технології”. Інститут проблем моделювання в енергетиці ім. Г. Є. Пухова. 2016, Випуск 76. С. 190 – 196.

3. Цмоць І. Г., Антонів В. Я. Удосконалення паралельного сортування масивів чисел методом злиття // Науковий вісник НЛТУ України : збірник наукових праць. Львів, 2020, Т. 30, № 4. С. 134 – 142.

4. Ivan Tsmots, Oleksa Skorokhoda, Volodymyr Antoniv. Parallel algorithms and structures for implementation of merge sort // International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 5 Issue 3, March 2016. Pp. 798 – 807.

5. Цмоць І. Г., Скорохода О. В., Красовський В. Б., Антонів В. Я. Методи та НВІС-структури узгоджено-паралельного обчислення максимальних і мінімальних значень // Збірник наукових праць інституту проблем моделювання в енергетиці ім. Г. Є. Пухова. 2014, Випуск 70. С.38 – 48.

6. Цмоць І. Г., Парубчак В. О., Антонів В. Я. Паралельно-вертикальне сортуванням одновимірних масивів даних методом злиття з використанням підрахунку // Збірник наукових праць інституту проблем моделювання в енергетиці ім. Г. Є. Пухова. 2013, Випуск 68. С.92 – 100.

7. Цмоць І. Г., Антонів В. Я. Апаратні засоби сортування даних методом злиття в реальному часі // Вісник національного університету “Львівська політехніка” Збірник наукових праць, 2015, № 814. - С. 171 – 186.

8. Цмоць І. Г., Кісь Я. П., Антонів В. Я. Застосування графічного процесора для підвищення швидкодії процесу сортування великих масивів даних // Науковий вісник

НЛТУ України: Збірник науково-технічних праць. – Львів : РВВ НЛТУ України. – 2015. – Вип. 25.6. – С. 328 – 334.

9. Ivan Tsmots, Oleksa Skorokhoda, Vasyl Rabyk, Volodymyr Antoniv Vertically-Parallel Method and VLSI-Structure sorting of Arrays of Numbers // Advances in Intelligent Systems and Computing III. Springer International Publishing AG 2019. Pp.267 – 284.

10.Цмоць І. Г., Антонів В. Я. Алгоритми та паралельні структури сортування даних методом вставки // Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів : РВВ НЛТУ України. – 2016. – Вип. 26.1. – С. 340 – 350.

11.Ivan Tsmots, Oleksandr Kuzmin, Vasyl Dubuk, Volodymyr Antoniv Improvement and orientation of method of data arrays sorting by confluence on architecture of graphic processor unit // Advances in Intelligent Systems and Computing V. International Conference on Computer Science and Information Technologies, CSIT 2020, September 23-26, 2020, Zbarazh, Ukraine. Pp. 276 – 286.

12.Цмоць І. Г., Антонів В. Я. Метод розробки апаратних засобів паралельно-узгодженого сортування масивів даних у реальному часі // Матеріали міжнародної конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту» ISDMCI 2015, 25–28 травня. — Залізний Порт, 2015. – С.221 – 222.

13.Парубчак В. О., Антонів В. Я. Методи паралельного сортування одновимірних масивів даних // Матеріали міжнародної конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту» ISDMCI 2014, 28-31 травня. — Залізний Порт, 2014. – С.23 – 24.

14.Цмоць І. Г., Антонів В. Я. Застосування графічного процесора для підвищення швидкодії сортування великих масивів даних // Матеріали XIV Міжнародного наукового семінару «Сучасні проблеми інформатики в управлінні, економіці, освіті» , Світязь. – 2015. – С.90 – 92.

15.Цмоць І. Г., Антонів В. Я. Методи та апаратно-програмні засоби паралельно-вертикального сортування масивів чисел // Збірник наукових праць міжнародної наукової конференції “Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту ISDMCI’2016”. – Залізний Порт, 2016. – С. 226 – 227.

16.Tsmots I., Rabyk V., Skorokhoda O., Antoniv V. FPGA implementation of vertically parallel minimum and maximum values determination in array of numbers // 2017 14th International Conference The Experience of Desingning and Application of CAD Systems in Microelectronics (CADSM). PROCEEDINGS. Polyana. February 21-25, 2017. Pp. 234 – 236.

17.Tsmots I., Skorokhoda O., Antoniv V., Rabyk V. Vertically-Parallel Method and VLSI-Structure for Sorting of One-Dimensional Arrays // Computer Sciences and Information Technologies. In Proceedings of 13th International Scientific and Technical Conference CSIT 2018. Pp. 112 – 116.

18.Цмоць І. Г., Скорохода О. В., Медиковський М. О., Антонів В. Я. Пристрій для визначення максимального числа з групи чисел. Патент України на винахід №110187, 25.11.2015, Бюл. №22.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CPU	Central processing unit
CUDA	Compute Unified Device Architecture
FPGA	Field-Programmable Gate Array
GPU	Graphics proccesing unit
GPGPU	General-purpose computing on graphics processing units
OpenCL	Open Computing Language
OLTP	Online Transaction Processing
SIMD	Single instruction, multiple data
SSAS	SQL Server Analysis Services
SSIS	SQL Server Integration Services
SSRS	SQL Server Reporting Services
БД	База даних
ІАД	Інтелектуальний аналіз даних
ІТ	Інформаційна технологія
КН	Комп’ютерні науки
СППР	Система підтримки прийняття рішень
СД	Сховище даних
П-ВСД	Паралельно-вертикальне сортування даних
П-ВП	Паралельно-вертикальний пошук
ПЕ	Процесорний елемент
ПЛІС	Програмована логічна інтегральна схема
НВІС	Надвелика інтегральна схема
ЯПФ	Ярусно-паралельна форма

ВСТУП

Актуальність теми.

Сучасний етап розвитку інформаційних технологій сортування та пошуку даних характеризується розширенням галузей їхнього застосування, значна частина з яких пов’язана з накопиченням, сортуванням і пошуком даних у реальному часі. До таких застосувань належать системи збирання та попередньої обробки телеметричних даних, управління складними об’єктами, автоматизовані системи багаторівневого управління технологічними процесами, де на нижніх рівнях накопичуються великі обсяги даних, які потребують попереднього опрацювання у реальному часі. При попередньому опрацюванні масивів даних найчастіше доводиться використовувати операції сортування та пошуку даних, які можуть займати до 40% із загального часу роботи з базами даних. Для ефективного опрацювання потоків даних необхідні інформаційні технології паралельного сортування та пошуку даних у реальному часі, що ґрунтуються на нових і вдосконалених методах, моделях, які повинні бути орієнтовані на паралельно-потокове надходження даних і на адаптацію до інтенсивності надходження даних. Ефективне сортування та пошук даних у реальному часі вимагає розроблення нових методів і алгоритмів, які орієнтовані на сучасну елементну базу (графічні процесори та програмовані логічні інтегральні схеми).

Для сортування і пошуку максимальних та мінімальних значень у реальному часі необхідне створення спеціалізованих апаратних засобів із високою ефективністю використання обладнання. Забезпечити такі вимоги можна розробленням методів паралельно-вертикального сортування і пошуку даних, особливістю яких є надходження даних паралельно-розрядними зрізами, використання однотипних процесорних елементів з регулярними та локальними зв’язками, суміщення у часі процесів приймання та опрацювання даних.

При створенні ефективних програмно-апаратних засобів інформаційних технологій сортування та пошуку максимальних і мінімальних значень у реальному часі необхідно розробити просторово-часові моделі з механізмами узгодження

інтенсивності надходження даних з інтенсивністю виконання сортування та пошуку програмно-апаратними засобами. Процес переходу від моделей до структури апаратних засобів є складним і вимагає взаємної адаптації алгоритмів і структур.

З наведеного випливає, що розробляти інформаційні технології паралельного сортування та пошуку максимальних і мінімальних значень у реальному часі необхідно за інтегрованим підходом, який охоплює методи, моделі та засоби сортування та пошуку даних, методи їх розпаралелення, сучасну елементну базу, засоби автоматизованого проєктування та автоматизації процесу програмування.

Розроблені у цій роботі інформаційні технології паралельного сортування та пошуку максимальних і мінімальних значень у реальному часі ґрунтуються на напрацюваннях відомих зарубіжних та українських вчених, які створили теоретичні та практичні засади їх побудови, таких як Г. Лорин, Р. Седжвик, Т. Б. Мартинюк, В. В. Пасічник, В. П. Гергель, К. Кушері, П. Пушнер, Д. Валенсія, М. Возняк та інші.

Отже, актуальним науковим завданням є розроблення нових та удосконалення існуючих методів, моделей і засобів інформаційних технологій паралельного сортування та пошуку даних у реальному часі з високою ефективністю використання обладнання.

Зв'язок роботи з науковими програмами, планами, темами.

Дисертаційне дослідження проводили згідно з планами науково-дослідних та навчальних робіт кафедри автоматизованих систем управління Національного університету «Львівська політехніка», в тому числі в межах держбюджетних науково-дослідницьких робіт:

- «Розвиток теорії синтезу нейронних мереж на НВІС-структурах для обробки сигналів у робототехнічних системах» (номер держ. реєстр. 0112U001204);
- «Інструментальні засоби та інтелектуальні компоненти синтезу інтегрованих автоматизованих систем управління» (номер держ. реєстр. 0113U003186);
- «Інтелектуальні інформаційні технології багаторівневого управління енергоефективністю регіону» (номер держ. реєстр. 0117U004450).

Результати дисертаційної роботи використано під час розроблення засобів збирання та попереднього опрацювання телеметричних даних на ДП “Львівський державний завод “ЛОРТА” та впроваджено у навчальний процес кафедри автоматизованих систем управління Національного університету “Львівська політехніка”.

Мета і завдання дослідження.

Метою дисертаційної роботи є розроблення нових та удосконалення існуючих методів, моделей і засобів інформаційних технологій паралельного сортування і пошуку даних у реальному часі з високою ефективністю використання обладнання.

Об'єктом дослідження є процеси сортування та пошуку даних у реальному часі.

Предметом дослідження є методи, моделі, алгоритми і засоби інформаційних технологій паралельного сортування та пошуку даних у реальному часі.

Завдання дослідження. Досягнення поставленої мети передбачало вирішення таких завдань:

- аналіз інформаційних технологій сортування та пошуку даних і визначення напрямів їх розвитку;
- розроблення методу паралельно-вертикального сортування даних;
- розроблення методу паралельно-вертикального пошуку максимальних і мінімальних чисел у масивах;
- удосконалення методу паралельно-потокового сортування даних злиттям;
- розроблення інформаційної технології паралельного сортування даних;
- розроблення інформаційної технології паралельного пошуку даних;
- розроблення засобів інформаційних технологій паралельного сортування та пошуку даних.

Методи дослідження.

У дисертаційній роботі використано: методи паралельного сортування – для розроблення апаратно-програмних засобів сортування масивів даних; методи

паралельного пошуку максимальних і мінімальних значень – для розроблення апаратно-програмних засобів пошуку максимальних і мінімальних значень; елементи теорії графів, теорію та методи розпаралелення алгоритмів, теорію проєктування та синтезу програмно-апаратних засобів – для розроблення паралельних програмно-апаратних засобів сортування та пошуку даних.

Наукова новизна отриманих результатів.

За результатами виконаних теоретичних та експериментальних досліджень розв’язано актуальну наукову задачу – розроблено нові та вдосконалено існуючі методи, моделі і засоби інформаційних технологій паралельного сортування та пошуку даних у реальному часі з високою ефективністю використання обладнання. При цьому отримано такі нові результати:

вперше розроблено:

- інформаційну технологію паралельного сортування даних, яка завдяки використанню розроблених і вдосконалених методів, функціональних моделей паралельно-потокового сортування даних та врахуванню інтенсивності надходження даних, розмірів масивів даних і засобів реалізації забезпечує виконання сортування даних у реальному часі з високою ефективністю використання обладнання;
- метод паралельно-вертикального пошуку максимальних і мінімальних чисел у масивах, який внаслідок паралельного опрацювання *i*-го розрядного зрізу масиву чисел і паралельного формування слів управління забезпечує зменшення часу пошуку, який визначається в основному розрядністю чисел;

вдосконалено:

- метод паралельного сортування злиттям, який завдяки використанню базової операції об'єднання двох масивів з одночасним формуванням елементів зростаючого і спадаючого масивів забезпечує зменшення часу сортування;
- метод паралельно-вертикального сортування даних, який завдяки підрахунку одиниць у *i*-му вхідному розрядному зрізі та паралельному формуванню *i*-го

розрядного зрізу відсортованого масиву чисел забезпечує зменшення часу сортування.

Практичне значення отриманих результатів.

Практичне значення отриманих результатів полягає у розробленні інформаційних технологій паралельного сортування та пошуку даних у реальному часі з високою ефективністю використання обладнання, а саме:

1. Сортування одновимірних масивів даних на графічному процесорі GPU із використанням вдосконаленого методу паралельного сортування злиттям забезпечує зменшення часу сортування на 31%.
2. Використання процесу злиття двох масивів шляхом одночасного формування елементів зростаючого і спадаючого масивів забезпечує зменшення часу сортування вдвічі порівняно з послідовним злиттям.
3. Апаратна реалізація розробленого методу паралельно-вертикального пошуку максимальних і мінімальних чисел забезпечує просте модульне нарощування кількості входів і зменшення часу пошуку, який в основному визначається розрядністю чисел.
4. Використання методу паралельно-вертикального сортування даних під час розроблення апаратних засобів забезпечує їх реалізацію з високою ефективністю використання обладнання та сортування інтенсивних потоків даних у реальному часі.
5. Використання розроблених інформаційних технологій паралельного сортування та пошуку даних забезпечує виконання сортування та пошук даних у реальному часі з високою ефективністю використання обладнання.

Особистий внесок здобувача.

Усі наукові результати дисертаційної роботи автор отримав самостійно. У працях, опублікованих у співавторстві, здобувачеві належать: [1] – розроблено метод паралельно-вертикального сортування масивів чисел; [2] – розроблено потокові графи для методу паралельно-вертикального пошуку максимальних і мінімальних чисел; [3] – удосконалено метод сортування злиттям для підвищення швидкодії сортування; [4] – розроблено структури пристрою сортування даних методом злиття;

[5] – розроблено структури пристрою узгоджено-паралельного обчислення максимальних і мінімальних значень; [6] – розроблено метод паралельно-вертикального сортуванням одновимірних масивів даних злиттям; [7] – розроблено потоковий граф методу злиття та орієнтовано його на апаратну реалізацію; [8] – розроблено програмні рішення на графічному процесорі для підвищення швидкодії сортування; [9] – розроблено структури пристрою паралельно-вертикального сортування даних; [10] – розроблено структури пристрою сортування даних методом вставки; [11] – розроблено програмні засоби сортування на графічному процесорі; [12] – адаптовано метод паралельно-узгодженого сортування та апаратну реалізацію; [13] – розроблено метод паралельного сортування; [14] – реалізовано алгоритм сортування на графічному процесорі; [15] – адаптовано метод паралельно-вертикального сортування на графічний процесор; [16] – реалізовано метод пошуку мінімальних і максимальних чисел на НВІС; [17] – розроблено структури пристрою паралельно-вертикального сортування даних; [18] – розроблено структури процесорного елемента пристрою.

Апробація результатів дисертації.

Основні результати наукових досліджень неодноразово доповідалися, обговорювалися на міжнародних та всеукраїнських науково-технічних конференціях, зокрема на: Міжнародній конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту», ISDMCI 2014 (28-31 травня. — Залізний Порт, 2014); Міжнародній конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту», ISDMCI 2015 (25-28 травня. — Залізний Порт, 2015); XIV Міжнародному науковому семінарі «Сучасні проблеми інформатики в управлінні, економіці, освіті» (29 червня – 2 липня. — Світязь – 2015); Міжнародній конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту», ISDMCI 2016 (24-28 травня. — Залізний Порт, 2016); 14th International Conference «The Experience of Desingning and Application of CAD Systems in Microelectronics» CADSM (Polyana, February 21-25, 2017); XIIIth International Scientific and Technical Conference «Computer science and information

technologies», CSIT’2018 (Lviv, 11-14 September, 2018); XV International Scientific and Technical Conference «Computer Science and Information Technologies», CSIT’2020 (Lviv, Ukraine, 23-26 September, 2020).

Публікації.

За темою дисертації опубліковано 18 друкованих праць, серед яких 8 статей у наукових фахових виданнях України та 1 стаття у науковому періодичному виданні іншої держави, 8 тез доповідей на міжнародних конференціях та 1 патент України на винахід.

Структура та обсяг дисертації.

Дисертаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Основний зміст викладено на 206 сторінках друкованого тексту, включаючи 166 сторінок основного тексту, містить 67 рисунків, 8 таблиць. Список використаних джерел містить 150 найменування на 13 сторінках та 3 додатки на 13 сторінках.

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ СОРТУВАННЯ І ПОШУКУ ДАНИХ

1.1. Аналіз об’єкту дослідження

Для дослідження, моделювання та ефективного вирішення завдання розроблення нових та удосконалення існуючих методів, моделей і засобів інформаційних технологій паралельного сортування і пошуку даних у реальному часі з високою ефективністю використання обладнання необхідно здійснити аналіз об’єкта дослідження, існуючих інформаційних технологій сортування і пошуку, визначити галузі застосування та напрями їх розвитку.

1.1.1. Аналіз галузей застосування інформаційних технологій сортування і пошуку даних

Сучасний етап розвитку інформаційних технологій сортування та пошуку даних характеризується розширенням галузей їхнього застосування, значна частина з яких пов’язана з накопиченням великих масивів даних і паралельно-потоковим сортуванням і пошуком даних у реальному часі. До таких застосувань належать системи збирання та попередньої обробки телеметричних даних, управління складними об’єктами, автоматизовані системи багаторівневого управління технологічними процесами, енергоефективністю підприємств міст регіону тощо, на нижніх рівнях яких виконують збирання та попередня обробку технологічних даних. Основними операціями попередньої обробки є сортування та пошук даних, які необхідно виконувати в реальному часі. Результати сортування та пошуку даних передаються на вищі рівні, де здійснюють їхнє накопичення та попереднє інтелектуальне опрацювання. Тому для ефективного вирішення поставленого в цій роботі завдання, перш за все необхідно визначити поняття сортування і пошуку даних.

Згідно з загальноприйнятим визначенням [19], сортування даних – це процес розставляння елементів масиву у деякому визначеному порядку за попередньо заданим законом з метою їх подальшого ефективного аналізу. Можна сказати, що

задача сортування масивів даних $\{x_i\}_i^N$ полягає в отримані масивів даних $\{y_i\}_i^N$, які складаються із елементів даних x_i, переставлених в необхідному порядку (по спаданню або зростанню значень). До прикладу сортування даних у технологіях цифрової обробки сигналів використовується при формуванні вхідних (вихідних) потоків даних, виявленні та виділенні сигналів [38].

В свою чергу пошук – це операція знаходження неочевидного, втраченого, прихованого чи конкретного елемента (значення) з ключем K за певним вказаним аргументом. Результатом виконання операції пошуку може бути тільки один з двох варіантів: пошук закінчився успішно і запис із вказаним ключем було знайдено, або безуспішно, де запис не було знайдено [21]. Операція пошуку даних являється однією з найбільш часозатратних частин багатьох систем, до прикладу вони можуть займати до 40% із загального часу роботи у базах даних. Тому використання попередньо відсортованих даних при виконанні пошуку суттєво підвищує ефективність виконання алгоритмів. Оскільки при використанні відсортованих даних досить часто виникає можливість взагалі позбутися пошуку. Також ключовим моментом при розробці ІТ є вибір вдалих алгоритмів. Заміна невдалого алгоритму на більш вдалий, може значно підвищити швидкість роботи системи. Так можна розглянути ефективність виконання алгоритмів пошуку на прикладі бінарного пошуку. Цей алгоритм виконується швидше ніж лінійний пошук, але лише тоді коли, масив даних вже упорядкований, тобто відсортований [27].

1.1.2. Аналіз систем збирання та попереднього опрацювання сейсмічної інформації

Як вже було сказано, сучасний етап розвитку ІТ сортування та пошуку характеризується широким спектром галузей застосування, однією з таких галузей є системи збирання та попереднього опрацювання сейсмічної інформації. В системах сейсмічної розвідки збір корисної інформації із польових сейсмічних записів виконується в процесі їх обробки і інтерпретації. Від якості виконання даних операцій залежить повнота, надійність і точність отриманих геологічних результатів. Завдання попереднього опрацювання сейсмічної інформації – це подавлення регулярних та

нерегулярних хвиль-перешкод та виявлення кінематичних (час приходу хвилі) та динамічних (амплітуда сигналів) характеристик хвиль. Після попереднього опрацювання їх необхідно ідентифікувати однократними, відбитими або заломленими хвилями. Таким чином, в результаті опрацювання сейсмічних даних отримуються часи приходу на місця вимірювання різного роду хвиль, які знаходяться на різних відстанях від пункту збудження. За ними будуються годографи хвиль, профілограми, тимчасові зрізи.

В системах збирання та попереднього опрацювання сейсмічної інформації виділяють такі основні етапи [73]:

1. Етап фільтрації. Фільтрація служить для збільшення відношення сигнал/перешкода, тобто посилює амплітуду корисного сигналу і послабляє амплітуду перешкод.

2. Етап розрахунку і кореляції статичних поправок, які позволяють позбутися спотворення часу переходу хвиль за рахунок неоднорідності верхньої частини зрізу.

3. Етап розрахунку і кореляції динамічних поправок, які дозволяють позбутися розбіжності у часі приходу хвиль, утворених неоднорідністю відстані пунктів збору сигналів від пунктів збудження та нахилом відбиваючих границь.

Для ефективного виконання кожного з етапів попереднього опрацювання сигналів вимагається використання операцій сортування та пошуку даних. На етапі фільтрації доцільно використовувати метод паралельно-вертикального пошуку мінімальних і максимальних значень у реальному часі оскільки вимагається знаходження найбільших значень для посилення амплітуд, в свою чергу паралельно-вертикальне сортування на етапі розрахунку і кореляції.

1.1.3. Аналіз систем телеметрії

З визначення, телеметрія — це сукупність технічних засобів і методів вимірювання на відстані різних фізичних, технічних та інших величин у промислових, енергетичних, транспортних та інших установках. Передавання визначених даних з будь-якої точки на віддалений термінал [56].

Тому в системах телеметрії, автоматизований збір вимірювальних даних та їх передавання до приймального обладнання моніторингу здійснюється у реальному часі, що для систем даного типу є надзвичайно важливим завданням. Оскільки затримка може призвести до критичних пошкоджень на виробництві та при проведені різного роду випробувань, як: льотних випробуваннях, нафто-газовій індустрії, воєнних системах, медицині тощо.

Вимірюванні здійснюється за допомогою інтелектуальних сенсорів та давачів, отримана інформація автоматично передаються у вигляді кодованих радіо, інфрачервоних, ультразвукових, GSM, супутниковими чи кабельними сигналами старшими розрядами вперед на приймальні пристрої, де вони розшифровується і записується в бази даних. Найчастіше використовують давачі для виміру таких значень як тиск, навантаження, температура, прискорення, термопари, витрати і тп.

1.1.4. Аналіз автоматизованих систем управління технологічними процесами

В автоматизованих системах управління технологічними процесами [31], які є багаторівневими на нижніх рівнях виконується такі операції, як: збирання, попереднє опрацювання та збереження технологічних даних у вигляді бази даних. При попередньому опрацюванні технологічних даних значну частину часу займають операції сортування та пошуку даних. Інформація від давачів та інтелектуальних сенсорів передається з використанням послідовних інтерфейсів старшими розрядами вперед. Операції сортування та пошуку даних на нижньому рівні у більшості випадків необхідно виконувати у реальному часі. Структура автоматизованої системи багаторівневого управління технологічними процесами наведена на рис.1.1, де П-ВСД – паралельно-вертикальне сортування даних; ПВП*max/min* – паралельно-вертикальний пошук максимальних і мінімальних значень; ПК – персональний комп’ютер; TCP/IP – стек протоколу обміну; ІС – інтелектуальний сенсор; Д – давач; ВМ –виконавчий механізм; МК – мікроконтролер.

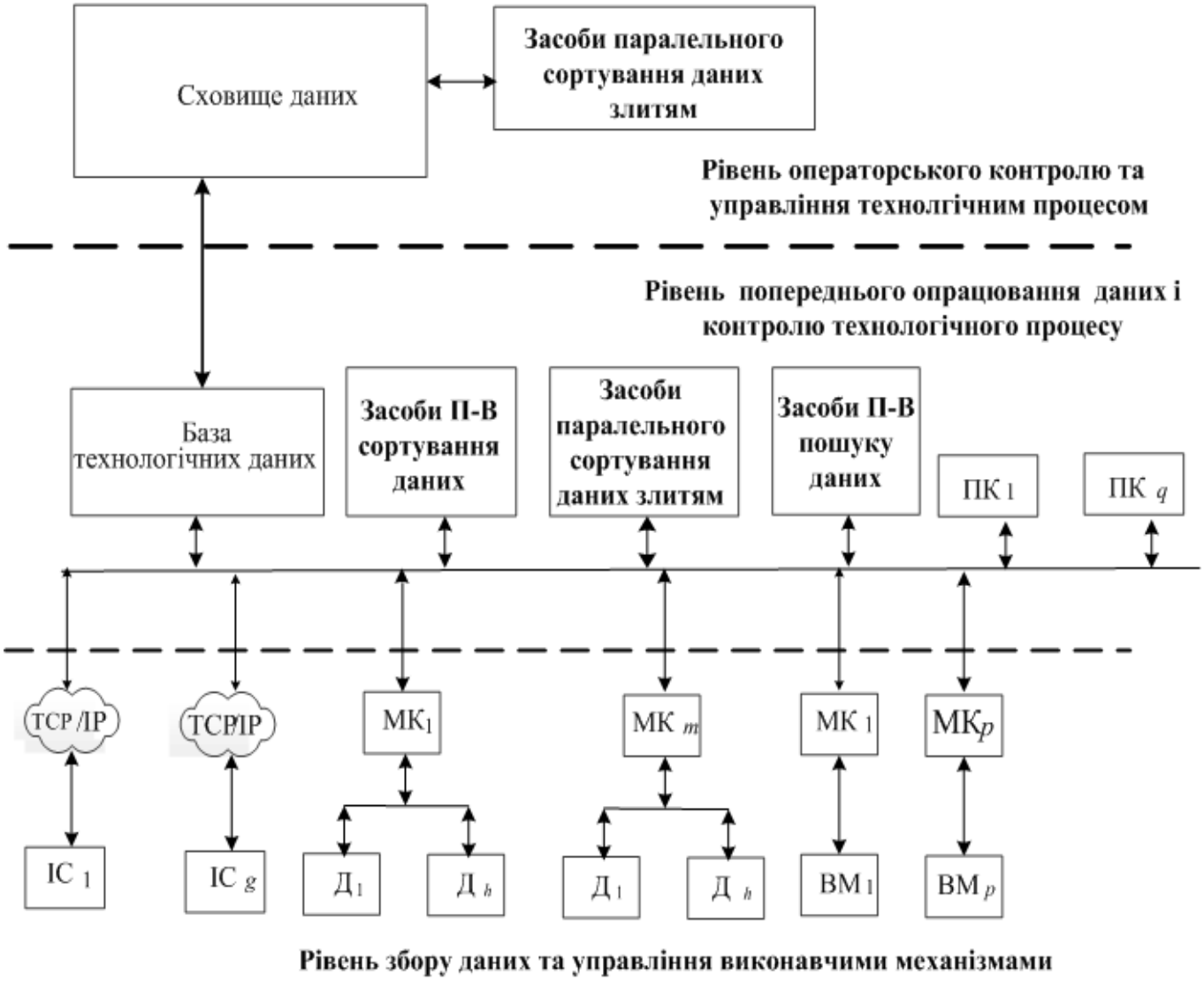


Рис.1.1. Структура автоматизованої системи багаторівневого управління технологічними процесами

Автоматизована система управління технологічними процесами складається із наступних рівнів:

- збирання даних та управління виконавчими механізмами;
- управління та контролю технологічним процесом;
- операторського контролю та формування управлінських рішень.

В автоматизованій системі управління технологічними процесами дані від більшості давачів і інтелектуальних сенсорів поступають на засоби паралельно-вертикального сортування даних та паралельно-вертикального пошуку даних. Сортування та пошук максимальних і мінімальних значень доцільно виконувати

паралельно-вертикальним методом, який передбачає паралельне надходження даних розрядними зрізами починаючи із старших розрядів.

Для зберігання оперативної інформації про стан технологічного процесу використовується база технологічних даних, яка знаходиться на рівні контролю та управління технологічним процесом. На операторського рівні контролю та формування управлінських рішень находиться сховище даних у якому зберігається як оперативна, так і історична інформація. Дані, які зберігаються у сховищі даних мають предметну організацію відповідно до основних напрямів діяльності підприємства. У цьому полягає відмінність сховищ даних від оперативної бази даних, в якій дані подаються відповідно до технологічних процесів. Організація даних не лише полегшує аналіз, а й в значній міри збільшує швидкодію проведення аналітичних обчислень. Дані, які вибираються з оперативної бази технологічних даних, накопичуються в сховищах даних у вигляді "історичних зрізів", кожен із яких містить дані за певний проміжок часу. Це в свою чергу надає можливість проведення аналізу зміни показників у часі. Дані для кожного "історичного зрізу" ні в якому разі не змінюються. Це є основною відмінністю в даних, які знаходяться в сховищах даних і оперативних даних.

1.2. Аналіз методів та алгоритмів сортування даних

Як було зазначено у підрозділі 1.1, сортування є процес перегрупування заданої множини елементів в деякому конкретному порядку, основною метою якого є підвищення ефективності пошуку елементів у відсортованій множині.

Аналіз джерел [19, 26-28] показав, що класифікація алгоритмів залежить від структури даних, яка опрацьовується. У випадку сортування, методи та алгоритми поділяться на наступні основні типи, як: внутрішнє сортування і зовнішнє сортування.

- Внутрішнє сортування – це алгоритми сортування, в яких процес впорядкування даних використовує тільки оперативну пам'ять комп’ютера чи сервера. Тобто об’єм оперативної пам'яті достатній для розміщення в ньому всього масиву даних з можливістю доступу до довільних комірок пам’яті і власне для

виконання самого алгоритму. Алгоритми, що відносяться до алгоритмів внутрішнього сортування використовується у переважній частині задач опрацювання масивів, за винятком однопрохідного вичитування даних та однопрохідного записування відсортованих даних. В залежності від вибраного алгоритму і методу його реалізації, дані можуть сортуватися в одній і тій самій області пам'яті, або ж використовувати допоміжну оперативну пам'ять.

- Зовнішнє сортування – це алгоритми сортування, при яких для упорядкування даних використовується зовнішня пам'ять, зазвичай, жорсткі диски. Методи зовнішнього сортування використовуються для обробки великих масивів даних, які повністю не поміщаються до оперативної пам'яті. При виконанні цього сортування, операції звернення до різних носіїв накладають додаткові обмеження на ці алгоритми, а саме, доступ до носіїв здійснюється послідовно, тобто в кожен момент часу можна вичитати або записати тільки один елемент, який є наступним за поточний, при цьому обсяг даних не дозволяє їх розмістити в оперативну пам’ять.

Внутрішнє сортування є основою для будь-якого алгоритму зовнішнього сортування, де окремі частини масиву даних сортуються в оперативній пам'яті і за допомогою спеціального алгоритму з’єднуються в один масив. Також, внутрішнє сортування значно ефективніше зовнішнього, оскільки звернення до оперативної пам'яті вимагає набагато менше часу, ніж до носіїв.

Також маючи характеристики вхідних даних, алгоритми сортування класифікуються за такими основними параметрами, як: час сортування, пам'ять, стійкість та природність поведінки [22, 25-28, 48].

- Час сортування – це основний параметр, який характеризує швидкодію алгоритму;

- Пам'ять – це параметр, який характеризує те, що ряд алгоритмів сортування вимагають виділення додаткової пам'яті під тимчасове зберігання даних. При оцінці використовуваної пам'яті не враховується місце, яке займає вихідний масив даних і вона не залежать від вхідної послідовності і системних витрати, наприклад на зберігання коду програми.

- Стійкість – це параметр, який характеризує, що в процесі упорядкування відносне розміщення елементів з рівними значеннями не міняється. Стійкість сортування часто являється бажаним фактором, якщо мова йде про елементи, що вже впорядковані за деякими вторинними ознаками (ключами), які не впливають на основний ключ.

- Природність поведінки – це параметр, який характеризує ефективність методу при обробці вже частково впорядкованих або повністю відсортованих масивів. Алгоритм поводиться природно, якщо при опрацюванні масивів він враховує дану характеристику вхідної послідовності і виконується краще.

Базуючись на описаних вище характеристиках, алгоритми можна класифікувати на два класи – прямі та швидкі.

- Прямі методи сортування є зручними для програмування та відлагодження, а програми, що їх реалізують є компактними. Особливістю даних методів сортування є повторення N етапів сортування даних із зменшенням на кожному наступному етапі кількості порівнювальних елементів. Такі алгоритми сортування мають складність $R=O(N^2)$ і їх доцільно використовувати для "коротких" масивів, тобто для масивів які для опрацювання не вимагають зовнішніх носіїв.

- Швидкі методи сортування вимагають меншу кількість етапів опрацювання, проте ці етапи є складнішими. Особливість таких етапів сортування є те, що переміщення елементів на "своє" місце супроводжується перегрупуванням решти елементів. Такі алгоритми сортування є ефективнішими при опрацюванні великих масивів даних.

1.2.1. Прямі методи сортування

На сьогоднішній день існує багато різноманітних методів прямого сортування, але аналіз [19, 20 , 28, 48] показав, що не всі з існуючих алгоритмів можна ефективно використати при створенні засобів інформаційних технологій паралельного сортування даних. Тому для ефективного розроблення ІТ було виділено наступні алгоритми сортування, як: сортування включенням, сортування вибором і сортування обміном.

Сортування включенням. Аналіз показав, що в основі алгоритму сортування включенням лежить операція пошуку для чергового елемента масиву x_{i+1} відповідного місця у відсортованій частині $a_1, a_2, ..., a_i$ із наступним включенням його у знайдену позицію [26]. Результатом цього є те, що елементи масиву можна умовно поділити на дві групи – вже відсортовану послідовність a_1 , a_2 , ..., a_i та вхідну послідовність. На кожному етапі опрацювання, починаючи з $i=2$, із вхідного масиву вибирається один елемент і включається в потрібне місце впорядкованої послідовності.

Процес порівняння (пошук потрібного місця для включення елемента x_{i+1}) може перерватися при виконанні однієї із двох умов:

1) знайдений елемент a_j з ключем, менший ніж ключ у x_{i+1};

2) досягнутий лівий кінець впорядкованої послідовності a_1.

Кількість порівнянь ключів C_i при i-му просіюванні найбільше дорівнює i, найменше – 1, а середня кількість – $i/2$. Кількість перестановок включаючи бар'єр дорівнює $M_i=C_i+2$. Тому для оцінки ефективності алгоритму у випадках коли масив початково є впорядкованим, зворотно впорядкованим або довільним, можна скористатись наступними формулами:

$$C_{min} = \sum_{i=2}^{N} C_i^{min} = \sum_{i=2}^{N} 1 = N - 1 \tag{1.1}$$

$$M_{min} = \sum_{i=2}^{N} M_i^{min} = \sum_{i=2}^{N} 3 = 3(N - 1) \tag{1.2}$$

$$C_{max} = \sum_{i=2}^{N} C_i^{max} = \sum_{i=2}^{N} i = \frac{N^2 + N - 2}{2} \tag{1.3}$$

$$M_{max} = \sum_{i=2}^{N} M_i^{max} = \sum_{i=2}^{N} (i + 2) = \frac{N^2 + 5N - 6}{2} \tag{1.4}$$

$$C_{mid} = \sum_{i=2}^{N} C_i^{mid} = \sum_{i=2}^{N} \frac{i}{2} = \frac{N^2 + N - 2}{4}$$

(1.5)

$$M_{mid} = \sum_{i=2}^{N} M_i^{mid} = \sum_{i=2}^{N} \left(\frac{i}{2} + 2\right) = \frac{N^2 + 9N - 10}{4}$$

(1.6)

Перевагами цього алгоритму є його простота реалізації, висока ефективність при роботі з невеликими масивами і частково відсортованими масивами, де продуктивність рівна *O(n + d)*, але на практиці ефективність рівна $O(n^2)$, що є кращим показником за більшість інших квадратичних алгоритмів. Також перевагою являється те, що даний алгоритму є стабільним.

Сортування вибором. Аналіз даного алгоритму показав, що в порівнянні з сортуванням включенням, коли для чергового елемента виконувався пошук відповідного місця в упорядкованій послідовності, цей метод сортування основується на виборі відповідного елемента масиву для певної позиції у масиві [28]. Цей підхід складається з наступних кроків:

1) на *i*-му етапі з елементів масиву a_i, a_{i+1}, ..., a_N вибираємо елемент з найменшим ключем a_{min};

2) проводимо обмін елементів масиву a_{min} та a_i місцями.

Процес послідовного вибору елемента і його перестановки відбувається поки не залишиться елемент з найбільшим ключем.

Аналіз алгоритму показав, що кількість порівнянь ключів C_i на *i*-му етапі вибору не залежить від початкового розміщення елементів масиву і дорівнює *N-i* і обчислюється так:

$$C = \sum_{i=1}^{N-1} C_i = \sum_{i=1}^{N-1} (N - i) = \frac{N^2 - N}{2}$$

(1.7)

При цьому кількість перестановок при опрацюванні масиву залежить від початкової впорядкованості елементів масиву.

Найкращий результат алгоритм показує для початково впорядкованого масиву $M_i=4$, а найгірший результат для зворотно впорядкованого масиву $M_i=C_i+4$. При опрацюванні довільно впорядкованого масиву в середнє значення дорівнює $M_i=C_i/2+4$. Тому оцінювання ефективності алгоритму основується на перестановці елементів та обчислюється так:

$$M_{min} = \sum_{i=1}^{N-1} M_i^{min} = \sum_{i=1}^{N-1} 4 = 4(N-1) \tag{1.8}$$

$$M_{max} = \sum_{i=1}^{N-1} M_i^{max} = \sum_{i=1}^{N-1} (N-i+4) = \frac{N^2+7N-8}{2} \tag{1.9}$$

$$M_{mid} = \sum_{i=1}^{N-1} M_i^{mid} = \sum_{i=1}^{N-1} \left(\frac{N-i}{2}+4\right) = \frac{N^2+15N-16}{4} \tag{1.10}$$

Перевагами цього алгоритму є його простота, ефективність n^2, стабільність. Недоліками алгоритму є його низька ефективність при опрацюванні великих масивів даних.

Сортування обміном. Аналіз алгоритму сортування обміном показав, що даний алгоритм заснований на операції порівнювання ключів сусідніх елементів при русі вздовж масиву. Якщо при порівнянні наступний елемент є меншим за попередній відбувається обмін. Таким чином при русі з права наліво, ключ з найменшим значенням переноситься на початок масиву за один етап проходження по ньому. При кожному наступному проході можна переривати процес опрацювання масиву у випадку, якщо було знайдено позицію мінімального елемента, так як всі елементи перед ним вже упорядковані. Цей процес подібний до поведінки бульбашок повітря у посудині з водою. Тому цей алгоритм також називають сортуванням бульбашкою [19, 20].

При опрацюванні масиву зліва на право, на етапі порівняння та обміну буде відбуватися виштовхування найбільшого елемента в кінець масиву. Даний процес подібний до опускання камінця у воді.

Базуючись на описі принципу роботи алгоритму даний алгоритм вимагає великої кількості операцій обміну. При його реалізації виникають питання щодо оптимізації і підвищенні ефективності за рахунок зменшення операцій порівняння. Цього можна досягти за допомогою наступних удосконалень:

1) на кожному етапі фіксувати, чи були перестановки елементів. Якщо перестановок не було, то робота алгоритму переривається;

2) крім факту обміну також потрібно фіксувати ще і індекс(позицію) останнього обміну. Так як всі елементи масиву перед ним або після нього, відповідно до принципів бульбашки і камінця будуть впорядковані;

3) почергово використовувати підходи камінця і бульбашки. Тому, що при чистій бульбашці за один прохід найменший елемент встановлюється на своє місце, а найбільший елемент опускається лише на один рівень. Побідна ситуація для камінця, тільки навпаки. Даний підхід називаються сортуванням змішуванням [47].

Кількість операцій порівняння ключів C_i на i-му проході алгоритму бульбашки не залежать від початкового розміщення елементів масиву і дорівнює N-i і обчислюється так:

$$C = \sum_{i=1}^{N-1} C_i = \sum_{i=1}^{N-1} (N - i) = \frac{N^2 - N}{2} \tag{1.11}$$

Кількість перестановок в свою чергу залежить від початкової впорядкованості масиву. Для різних характеристик вхідних масивів, можна отримати наступні значення кількості перестановок: найкращий результат при початково впорядкованому масиві $M_i=0$; найгірший результат при зворотно впорядкованому масиві $M_i=C_i*3$; у випадку довільного масиву отримуємо ймовірне значення $M_i=C_i*3/2$. Тому при оцінці ефективності алгоритму за кількістю перестановок можна використовувати наступні формули:

$$M_{min} = 0 \tag{1.12}$$

$$M_{max} = \sum_{i=1}^{N-1} M_i^{max} = 3\sum_{i=1}^{N-1}(N-i) = \frac{3(N^2-N)}{2} \tag{1.13}$$

$$M_{mid} = \sum_{i=1}^{N-1} M_i^{mid} = \frac{3}{2}\sum_{i=1}^{N-1}(N-i) = \frac{3(N^2-N)}{4} \tag{1.14}$$

Складність алгоритму бульбашки $O(n^2)$. Даний алгоритм, аналогічний до інших прямих методів сортування і має аналогічні до них переваги і недоліки. Хоча недоліків в ньому більше ніж преваг. Дональд Кнут прийшов до висновку, що «немає жодних підстав рекомендувати використання даного алгоритму, окрім, хіба що через примітну назву і через те, що він є лідером у кількості цікавих теоретичних проблем» [19]. Також недоліком даного алгоритму є те, що він погано використовує можливості сучасних мікропроцесорів.

На відміну від алгоритму обміном, в сортуванні змішуванням, кількість операцій порівняння залежать від початкового розміщення елементів в масиві. Кількість порівнянь в найкращому результаті для впорядкованої послідовності це значення буде $C_{min}=N-1$, а для зворотно впорядкованого масиву воно буде аналогічне до бульбашки.

В результаті, удосконалення алгоритму сортування обміном не змінює кількість переміщень елементів, а тільки зменшує кількість повторюваних порівнянь. В свою чергу для обміну елементів місцями вимагається більше пам’яті ніж порівняння ключів. Тому даний алгоритм буде вигідний тільки для частково впорядкованих масивів, і його складність буде прямувати до $O(n)$.

1.2.2. Швидкі методи сортування

Проаналізувавши [20, 21, 28, 29, 49] було знайдено, що на даний момент розвитку інформаційних технологій існує велика кількість різноманітних методів

швидкого сортування, що в тій чи іншій мірі показують високу ефективність при використанні їх для створення засобів ІТ паралельного сортування даних. В результаті аналізу було виділено алгоритми сортування, які дозволяють використовувати можливості апаратних засобів з максимальною ефективністю при реалізації їх на багатопроцесорних, багатоядерних системах та на елементній базі, як ПЛІС, а саме: сортування злиттям, швидке сортування і порозрядне сортування.

Сортування злиттям. Основна задача сортування даних методом злиття полягає злитті двох упорядкованих ділянок масиву в одну впорядковану ділянку іншого масиву**.** Метод сортування злиттям ґрунтується на базовій операції об'єднання двох упорядкованих масивів $\{a_{1i}\}_{i=1}^{2^{i-1}}$ та $\{a_{2i}\}_{i=1}^{2^{i-1}}$ у один упорядкований масив $\{b_{1i}\}_{i=1}^{2^{i}}$. На початку сортування вхідний масив чисел $\{a_j\}_{j=1}^{N}$ розділяється на $N/2$ упорядкованих масивів довжиною в одиницю. Результатом виконання першої базової операції є формування $N/4$ впорядкованих масивів довжиною два [3, 25]. Кількість базових операцій, які необхідні для сортування масиву із N чисел обчислюється за формулою [3]:

$$k=\log_2 N \tag{1.15}$$

Алгоритм сортування масивів даних методом злиття виконується на базі функціонального оператора Φ_{ji} [3]:

$$y=\begin{cases}0, & коли\ a_1 \le a_2\\ 1, & коли\ a_1 > a_2\end{cases},\quad b_1=\begin{cases}a_1, & коли\ y=1\\ a_2, & коли\ y=0\end{cases},\quad b_2=\begin{cases}a_1, & коли\ y=0\\ a_2, & коли\ y=1\end{cases}, \tag{1.16}$$

де y – результат порівняння чисел; a_1, a_2 – числа, які порівнюються; b_1, b_2 – виходи меншого та більшого відсортованих чисел.

Під час процесу сортування в дві допоміжні черги з основного масиву поміщаються перші дві відсортовані послідовності, які після цього зливаються в одну яка записується до тимчасової черги. Наступний етап це вибірка з основної черги наступних двох відсортованих підпослідовностей. Процес триває до поки основна черга не стане порожньою. Після цього послідовність з тимчасової черги переміщається в основну чергу. І знову продовжується сортування злиттям двох

відсортованих підпослідовностей. Виконання сортування буде триватиме допоки довжина відсортованої підпослідовності не стане рівною довжині самої підпослідовності.

Час роботи алгоритму *T(n)* по впорядкуванню n елементів задовольняється рекурентним співвідношенням [25]:

$$\mathrm{T}(n) = 2\mathrm{T}\left(\frac{n}{2}\right) + \mathrm{O}(n), \tag{1.17}$$

де $T\left(\frac{n}{2}\right)$ — час впорядкування половини масиву; $O(n)$ — час злиття половин.

Враховуючи, що $T(1) = O(1)$, розв'язкою співвідношення є:

$$T(n) = O(n \log n) \tag{1.18}$$

Також, алгоритм для своєї роботи вимагає $O(n)$ додаткової пам'яті.

Даний алгоритм є стабільним, оскільки не міняє порядок розміщення однакових елементів. Перевагою алгоритму є те що він ефективно працює як і з малими так і з великими масивами даних, є простим для реалізації і орієнтації на паралельне обчислення.

Швидке сортування. Алгоритм який був розроблений Чарльзом Гоаром, перевага цього алгоритму в тому, що він не вимагає виділення допоміжної пам'яті і виконується у середньому за *O(n log n)* операцій, а в найгіршому випадку за $O(n^2)$. Так як алгоритм використовує дуже прості операції і цикли, його швидкодія більша ніж в інших алгоритмів, що мають аналогічну асимптотичну оцінку складності [24].

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Для кожної з частин масиву впорядкування елементів відбувається рекурсивно.

Швидке сортування відноситься до алгоритмів на основі порівнянь, також він не є стабільним. Час виконання алгоритму залежить від збалансованості вхідного масиву, що характеризує розбиття. В свою чергу збалансованість залежить від елементу, який був обраний як опорний (елемент відносно якого виконується розбиття). Якщо розбиття збалансоване, то асимптотично алгоритм працює так само

швидко як і алгоритм сортування злиттям. У найгіршому випадку, асимптотична поведінка даного алгоритму є така ж погана, як і в алгоритмі сортування включенням [21].

Найгірша поведінка алгоритму виникає у випадку, коли процес розбиття масиву, породжує один підмасив з *n-1* елементом, а другий — з 0 елементами. Якщо припустити, що таке незбалансоване розбиття виникає на кожному рекурсивному виклику. Для виконання самої операція розбиття необхідно час *O(n)*. Тоді, рекурентне співвідношення для оцінки часу виконання, описується так:

$$T(n) = T(n-1) + T(0) + O(n) = T(n-1) + O(n) \quad (1.18)$$

Результатом даного співвідношення є: $T(n) = O(n^2)$

Найкращим розбиттям для даного алгоритму сортування є два підмасива, розмір кожного з яких не перевищує *n/2*. В даному випадку час виконання, обчислюється так:

$$T(n) <= 2T(n/2) + O(n) \quad (1.19)$$

Тоді:

$T(n) = O(n \log n)$ – асимптотичний найкращий час виконання.

У середньому випадку, очікуваний час роботи алгоритму для всіх можливих варіантів вхідних масивів становить *O(n log n)*, це показує, що середній випадок є ближчим до найкращого.

Порозрядне сортування. Порозрядне сортування це швидкий, стабільний алгоритм при якому сортування даних, відбувається шляхом групування їх ключів за індивідуальними значеннями та розподілення між ними їх позиції та значення. Позиційний запис необхідний, через те, що цілі числа можуть представляти рядки символів і числа з плаваючою комою. З цього випливає, що порозрядне сортування не обмежується цілими числами. Герман Холлерит розробив порозрядне сортування у 1887 році і використовував його на лічильно-перфораційних машинах [24].

В основі алгоритму лежить ідея початкового впорядкування всіх елементів за наймолодшим розрядом, потім впорядкування елементів за другим розрядом, потім

за третім, і так до найстаршого розряду. При такій роботі алгоритму на кожен розряд приходить значення з невеликого діапазону, що в результаті дозволяє виконувати його швидко і з малими затратами пам'яті [22].

Час роботи кожного циклу сортування залежить від алгоритму, який було обрано в якості допоміжного. Найчастіше використовується сортування підрахунком, час роботи якого $O(N+K)$, де K – кількість символів у алфавіті; N – кількість елементів в масиві і використовує додатково $O(N+K)$ пам'яті. Загальна кількість циклів впорядкування, відповідає кількості розрядів максимального елемента. Складність роботи алгоритму сортування підрахунком є $O(D*(N+K))$, де D – кількість розрядів. Якщо використовувати цей алгоритм для впорядковування масиву з цілих чисел, його складність визначається за формулою $O(NlogM)$, де M – найбільший елемент масиву [19].

1.2.3. Аналіз методів сортування великих масивів даних

Аналіз [25-30, 49, 56, 67] показав, що для сортування великих масивів даних характерне використання підходів зовнішнього сортування в комбінації з алгоритмами прямого та швидкого сортування, частина з яких була описана в підрозділі 1.2.1 і 1.2.2. Через те, що дані які необхідно впорядкувати зберігаються на зовнішніх пристроях і мають великий об’єм, який не дозволяє їх повністю перемістити в оперативну пам'ять та відсортувати, доводиться використовувати різного роду хитрощі. В даному випадку дані діляться на частини, які можна помістити в оперативну пам'ять, зчитуються, сортуються і записуються в тимчасові файли. Після чого відбувається фаза злиття, при якій всі ці тимчасові файли об'єднуються в один великий файл.

Найвідомішими методами для виконання у зовнішньому сортуванні є :

- сортування злиттям (просто злиття, природне злиття);
- покращене сортування (багатофазове сортування та каскадне сортування).

Основним поняттям при використанні зовнішнього сортування є поняття серії. Серія (впорядкована підмножина) – це послідовність елементів, впорядкована по ключу.

Кількість елементів серії називається довжиною серії. Серія, що складається з одного елемента, впорядкована завжди. Остання серія може мати довжину меншу за решту серій файлу. Максимальна кількість серій в файлі дорівнює N, де N – всі не впорядковані елементи. Мінімальна кількість серій рівна одній (всі елементи впорядковані).

В основі більшості методів зовнішнього сортування лежать процедури злиття та розподілу. Злиття – це процес об'єднання двох (або більше) упорядкованих серій в один впорядкований файл за допомогою вибору елементів, які доступні в даний момент. Розподіл – це процес поділу упорядкованих серій на два чи кілька допоміжних файлів.

Також важливим є поняття фази. Фаза – це операція по одноразовій обробці усієї послідовності елементів. На основі виконання фази алгоритми можна класифікувати на такі класи, як: двофазне сортування - це сортування, в якому окремо реалізуються фази розподілу і злиття та однофазне сортування - це сортування, в якому фази розподілу і злиття об'єднані в одну.

Одним з основних способів оптимізації та збільшення продуктивності алгоритмів зовнішнього сортування є використання паралелізму і його наступних принципів [24, 33, 56, 67]:

- для підвищення швидкодії послідовного зчитування та запису даних, доцільно використовувати декілька дисків паралельно, що також зменшує вартість розробки ІТ;

- шляхом використання сучасних багатоядерних процесорів та оптимізації програмного забезпечення, забезпечується підвищення швидкодії операцій сортування за рахунок використання потоків;

- використання асинхронного вводу та виводу при розробці програмного забезпечення, забезпечує збільшення швидкодії за рахунок одночасного сортування даних в одному пробігу та запису і вичитки в іншому;

- за рахунок використання декількох машин, які за допомогою високошвидкісної мережі пов'язані між собою, сортування даних виконується в паралельному режимі.

Також для збільшення швидкодії сортування використовують наступні методи [56, 67]:

- зменшення кількості операцій пошуку на диску при сортуванні даних, досягається шляхом використання великого об’єму оперативної пам'яті;

- використання сучасного апаратного обладнання, як швидкої зовнішньої пам'яті та твердотілих накопичувачів, забезпечує зменшення часу сортування, але збільшує вартість ІТ;

- за рахунок використання оптимальних та сумісних між собою апаратних засобів для створення ІТ забезпечується зменшення часу сортування. При виборі слід зважати на наступні параметри: швидкість, частота та кількість ядер процесора, латентність доступу до оперативної пам’яті, пропускна вхідна/вихідна здатність, швидкість ІО операцій диску, час пошуку та інші;

- баланс між економічною ефективністю та швидкодією можна досягти за рахунок використання кластерів та хмарових сервісів, де низька ціна досягається великою кількості нод кластера;

- оптимізація кількості входів, виходів, тимчасових системних файлів зменшує час сортування даних;

- використання стандартизованого підходу до паралелізму при всіх фазах сортування;

- використання максимального програмного розпаралелення при процесі сортування даних;

- використання оптимізації програмного коду за рахунок коректних типів даних тощо.

1.3. Аналіз методів та алгоритмів пошуку даних

При аналізі [20,21-24, 27, 30, 52, 74] було виявлено, що завдання пошуку даних в інформаційних технологіях є таким ж розповсюдженим як і завдання сортування даних.

Оскільки операція пошуку даних аналогічна до операцій сортування, або краще сказати що операція сортування є однією з невід’ємних складових пошуку. Методи

пошуку даних класифікуються на пошук даних у внутрішній пам’яті та за допомогою зовнішньої пам’яті по аналогії з методами сортування описаними в підрозділі 1.2.

У свою чергу алгоритми внутрішнього та зовнішнього пошуку даних можна поділити на два типи: методи інформативного пошуку та методи неінформативного пошуку [23].

Метод інформативного пошуку – це стратегія пошуку рішень в просторі станів при якій використовуються дані, що відносяться до конкретного завдання. Методи інформативного пошуку зазвичай забезпечують високу ефективність виконання пошуку в порівнянні з методами неінформованого пошуку. Інформація по конкретному завданні формується у вигляді евристичної функції. На кожному кроці виконання перебору даних, евристична функція виконує оцінювання альтернатив на підставі отриманої додаткової інформації. Метою даного оцінювання є прийняття рішення про напрям продовження перебору [22].

Неінформативні методи пошуку - це методи які використовують стратегію пошуку рішень в просторі станів при якій використовується тільки початкова інформація, яка представлена у визначенні завдання та не використовується додаткова інформація про стани. Основна мета методів неінформованого пошуку – це визначати наступників і відрізнити цільовий стан від нецільового [24].

1.3.1. Методи внутрішнього пошуку даних

Методи внутрішнього пошуку найчастіше використовують наступні підходи: пошук в динамічних деревовидних структурах і пошук в таблицях з використанням хешування [21].

Методи пошуку за допомогою дерев. Методи пов’язані з деревоподібними структурами широко поширені та інтуїтивно зрозумілі.

Існує декілька можливих визначень дерева [69, 70]. Використовуючи теорію графів, деревом називають неорієнтовний граф з встановленою вершиною (корінь дерева), який не містить в собі циклів. Потрібно виділити, що для створення ефективних засобів пошуку даних ІТ доцільно використовувати тільки орієнтовані графи. Використовуючи наступне рекурсивне визначення: дерево *B* базового типу *T*

– це порожнє дерево (дерево яке не містить вершин) або вершина типу *T* (корінь дерева) з кінцевим числом пов'язаних з нею дерев базового типу *T* (дані структури називаються піддеревами дерева *B*) можна сказати, що односпрямований список (рис. 1.2) є деревом [83].

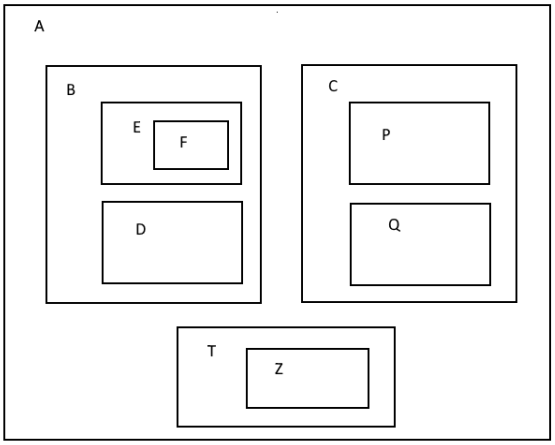


Рис 1.2. Структура односпрямованого списку

Дерева можна представляти по-різному (так як це однорідна ієрархічна структура). Наприклад, на рисунку 1.2 і 1.3 використовуються два різні підходи до візуалізації одного і того ж дерева. Базовий тип якого містить букви латинського алфавіту.

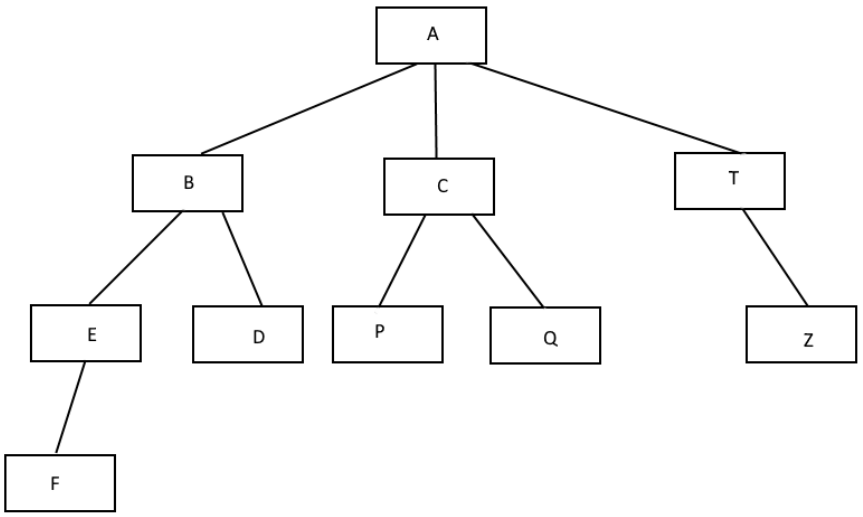


Рис 1.3. Граф у вигляді дерева

Використовуючи термінологію теорії графів, гілки – це зв'язки між піддеревами, в свою чергу вершинами називається корінь кожного піддерева. Дерево називається впорядкованим, тоді, коли гілки, що виходять з кожної вершини дерева є впорядкованими [28]. Приклад впорядкованих дерев зображений на рисунку 1.4.

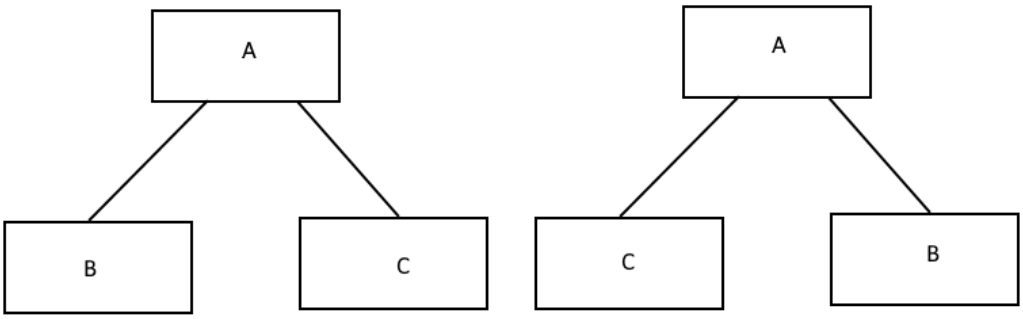


Рис 1.4. Впорядковані дерева

Для ефективної реалізації програмних засобі ІТ для пошуку даних крім базових визначень дерев, а саме вершини і коренів, також слід визначити поняття безпосередніх предків та нащадків вершин дерева та поняття листка дерева та ступеня дерева. Вершина *A* дерева яка безпосередньо пов’язана з вершиною *B* наступного рівня називається безпосереднім предком, в свою чергу вершина *B* такої вершини, називається безпосереднім нащадком. Листком дерева називається вершина без безпосередніх нащадків. Внутрішніми вершинами дерева називають нелисткові вершини. Ступінь внутрішньої вершини – це число безпосередніх нащадків. Ступінь дерева – це ступінь всіх вершин з однією і тою самою стеженню. Однакова ступінь вершин дерева досягається шляхом додавання спеціальних вершин в точки, де відсутні піддерева (рис. 1.5) [68, 83]. Довжиною шляху називається число вершин або гілок, які необхідно пройти від кореня дерева до його вершини. Глибина дерева – це максимальна довжина вершини.

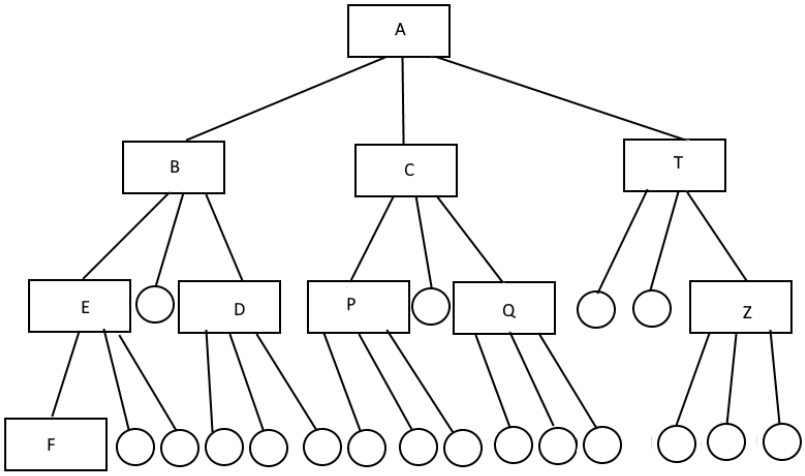


Рис 1.5. Дерево і піддерева

До методів пошуку даних за допомогою дерев відносяться наступні [23, 25-28]:

- пошук за допомогою двійкових дерев – це пошук в структурі даних, в якій кожна вершина має не більше двох піддерев. Пошук з використанням двійкових дерев є ефективнішим ніж пошуку в лінійних структурах оскільки середній час опрацювання дорівнює $O(nlog_2n)$;

- пошук за допомогою збалансованих двійкових дерев – це пошуку в структурі на подобі бінарного дерева, яка підтримує свою висоту автоматично, тобто під коренем дерева кількість рівнів вершин є мінімальною. Ідеально збалансоване дерево – це дерево у якому для кожної вершини різниця між висотами правого та лівого піддерев не перевищує одиниці.

- пошук за допомогою оптимальних дерев – це пошук у дереві, де всі відстані між вершинами утворюють найдовшу можливу послідовність чисел натурального ряду. Значення відстаней можуть повторюватися.

Методи пошуку за допомогою хешування. Аналіз методів пошуку за допомогою хешування показав, що в основі даних методів закладена ідея використання певної *f(H)* хеш-функції для заданому аргументу пошуку *x*, для отримання значення *f(x)*, яке найкращим чином б характеризувало місце ключа в основній (або зовнішній) пам'яті [26, 27, 28].

Аналіз показав, що до методів пошуку даних за допомогою хешування відносяться наступні [23, 25-28]:

- метод досконалого хешування – пошуку за допомогою використання додаткової таблиці з постійним найгіршим часом доступу;

- лінійне зондування – також ще називають пошуком за допомогою відкритої адресації або закрите хешування. При даному методі всі записи зберігаються в хеш-таблиці у вигляді ключ-значення. Пошук відбувається шляхом послідовного пошуку по таблиці відповідного значення або пустої комірки;

- подвійне хешування – даний метод подібний до лінійного зондування, різниця полягає в тому, що пошук інтервалу виконується за допомогою двох хеш-функцій;

- використання ланцюжків переповнення – це пошуку даних за допомогою зіткнення(порівняння за допомогою хеш-функції) та пов’язаних списків. В даному методі запис у хеш-таблиці є пов’язаним список.

1.3.2. Методи зовнішнього пошуку даних

Аналіз методів зовнішнього пошуку показав, що дані методи застосовуються у випадку коли об’єму основної пам'яті який необхідний для опрацювання даних є недостатній для розміщення даних цілком, по аналогії з методами зовнішнього сортування. Методи зовнішнього пошуку використовують таку саму класифікацію, як і методи внутрішнього пошуку, а саме пошук даних за допомогою дерев та за допомогою хешування, але відмінність полягає в оптимізації. Методи зовнішнього пошуку розроблені таким чином, щоб мінімізувати число звернень до зовнішньої пам'яті [28, 75].

Базовим методом для пошуку даних у зовнішній пам'яті є пошук за допомогою деревоподібних структур, саме B-дерева. Основною проблемою при роботі з даними методами є загальний час доступу до структур даних, оскільки він визначається в більшості випадків не обсягом даних, а часом руху магнітних головок у накопичувачах. Тому в основі даних методів лежить ідея економного використання внутрішньої пам’яті за рахунок отримання максимальної кількості даних з зовнішньої пам’ятні за одне звертання [23]. Одиницею обміну в методах пошуку у зовнішній памяті є сторінка, дані при такому підході організуються у вигляді набору сторінок.

Також для ефективного використання даних методів необхідно забезпечити такі структури, при яких пошук по ключу буде використовувати заздалегідь відому кількість обмінів із зовнішньою пам’яттю.

Аналіз показав, що до пошуку даних за допомогою B-дерев відносять наступні методи [23, 25, 26]:

- класичні B-дерева – це пошук за допомогою збалансованих дерев, які забезпечують ефективне збереження інформації на магнітних накопичувачах та інших пристроях з прямим доступом до пам’яті;
- B+ дерева – це пошук за допомогою бі плюс дерева, в якому корінь містить весь діапазон значень дерева, а кожний внутрішній вузол є підінтервалом;
- різновиди B+ дерев для організації індексів в базах даних – це пошук в базах даних, який використовує додаткові структури, як кластеризовані та некластеризові індекси на сторінках яких містяться дані, а на вершинах їх фізичне розміщення або як у випадку некластеризованих індексів посилання на кластеризований індекс або хіп-таблицю;
- R-дерева в просторових базах даних – це пошук за допомогою деревоподібних структур які використовуються просторову організацію даних.

1.4. Аналіз елементної бази для реалізації програмно-апаратних засобів інформаційних технологій паралельного сортування та пошуку даних

Аналіз методів паралельного сортування та пошуку даних [19-30, 50] показав, що одним з найефективніших засобів для реалізації являються багатоядерні і багатопроцесорні системи. Одним з найпоширеніших засобі з цієї категорії є графічні процесори.

Графічний процесор – це окремий пристрій сервера, ігрової платформи або персонального комп'ютера, який виконує графічні обчислення(рендеринг). Графічні процесори орієнтовані на ефективну обробку і відображення комп'ютерної графіки за рахунок спеціалізованої конвеєрної архітектури. Вони набагато ефективніші в опрацюванні графічної інформації ніж центральний процесор [40].

Зазвичай графічний процесор використовується для підвищення швидкості опрацювання тривимірної графіки, але для деяких задач його можна використовувати як засіб обрахунків (GPGPU). Основна відмінність обчислень на GPU в порівнянні із CPU є, те що архітектура GPU орієнтована на збільшення швидкодії обчислень графічних об'єктів, з використанням обмеженого списку команд [41].

Аналіз [40-41, 94, 125, 130] показав, що обчислення загального призначення на графічних процесорах GPGPU (General-purpose computing for graphics processing units) – це метод, при якому графічний процесор відеокарти використовується для виконання обчислень в застосунках, які зазвичай проводяться на центральному процесорі. Тобто, GPGPU використовуються для паралельних обчислювальних платформ, які дозволяють розробникам програмного забезпечення та інженерам отримувати доступ до графічних карт та писати програми, що дозволяють графічним процесорам керувати будь-яким завданням, яке можна розпаралелити. GPGPU відноситься до методів прискорених обчислень, при яких обчислювальні частини програми призначаються графічному процесору, а загальні обчислення переносяться на центральний процесор.

Для реалізації GPGPU обчислень найчастіше використовуються наступні платформи:

- CUDA – це архітектура програмно-апаратних обчислень за допомогою графічних процесорів, яка розроблена компанією NVIDIA. Мова програмування C [40].

- OpenCL – це відкритий стандарт розробки програм, які виконуються на графічних багатоядерних процесорах та центральних процесорах. Для розробки використовується мова програмування C [85].

- DirectCompute – це програмний інтерфейс який дозволяє виконувати обчислення на відеоадаптерах, інтерфейс є частиною DirectX починаючи з 10 версії [96].

- C++ AMP – це бібліотека, яка була розроблена компанією Microsoft для обчислень на графічних процесорах. Мова програмування C++ [90].

Основними виробниками графічних процесорів є NVIDIA і AMD. Відеокарти цих виробників відрізняються реалізацією форматів цілочисельних даних та даних з плаваючою комою.

Більшість операцій на GPU працюють векторизованим чином, тобто одна команда може виконуватись на багатьох потоках (значеннях) одночасно, це показує, що графічний процесор відноситься до класу процесорів SIMD, використання яких є ефективне при побудові засобів паралельного сортування та пошуку даних.

1.4.1. Програмно-апаратне порівняння GPU та CPU

З появою графічних процесорів їх використання початково зводилося до простої операції передачі даних від центрального процесора до GPU, а далі до засобів відображення. З розвитком та ростом популярності графічних процесорів, він став ефективним для зберігання простих, а пізніше складних структур даних представлених у форматі 2D або 3D. Операції обробки графіки на GPU виконуються набагато швидше, оскільки графічний процесор через свою багатоядерну архітектуру має одночасний доступ до кожної операції за рахунок великої кількості процесорних ядер. На відміну від центрального процесор, який повинен опрацювати кожен піксель або елемент даних. Також графічний процесор набагато швидше працює з оперативною пам’яттю ніж центральний процесор, так як він містить свою власну, але меншу за обсягом пам'ять до якої має безпосередній доступ [126, 130, 134].

Також можна виділити особливість GPGPU, щодо обміну інформації між GPU та CPU, яка дозволяє передавати інформацію в обох напрямках, в результаті зростає швидкість роботи алгоритмів. GPGPU конвеєри часто використовуються для підвищення ефективності роботи алгоритмів на великих масивах даних. Часто GPGPU використовується в складних наукових обчисленнях з великими наборами даних, зокрема в біології і органічної хімії, для відображення геному, аналізу біомолекул, досліджень білків та інше. Також конвеєри графічних процесорів використовують для покращення ефективності обробки зображень та комп'ютерного зору [125].

GPGPU – це більше історія про програмне забезпечення, а не апаратне. Тим не менш, використання спеціалізованих апаратних архітектур може значно підвищити ефективність GPGPU потоків, які виконують декілька алгоритмів на великих масивах даних. Завдання розпаралелення масиву даних на велику кількість потоків, реалізуються за допомогою спеціальних застосунків, таких як обчислювальні стійки.

Основні відмінності розробки програмних застосунуків під CPU та GPU полягають у роботі з потоками. Реалізація потоків на GPU займає набагато менше часу і ресурсів, оскільки сама архітектура є багатоядерною.

Таб 1.1.

Операція	CPU	GPU
Створення потоку	Займає багато часу	Займає мало часу
Робота у потоці	Може виконувати будь-яку операцію	Спеціалізується на виконанні простих математичних операцій
Кількість потоків	Не велика кільсть, в залежності від процесора 4-6	Велика бількість, > 100
Робота з кешем	Підтримується	Підтримується

Таб.1.1. Відмінності в розробці програмних засобів під CPU та GPU.

Оскільки GPU орієнтовані на роботу з потоками, були виділені наступні операції роботи, як [40, 41, 125]:

- складання (Map) – це операція при якій задана базова функція(ядро) застосовується до кожного елементу потоку. До прикладу, операція ділення кожного значення в потоці на задану константу(зменшення яскравості зображення);
- редукція – це операція зменшення потоку, оскільки для деяких обчислення необхідне опрацювання меншого потоку з більшого потоку;

- фільтрація потоку – це нерівномірна редукція. Фільтрація включає в себе операції видалення елементів з потоку, базуючись на певних заданих умовах;
- сканування (scan) або паралельний префікс суми – операція створення комбінацій елементів масиву. Можна виділити виключаюче сканування і включаюче сканування;
- розкид (Scatter) – є найбільш природно визначеною операцією у вертексних процесорах. Даний процесор має здатність визначити позицію вершини, яка дозволяє контролювати місце у сітці для збереження інформація. Також він надає інші функції, такі як контроль об'єму місця в сітці;
- збір (Gather) – це операція зворотна до операції розкиду. Збірка відновлює порядок елементів відповідно до сітки, яка використовувалась для операції розкиду;
- сортування (Sort) – це операція перетворення невпорядкованого набору елементів масиву у впорядкований. Найчастіше використовуються базові швидкі методи сортування, оскільки їхня простота дозволяє їх використовувати як базову операцію на багатьох потоках;
- пошук (Search) – це операція, яка дозволяє знайти заданий елемент в потоці або знайти сусідні елементи до заданого. Використання GPU як засобу збільшення швидкодії пошуку окремого елемента не є ефективним і не часто використовується на практиці, зазвичай GPU використовується для одночасного виконання декількох пошуків. В основному для пошуку використовується алгоритм бінарного пошук на відсортованих масивах;
- структури даних в GPU можна класифікувати як: статичні або динамічні розріджені маси, адаптивні структури (комбінації), щільні масиви.

1.4.2. Аналіз мов і платформ розробки засобів паралельного сортування та пошуку даних

З розвитком інформаційних технологій зростає кількість мов програмування. Деякі мови є вузько спеціалізовані, інші мови в свою чергу є універсальними за рахунок діалектів, розширень та бібліотек. Аналіз платформи CUDA показав [40, 41, 130, 131, 134], що найефективніше працювати з мовою програмування C. Базуючись

на джерелах [86-89], мова С – це універсальна, процедурна мова програмування загального призначення, створена Деннісом Рітчі у 1972 році в компанії Bell Telephone Laboratories з метою розроблення операційної системи UNIX [91]. Початково мова С була розроблена для написання системного програмного забезпечення, в даний момент вона часто використовується для написання різного роду прикладного програмного забезпечення.

Основною перевагою мови С – це те, що ця мова програмування є мінімалістичною, при її розробці були сформульовані такі головні цілі, як: можливість прямолінійної реалізації компіляції, за рахунок використання відносно простого компілятора; забезпечення низькорівневого доступу до оперативної пам'яті; формулювання невеликої кількості команд машинної мови для кожного елементу мови [86]. Як результат, код мови С став придатний для більшості системного забезпечення, яке зазвичай розробляли за допомогою асемблера. Незважаючи на низькорівневість, мову проєктували для кросс-платформерного програмування. За рахунок сумісності зі стандартами кросс-платформеності, розроблена на мові С програма, може були легко скомпільована на великій кількості операційних систем та апаратних платформ з мінімальними змінами в коді програми. Мова програмування С підтримує велику кількість платформ, від мікроконтролерів до суперкомп'ютерів.

Так як мову програмування С перш за все розробляли для використання у системному програмуванні в неї є ряд своїх переваг і недоліків. Код написаний на С компілюється в простий набір машинних операцій, які забезпечують простий та безпосередній доступ до будь-якого об'єкта системи від компонентів операційної системи до апаратних засобів, тому при роботі з С часто виникають непередбачені помилки з неочікуваними результатами. Оскільки в мові С спрощений етап валідації коду і зазвичай компілятор не виявляє помилки в коді до моменту запуску програми. Тому при програмуванні та підтримці програмного забезпечення слід дотримувались стандартизації коду і суворих правил, щоб позбутися можливих проблем зі швидкодією, стабільністю та безпекою. Тому для С активно розбавляються додаткові утиліти для покращення валідації, але за це потрібно платити швидкодією виконання програмного забезпечення [89].

Оскільки С заснована на традиції АЛГОЛ, вона як і більшість імперативних мов має наступні можливості [88]:

- структурного програмування;
- дозволяє здійснювати рекурсії;
- строго типізована з статичними типами.

Код програм С міститься у функціях. За рахунок гетерогенних сукупностей типів даних, С дозволяє об’єднувати пов'язані типи даних і маніпулювати ними, як єдиним цілим.

Виділяють такі етапи і стандарти розвитку мови С: K&R C (1978), ANSI C (1983, 1990), C99 (1999, 2000), C11 (2011).

С використовують як проміжну мову у деяких високорівневих мовах програмування. Також С є основою для компіляторі, бібліотек та інтерпретаторів багатьох високорівневих мов.

Останнім часом С часто критикують, незважаючи на її велику популярність. Зазвичай критиці піддають те, що на С важко реалізовувати операції, які легко реалізуються на високорівневих мовах та за можливість легко здійснити дії, які призводять до небажаного результату. Також можна виділити, те що С вимагає більше професійних навичок від розробника, ніж для деяких інших мов.

Мова С стала основою для таких мов програмування: C++, Objective-C, C#.

1.4.3. Аналіз платформи CUDA для інформаційних технологій реалізаццї паралельного сортування та пошуку даних на GPU

Аналіз сучасних засобів реалізації GPGPU [40, 41, 130, 131, 134] показав, що найбільш ефективними і розповсюдженими являються програмно-апаратні архітектури CUDA і OpenCL. CUDA – це програмно-апаратна архітектура паралельних обчислень розроблена компанією Nvidia [91]. Для реалізації програмних рішень на CUDA, використовується спрощений діалекті мови програмування С. Архітектура CUDA надає повний доступ до апаратних ресурсів графічної карти (процесора): ядрам, пам’яті тощо. На даний момент архітектуру паралельних обчислень CUDA підтримують наступні серії графічних процесорів, а саме GeForce,

ION, Quadro і Tesla. В основі архітектури CUDA лежить використання моделі пам'яті GRID, кластерне моделювання потоків і використання SIMD-інструкцій. Паралельні обчислення за допомогою CUDA застовуються не тільки для високопродуктивних графічних даних, але і у наукових обчисленнях, а саме в астрофізиці, біології, хімії, моделюванні динаміки рідин, електромагнітних взаємодій, комп'ютерній томографії, геологічну аналізі і в інших сферах де необхідна обробка великих масивів даних [41].

На відміну від традиційних підходів до обчислень загального призначення в архітектури CUDA є наступні переваги [121, 122, 126]:

- інтерфейси CUDA API засновані на стандартній мові програмування C з деякими обмеженнями;

- ефективна взаємодія між пам'яттю центрального процесора і пам’яттю відеокарти;

- загальна для потоків пам'ять (shared memory) розміром 16 Кб використовується для реалізації ефективного кешування за рахунок широкої пропускної смуги у порівнянні з вибіркою із звичайних структур;

- повна апаратна підтримка побітових і цілочисельних операцій;

- підтримка компіляції коду для GPU засобами відкритого LLVM.

В концепції архітектури CUDA, обчислення виконуються як на GPU, так і CPU. GPU використовується, як масивно-паралельний співпроцесор, на якому ядро (базова програма) одночасно виконуються на великій кількості потоків. В свою чергу на CPU виконуються послідовні частини програмного засобу та здійснюється підготовка для GPU-обчислень. Основну програму в такій моделі називають хостом (host), а сам GPU називають приладом (device) [40, 41].

Варто також зазначити основні відмінності між потоками центрального процесору CPU і потоками графічного процесору GPU:

- потік (thread) GPU дуже легкий, та містить мінімальну кількість контексту, а регістри розподіленні заздалегідь;

- ефективність програних засобів на GPU залежить від потоків, чим більше було задіяно потоків, тим ефективніше програма використовує ресурси графічного

процесора, на відміну до CPU, де максимальна ефективність, здебільшого досягається при кількості потоків рівній або в кілька раз більшій за кількість ядер.

На рисунку 1.6. зображений приклад схеми обчислень в CUDA.

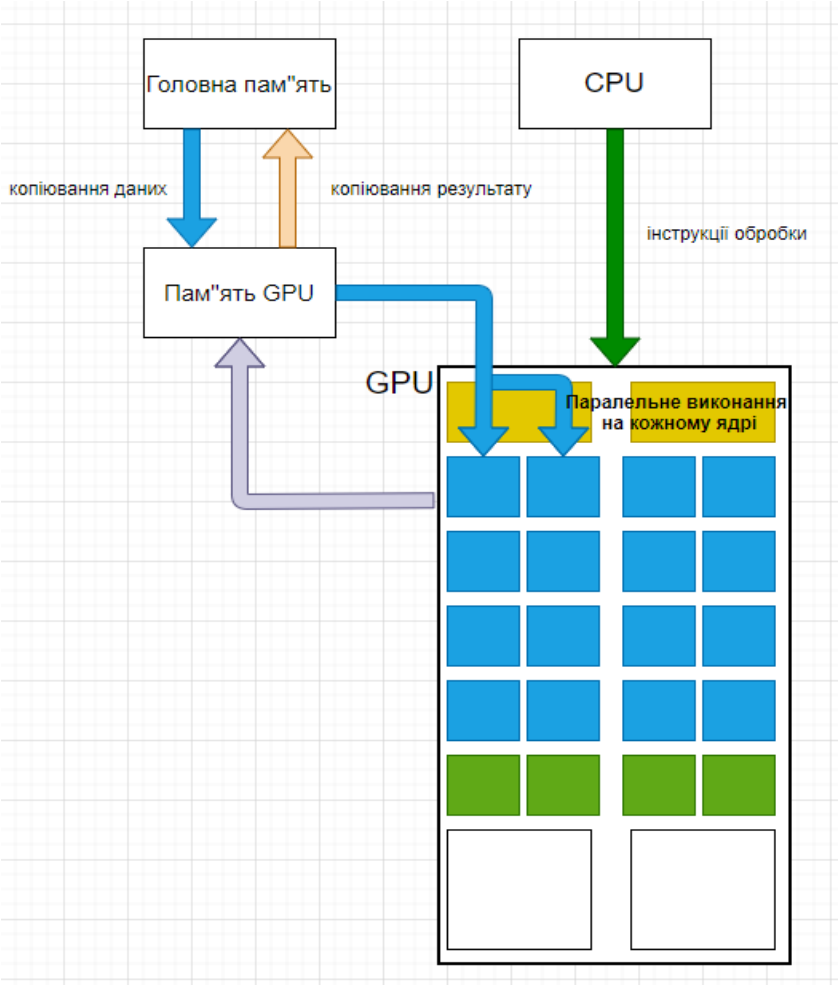


Рис. 1.6. Схеми обчислень в CUDA

Потоки в GPU працюють за принципом SIMD, але з невеликою різницею в реалізації. В графічному процесорі тільки потоки в межах однієї групи або як в термінології CUDA варпу (warp) фізично виконуються одночасно. Під час виконання програми, потоки різних варпів можуть знаходитися на різних стадіях виконання. Даний метод опрацювання даних називається SIMT (Single Instruction – Multiple Theads). Здійснення управління варпами виконується на апаратному рівні. Останнім часом з ростом популярності CUDA, GPU поступового перетворюється в самодостатній пристрій, який повністю заміняє звичайний CPU за рахунок реалізації системних викликів і інтеграції в GPU спрощеного енергоефективного CPU-ядра

(архітектура Maxwell). Також перевагою CUDA є те, що вона підтримує високорівневі мови програмування.

В свою чергу, платформа апаратно та програмно організована так, що ядро обрахунків може бути виконано кількома блоками потоків, так що загальне число потоків для паралельної роботи дорівнює кількості потоків на один блок помноженому на кількість блоків.

Блоки організовані у вигляді одновимірної, двовимірної або тривимірної сітки блоків потоків, як показано на рис.1.7. Кількість блоків в сітці зазвичай диктується розмірами оброблюваних даних [41].

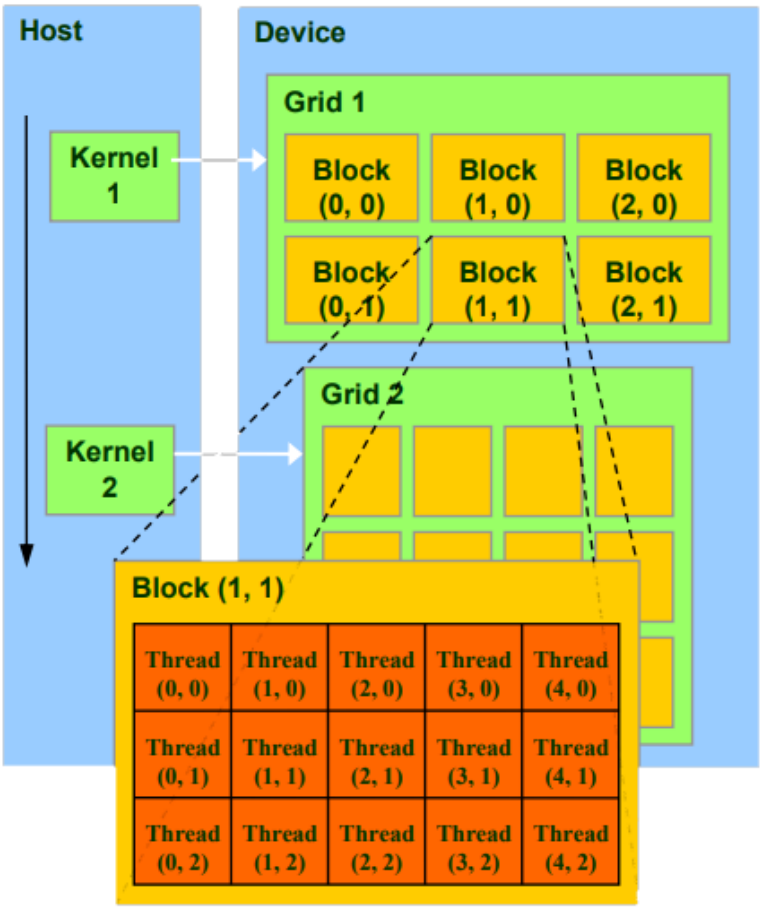


Рис.1.7. Сітка блоків потоків.

Блоки являють собою об’єднання процесорів які мають спільну пам'ять і виконують операції синхронно. На відміну від блоків де всі процесори працюють синхронно і мають спільну пам'ять, в сітці не можливо передбачити порядковість запуску блоків і також між блоками немає і не може бути спільної пам'яті, оскільки це зумовлено особливостями архітектури програмних рішень CUDA.

1.4.4. Аналіз платформи OpenCL для реалізації інформаційних технологій паралельного сортування та пошуку даних на GPU

Як було сказано в попередньому підрозділі, поширеність архітектури OpenCL стоїть на одному рівні з CUDA. OpenCL – це фреймворк для розробки програмних засобів, пов'язаних з паралельними обчисленнями на графічних та центральних процесорах [85]. Як і CUDA, фреймворк OpenCL використовує мову програмування, яка побудована на стандарті C99. OpenCL забезпечує розпаралелення як на рівні інструкцій програми так і на рівні роботи з даних, і є реалізацією техніки GPGPU. OpenCL – відкритий стандарт розроблений компанією AMD та його використання доступне на базі вільних ліцензій.

Стандарт OpenCL замислювався як технологія для створення додатків, які могли б виконуватися в гетерогенному середовищі. Для координації роботи всіх цих пристроїв в гетерогенній системі завжди є один «головний» пристрій, який взаємодіє з усіма іншими за рахунок OpenCL API. Даний пристрій називається «хостом», він визначається поза OpenCL. Хост передбачає використання для обчислень «процесора», тобто пристроя який виконує команди. Оскільки стандарт OpenCL розроблений для паралельних обчислень, то даний «процесор» може в собі містити засоби паралелізму (наприклад, кілька ядер одного CPU , кілька SPE процесорів в Cell). Також ефективним способом збільшення продуктивності паралельних обчислень є встановлення декількох «процесорів» на одному пристрою. І природно в гетерогенній системі є використання декількох OpenCL-пристроїв (також і з різною архітектурою). Крім обчислювальних ресурсів пристрій має свій обсяг пам'яті. Притому відсутні вимоги до цієї пам'яті, вона може бути як на пристрої, так і взагалі бути розмічена на ОЗУ хоста (до прикладу, вбудовані відеокарти) [85].

Таке широке поняття про пристрій дозволяє не накладати обмеження на програми, розроблені для OpenCL. Ця технологія дозвляє розробляти як додатки, які оптимізовані під конкретну архітектуру специфічного пристрою, що підтримує OpenCL, так і кросплатфомлені програмні засоби, які демонструють стабільну

продуктивність на різних типах пристроїв (при умові еквівалентної продуктивності цих пристроїв).

Для опису основних ідей OpenCL використовується ієрархія з 4-х моделей [85]:

- модель платформи (Platform Model);
- модель виконання (Execution Model);
- модель пам'яті (Memory Model);
- програмна модель (Programming Model).

Модель платформи (Platform Model). Платформа OpenCL складається з хоста який з'єднаний з пристроями, які підтримують OpenCL. Кожний OpenCL-пристрій складається з обчислювальних блоків (Compute Unit), які поділяються на один або більше процесорних елементів (Processing Elements, ПЕ). OpenCL-додаток виконується на хості відповідно до моделей вказаної платформи. У даному випадку хост програмного засобу відправляє команди виконання обчислення на ПЕ, а ПЕ у свою чергу в рамках обчислювального блоку виконують один потік команд як SIMD блоки, або як SPMD блоки.

Модель виконання (Execution Model). Виконання програмних засобів OpenCL складається з двох частин: частини яка виконується на хості і ядрах (kernels), що виконуються на OpenCL-пристрої. Хост програмного засобу визначає контекст, в якому будуть виконуватись ядра та виконує управління. Основна частина моделі виконання OpenCL описує виконання ядер. Коли ядро стає в чергу на виконання, визначається простір індексів (NDRange). Копія ядра, що виконується для конкретного індексу називається робочою одиницею («Work-Item») і визначається ключем в просторі індексів, тобто кожній одиниці ядра присвоюється всій унікальний глобальний індетифікатор. Всі Work-Item виконують однаковий код, але кожний конкретний шлях виконання та дані з якими він працює, можуть бути різними. Work-Item'и організовуються в групи (Work-Groups). Групи надають більш велике розбиття в просторі індексів. Кожній групі приписується груповий ID з такою ж розмірністю, яка використовувалася для адресації окремих елементів. Кожному елементу зіставляється унікальний, в рамках групи, локальний ID. Таким чином, Work-Item'и

можуть бути адресовані як по глобальному ID, так і по комбінації групового і локального ID [94].

Робоча одиниця, яку виконує ядро може використовувати чотири різних типи моделей пам'яті:

- глобальна пам'ять. Ця пам'ять яка надає доступ на читання і запис елементам всіх груп. Кожна робоча одиниця може записати і вичитувати з будь-якої частини об'єкта пам'яті. Записування і читання глобальної пам'яті в залежності від можливостей пристрою може кешуватися;
- константна пам'ять. Постійна область глобальної пам'яті, яка не змінюється під час виконання ядра. Хост виділяє та ініціалізує об'єкти в константній пам'яті;
- локальна пам'ять. Область пам’яті для групи. Ця область пам'яті може використовуватися для створювання змінних, що використовуються всією групою. Вона може бути реалізована як окрема пам'ять на OpenCL-пристрої;
- приватна (private) пам'ять. Область пам'яті, що належить робочій одиниці. Змінні, що визначені у приватній пам'яті одної робочої одиниці, не видимі іншим.

Модель виконання OpenCL підтримує дві програмні моделі (Programming Model), як : паралелізм даних (Data Parallel) і паралелізм завдань (Task Parallel), так само підтримуються і гібридна модель. Основна модель OpenCL, це паралелізм даних.

Програмна модель з паралелізмом даних - це модель, яка визначає обчислення як послідовність інструкцій, вживаних до безлічі елементів об'єкта пам'яті. Простір індексів, асоційований з моделлю виконання OpenCL, визначає Work-Item'и і як дані будуть розподілятися між ними. В жорсткій моделі паралелізму даних існує сувора відповідність один одному між Work-Item і елементом в об'єкті пам'яті, з яким ядро може працювати паралельно. OpenCL реалізує більш м'яку модель паралелізму даних, де чітка відповідність один до одного не є потрібною [85].

OpenCL надає ієрархічну модель паралелізму даних. Існує два способи визначення ієрархічних розподілів. У явній моделі розробник визначає загальну кількість елементів, які повинні виконуватися паралельно, а також яким чином ці елементи будуть розподілені по групах. У неявній моделі розробник лише визначає

загальну кількість елементів, які повинні виконуватися паралельно, а поділ за робочими групам виконується автоматично.

З цього випливає, що OpenCL орієнтується на гетерогенну модель на відміну від CUDA, але поточна реалізація платформи OpenCL базується на драйверах для платформи CUDA та графічних процесорах фірми Nvidia. Основні підходи до організації моделі обчислень на певній сітці процесорних елементів із потоками характерні для них обох.

1.5. Аналіз елементної бази та мов розробки апаратних засобів інформаційних технологій паралельного сортування та пошуку даних

Перспективною елементною базою для реалізації апаратних засобів інформаційних технологій паралельного сортування і пошуку даних є програмовані логічні інтегральні схеми (ПЛІС). Основними перевагами використання ПЛІС є [61-63,102, 136-140]: невисока вартість, що зумовлена масовим виробництвом; доступність; висока швидкодія; продуктивність; надійність; універсальність; різноманіття у виборі живлення та параметрів вхідних/вихідних сигналів. Також перевагами є низьке енергоспоживання, що є особливо важливим для реалізації портативної апаратури, наявність різноманіття добре поширених, ефективних і розвинутих програмних засобів для автоматизованого проектування та зменшення часу проектування і відлагодження проектів та реконфігурацію, що забезпечує велику гнучкість проектування. Суттєвими особливостями останніх поколінь ПЛІС є забезпечення часткової реконфігурації в процесі роботи. Зокрема, системи на базі ПЛІС можуть бути адаптовані для конкретних конфігурацій ШНМ.

Порівняно з ПЛІС попереднього покоління, спостерігається значне зниження рівня споживаної потужності при збільшенні продуктивності роботи. Наприклад, використання ПЛІС сімейства Artix-7 порівняно з Spartan-6 дозволяє в вдвічі зменшити енергоспоживання та підвищити продуктивність на 30%. Аналогічно, використання Kintex-7 замість Virtex-6 також дозволяє вдвічі зменшити енергоспоживання [63].

ПЛІС поділяються на 4 класи [63]: CPLD (Complex Programmable Logic Devices), FPGA (Field Programmable Gate Arrays), SPLD (Simple Programmable Logic Devices) та ПЛІС з комбінованою архітектурою. Найчастіше для реалізації використовуються FPGA, до яких часто відносять і ПЛІС з комбінованою архітектурою, оскільки в них найбільш виразно проявляються характеристики класу FPGA. ПЛІС цього класу дозволяють реалізовувати складні засоби, оскільки містять більшу кількість логічних блоків порівняно з іншими класами. Наприклад, SPLD містять декілька сотень логічних вентилів, тоді як FPGA – декілька мільйонів.

Програмні засоби розробки апаратних пристроїв сортування та пошуку на ПЛІС. В залежності від вибору апаратного засобу для алгоритмів сортування та пошуку даних виникає проблема вибору програмних засобів для реалізації цих методів. У свою чергу розроблення програмних рішень на НВІС та ПЛІС здебільшого використовуватися мови опису дискретних пристроїв, як:

- HDL;
- VHDL;
- Verilog HDL;
- AHDL.

При проектуванні апаратно-програмних засобів ІТ паралельного сортування та пошуку доцільно використовувати мови опису дискретних пристроїв HDL. HDL - це спеціалізована формальна комп'ютерна мова, яка використовується для проектування структур мікросхем, їх дизайну електронних мікросхем та моделювання їх роботи. Вона надає можливість автоматично імітувати, аналізувати та тестувати створюваний пристрій. Компілятор HDL надає можливість переведення програми, написаної на будь-якій з мов опису апаратури на низькорівневу специфікацію фізичних електронних компонентів з метою створення мікросхем [81].

Можна виділити наступні преваги мови проектування апаратури над схемним проектуванням [63]:

- можливість проектування пристроїв, для яких розроблення схем неможливе через велику складність засобу, до прикладу, розробка сучасних процесорів;
- опис коду об'єднує структурну і функціональну складові;

- проект на мові проектування апаратури портативний та універсальний, проект без проблем переноситься на інші елементні бази і може використовуватися в іншому проекті. Код продукту можна дуже довго підтримувати, так як він залежить тільки від «застарівання» мікросхем;
- мови опису апаратури є високо надійними для розроблюваного пристрою та забезпечують автоматичне документування.

Мова опису апаратури дуже подібна до мови програмування С, так як її структура складається з такої самої семантики. Мови проектування апаратури також надають можливість описувати специфікації апаратного забезпечення, які можна застосовувати при розробці. Це надає ілюзію використання мови програмування, хоча вони відносяться до мов моделювання та проектування. Для цього існують декілька причин, перша відмінність в тому, що інструкції мов проектування апаратних засобі завжди виконуються паралельно. Наступна ключова відмінність в описі синхросигналу, що є особливістю проектування апаратного засобів. Мови опису апаратних засобів можуть використовуватися для створення пристрою у структурних, поведінкових формах та на рівнях регістрових передач з ідентичною функціональністю.

На основі мови опису дискретних пристроїв розроблені наступі мови: AHDL, VDHL і Verilog HDL.

ADHL – це мова опису апаратних засобів, розроблена компанією Altera, Призначена для опису цифрових автоматів, комбінаційних логічних пристроїв і таблиць істинності з врахуванням архітектурних особливостей ПЛІС. Мова має Ada-подібний синтаксис. AHDL сумісна з компіляторами Altera: Max+PLUS і Quartus, та надає можливість створювання ієрархічних проектів в межах мови та комбінувати різні типи файлів для створення єдиного проекту [61].

VHDL – мова опису апаратурної архітектури інтегральних схем. Дана мова автоматизованого проектування VHDL є базовою мовою для розробки архітектури та структур сучасних обчислювальних систем. VHDL створена як інструмент опису цифрових систем, проте існує діалект мови – VHDL AMS (аналогові та змішані

сигнали), що використовується для опису аналогових, змішаних та цифро-аналогових схем [80].

Verilog HDL – це мова опису апаратурної архітектури, що використовується для моделювання та опису електронних систем. Найчастіше використовується в проектуванні, верифікації і реалізації цифрових, аналогових та змішаних електронних систем різного рівня абстракції [81]. Для ефективного використання мови Verilog та спрощення її освоєння, розробники використали синтаксис мови який був б максимально подібний до синтаксису мови С. Також варто зазначити, що опис апаратурної архітектури, написаний мовою Verilog зазвичай називають програмами, але, на відміну від прийнятого поняття програми, як послідовності команд, дана програма представляє собою масив операторів, які за допомогою сигналів виконуються циклічно і паралельно. Кожний оператор є моделлю певного елементу функціональної інтегральної схеми, а сигнал – аналогом логічного сигналу. У мові автоматизованого проектування Verilog не застосовується термін «виконання програми». Можна сказати, що виконання Verilog-програми являє собою моделювання функціональної схеми, яку виконує спеціальна програма – Verilog-симулятором [81].

1.6. Формулювання наукового завдання та завдань дослідження

Зі всього вищезгаданого випливає, що вирішення головного наукового завдання розроблення нових та удосконалення існуючих методів, моделей і засобів інформаційних технологій паралельного сортування і пошуку даних забезпечить опрацювання інтенсивних потоків даних для конкретних галузей застосувань у реальному часі з високою ефективністю використання обладнання.

Досягнення поставленої мети передбачало вирішення таких завдань::

- розроблення методу паралельно-вертикального сортування даних;
- розроблення методу паралельно-вертикального пошуку максимальних і мінімальних чисел у масивах;
- удосконалення методу паралельно-потокового сортування даних злиттям;
- розроблення інформаційної технології паралельного сортування даних;

- розроблення інформаційної технології паралельного пошуку даних;
- розроблення засобів інформаційних технологій паралельного сортування та пошуку даних.

1.7. Висновки до розділу 1

1. Аналіз галузей застосувань, що в значній частині галузей застосування інформаційних технологій сортування та пошуку даних вимагається сортування та пошук максимальних і мінімальних значень у реальному часі при інтенсивному паралельному надходженні потоків даних з високою ефективністю використання обладнання.

2. Найчастіше інформаційні технології сортування та пошуку використовуються у системах обробки технологічних даних у реальному часі, а саме системи збирання та попередньої обробки телеметричних даних, управління складними об’єктами, автоматизовані системи багаторівневого управління технологічними процесами тощо.

3. Проаналізовано та проведено оцінювання складності методів і алгоритмів сортування та пошуку даних та визначено, що їх реалізація вимагає великої кількості операцій попарно порівняння та перестановки даних і в більшості випадків мають квадратичну залежність.

4. Показано, що більшість методів і алгоритмів сортування відрізняються вибором даних для попарного порівняння.

5. Визначено, що для паралельної реалізації алгоритмів сортування та пошуку даних вони повинні бути добре структурованими з детермінованим переміщенням даних, ґрунтуватися на однотипних операціях з регулярними та локальними зв’язками.

6. Проаналізовано сучасну елементну базу та засоби розробки, які використовується для створення програмно-апаратних засобів інформаційних технологій сортування та пошук даних у реальному часі. Запропоновано для виконання сортування та пошуку даних використовувати графічні процесори (GPU), які відносяться до класу SIMD та програмовані логічні інтегральні схеми (ПЛІС).

7. Показано, що основними перевагами використання ПЛІС є: доступність, висока швидкодія, надійність, універсальність, різноманітність у виборі напруг живлення і параметрів сигналів вводу/виводу, низьке енергоспоживання, наявність засобів автоматизованого проектування, які зменшують час проектування та відлагодження апаратних засобів.

8. На основі проведеного аналізу існуючих інформаційних технологій паралельного сортування та пошуку даних було сформована основні завдання для вирішення актуального наукового завдання розроблення нових і вдосконалення існуючий методів, моделей та засобів інформаційних технологій паралельного сортування і пошуку даних у реальному часі з високою ефективністю використання обладнання.

РОЗДІЛ 2. РОЗРОБКА МЕТОДІВ І ФУНКЦІОНАЛЬНИХ МОДЕЛЕЙ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ПАРАЛЕЛЬНОГО СОРТУВАННЯ ДАНИХ

У розділі сформульовано вимоги до засобів інформаційних технологій паралельного сортування та пошуку даних, основною з яких є забезпечення реального часу. Вдосконалено та орієнтовано метод сортування масиву чисел вставкою на паралельне та паралельно-потокове сортування одновимірного масиву. Вдосконалено та орієнтовано метод злиття на паралельне та паралельно-потокове сортування масивів даних. Розроблено функціональні моделі алгоритмів паралельного сортування потоків даних у реальному часі.

2.1. Формулювання вимог і вибір принципів побудови компонентів інформаційних технологій паралельного сортування та пошуку даних

Сучасний етап розвитку інформаційних технологій характеризується накопиченням великих масивів даних. Опрацювання таких масивів найчастіше вимагає використання операцій сортування та пошуку даних, метою яких є пришвидшення пошуку необхідної інформації. Ключовою вимогою, що ставиться до засобів сортування і пошуку даних є забезпечення високої швидкодії. Така проблема виникає, як правило, при сортуванні великих масивів даних і пошуку в реальному часі.

Тому для вирішення поставлених у дисертаційній роботі завдань перш за все необхідно сформулювати вимоги до компонентів інформаційних технологій паралельного сортування та пошуку даних, а саме до засобів паралельного сортування та пошуку даних. Основною вимогою з яких є забезпечення реального часу.

Для забезпечення режиму реального часу під час програмно-апаратного сортування та пошуку даних необхідно, щоб виконувалася така умова:

$$t_{сор/пош} \le t_{пм}\,, \tag{2.1}$$

де $t_{сор/пош}$ – час сортування/пошуку даних; $t_{пм}$ – час надходження масиву даних, який визначено так:

$$t_{пм} = \frac{Nn}{F_d k n_k}, \tag{2.2}$$

де N – розмір масиву даних; n – розрядність даних; F_d – частота надходження даних; k – кількість каналів; n_k – розрядність каналів надходження даних.

В свою чергу продуктивність програмно-апаратного засобів сортування та пошуку даних повинна бути:

$$П_{КЗ} = \frac{\lambda R_{сор/пош}}{t_{пм}}, \tag{2.3}$$

де λ – коефіцієнт, який враховує особливості архітектури комп'ютерних засобів; $R_{сор/пош}$ – кількість операцій, необхідна для виконання алгоритму сортування (пошуку) даних.

При паралельно-потоковій апаратній реалізації алгоритмів сортування та пошуку даних режим реального часу забезпечується, коли виконується наступна умова:

$$P_d \le P_{АЗ}, \tag{2.4}$$

де P_d – інтенсивність надходження даних; $P_{АЗ}$ – інтенсивність сортування (пошуку) даних апаратними засобами.

Інтенсивність надходження даних та інтенсивність сортування (пошуку) даних апаратними засобами обчислюється так:

$$P_d = F_d k n_k, \; P_{АЗ} = F_d m_t n_t, \tag{2.5}$$

де m_t – кількість трактів сортування (пошуку) даних; n_t – розрядність траків сортування (пошуку) даних.

Також апаратні засобів інформаційних технологій паралельного сортування та пошуку даних повинні забезпечувати високу ефективність використання обладнання, яку обчислюють так:

$$E = \frac{R_{сор/пош}}{t_{сор/пош} W_{сор/пош}}, \tag{2.6}$$

де $W_{сор/пош}$ – затрати обладнання на реалізацію апаратних засобів сортування/пошуку даних.

Ефективність використання обладнання зв’язує продуктивність апаратних засобів сортування/пошуку даних з витратами обладнання та оцінює елементи засобів за продуктивністю.

Як вже було сказано вище, основним шляхом підвищення швидкості сортування при опрацюванні великих масивів даних є використання розпаралелювання процесу сортування масивів даних та використання багатопроцесорних багатоядерних систем із великим обсягом пам’яті. До даних засобів можна віднести графічні процесори (GPU – Graphics Processing Unit), які являються процесорами класу SIMD (Single Instruction Multiple Data). Ключовою характеристикою процесорів класу SIMD є те, що одна операція застосовується для опрацювання множини незалежних даних. Програмна реалізація операцій пошуку та сортування масивів даних на багатопроцесорних багатоядерних систем із великим обсягом пам’яті вимагає вдосконалення та розроблення нових паралельних алгоритмів. Паралельні методи і алгоритми пошуку та сортування масивів даних для реалізації на багатопроцесорних багатоядерних систем повинні:

- бути добре структурованими з детермінованим переміщенням даних;
- ґрунтуватися на однотипних операціях попарного порівняння та перестановки даних з регулярними та локальними зв’язками;
- бути орієнтованими на реалізацію на множині взаємозв’язаних процесорних ядр;
- використовувати конвеєризацію та просторовий паралелізм.

При розробці паралельних алгоритмів сортування та пошуку даних необхідно враховувати одночасно багато взаємопов’язаних факторів. Передусім необхідно, щоб алгоритми були рекурсивними, локально залежними та орієнтованими на багатопроцесорні багатоядерні системи.

При розробці високоефективних засобів паралельного пошуку та сортування масивів даних пропонується використовувати інтегрований підхід, який охоплює:

- дослідження, розробку методів і алгоритмів паралельного пошуку та сортування масивів даних;

- пошук нових алгоритмічних рішень, орієнтованих на багатопроцесорні багатоядерні системи, які забезпечать мінімальний час сортування та пошуку даних;
- засоби автоматизації паралельного програмування.

Для забезпечення високої ефективності пошуку та сортування даних при розробці засобів пропонується використовувати наступні принципи:

- адаптації алгоритмів пошуку та сортування даних до структури багатопроцесорної багатоядерної системи;
- широкого використання конвеєризації і просторового паралелізму при розроблені засобів пошуку та сортування даних;
- модульності програмно-апаратних засобів.

За сформульованими принципами будемо розробляти компоненти інформаційних технологій паралельного сортування та пошуку даних.

2.2. Вибір форми відображення алгоритмів пошуку та сортування масивів даних

Для оцінки обчислювальних і структурних характеристик алгоритмів пошуку та сортування масивів даних використовується їх подання у вигляді функціональних моделей $F=(\Phi,\Gamma)$, де $\Phi=\{\Phi_1,\Phi_2,\ldots,\Phi_n\}$ – множина функціональних операторів; Γ – закон відображення зв'язків між операторами. Функціональна модель алгоритму відображає послідовність і взаємну залежність функціональних операторів. Графічно функціональна модель алгоритму відображається у вигляді вершин, що відповідають операторам алгоритму Φ_i та дуг, які відображають зв'язки між операторами. Складність функціональних операторів Φ_i визначається як структурними одиницями інформації, так і складністю виконуваних операцій. У більшості випадків функціональні оператори Φ_i зводяться до операцій попарного порівняння та перестановки даних. Подання алгоритму сортування масивів даних у вигляді функціонального графу не в повній мірі відображає просторово-часові залежності між функціональними операторами [3].

Виявити паралелізм алгоритмів пошуку та сортування масивів даних, управляти ним для знаходження оптимальних просторово-часових рішень забезпечує

подання функціонального графу в ярусно-паралельній формі (ЯПФ). При такій формі подання алгоритму здійснюється розподіл всіх його функціональних операторів Φ_i за ярусами таким чином, що в j-му ярусі розміщені функціональні оператори, які залежать хоча б від одного функціонального оператора (j-1)-го ярусу і не залежать від операторів наступних ярусів. В середині ярусу функціональні оператори між собою не мають з'єднань [3].

Кожний j-й ярус алгоритму описується наступними параметрами [3]:

- наборами незалежних функціональних операторів Φ_{ji}, де j - номер яруса; i-номер функціонального оператора в ярусі;
- набором каналів надходження даних і видачі проміжних результатів;
- розрядністю кожного каналу зв'язку.

Кількість ярусів у ЯПФ алгоритму є її висотою h, а максимальна кількість функціональних операторів у ярусі визначає її ширину L. За допомогою просторово-часових індексів у системі координат час-простір задається розміщення функціональних операторів. Таке відображення орієнтованого графа алгоритму будемо називати потоковою паралельною формою або потоковим графом [3,5,6]. Параметрами потокового графу є: складність функціональних операторів Φ_{ji}, ширина L і висота h. Дані параметри є взаємно залежними, зміна одного з них веде до зміни інших.

Для переходу від функціонального графу пошуку та сортування масивів даних до відображення його у вигляді потокового графу необхідно його записати в вигляді матриці інциде́нтності $n \times n$, де "1" – відповідає наявності каналу зв'язку, "0" – відсутності зв'язку між функціональними операторами [5]. Матриця формується таким чином, що для кожного функціонального оператора Φ_{ji} джерела інформації формується стрічка, яка відображає його зв'язки з іншими функціональними операторами. Якщо позначити через вектори $\vec{V}_{\Phi 1},...,\vec{V}_{\Phi n}$ відповідні стовпці матриці, то можна визначити результуючий вектор стовпець [3]:

$$\vec{V}_0 = \vec{V}_{\Phi 1} + \vec{V}_{\Phi 2} + ... + \vec{V}_{\Phi n} \qquad (2.7)$$

В векторі $\vec{V}_0$ визначаємо номера елементів, які дорівнюють нулю, наприклад, другий та п’ятий. За визначеними номерами знаходимо функціональні оператори, що не мають нащадків і відповідно утворюють нульовий ярус, в даному випадку Φ_2 і Φ_5. Далі за формулою обчислюємо [3]:

$$\vec{V}_1 = \vec{V}_0 - \vec{V}_{\Phi 2} - \vec{V}_{\Phi 5}. \tag{2.8}$$

У векторі $\vec{V}_1$ знаходимо номери нульових елементів, за якими визначаємо функціональні оператори, що утворюють перший ярус. Аналогічно обчислюємо наступні вектори та визначаємо функціональні оператори, що утворюють наступні яруси [3].

Розробку паралельних алгоритмів пошуку та сортування масивів даних будемо виконувати для двох реалізацій:

1) з використанням багатопроцесорної багатоядерної системи та врахуванням особливостей її архітектури;

2) апаратної реалізації алгоритмів пошуку та сортування масивів даних у реальному часі.

Для таких реалізацій алгоритми пошуку та сортування масивів даних доцільно подавати у вигляді потокових графів. При розробці потокових графів необхідно враховувати архітектуру багатопроцесорної багатоядерної системи, величину масиву та інтенсивність надходження даних.

2.3. Виділення базових операцій для паралельного сортування даних

Значна частина методів сортування ґрунтується на використанні базової операції попарного порівняння та перестановки даних. Цілеспрямоване використання даної базової операції забезпечує виконання алгоритмів сортування даних. Методи сортування даних в основному відрізняються вибором пар даних для порівняння. Базова операція попарного порівняння та перестановки даних, яка використовується для сортування масивів даних виконується так:

$$b_{\text{max}} = \begin{cases} a_1, \textit{коли } a_1 > a_2 \\ a_2, \textit{коли } a_1 \le a_2 \end{cases}, \tag{2.9}$$

$$b_{\min} = \begin{cases} a_1, \text{коли } a_1 \le a_2 \\ a_2, \text{коли } a_1 > a_2 \end{cases}. \qquad (2.10)$$

Граф схема виконання базової операції попарного порівняння та перестановки даних наведена на рис.2.1, де $Ф_1$ – оператор попарного порівняння; $Ф_2$ – оператор комутації (перестановки даних).

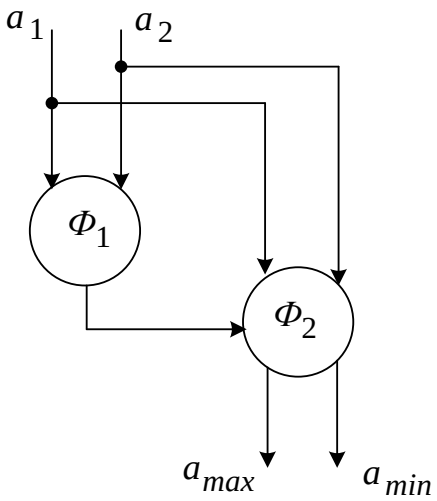


Рис. 2.1. Базова операція попарного порівняння та перестановки даних

У випадку, коли для сортування використовується на m, де $m>1$, обчислюваних елементах (процесорах або процесорних ядрах) масив даних розділяється на m частин (блоків), які опрацьовуються паралельно.

Базова операція сортування масиву чисел методом злиттям з використанням одного обчислювального елемента. Паралельне сортування масивів чисел методом злиття ґрунтується на базовій операції об'єднання двох упорядкованих масивів чисел $\{a_{1i}\}_{i=1}^{p}$ та $\{a_{2i}\}_{i=1}^{p}$ у один упорядкований масив $\{a_i\}_{i=1}^{2p}$ чисел. Таке об'єднання з використанням одного обчислювального елемента вимагає виконання $2p$ базових операції попарного порівняння та перестановки даних. Граф схема виконання базової операції об'єднання двох упорядкованих масивів чисел з використанням одного обчислювального елемента наведена на рис.2.2, де $Ф_{У1}$, $Ф_{У2}$ – оператори управління процесом сортування відповідно для першого та другого масивів; $Ф_{Пр1}$, $Ф_{Пр2}$ – оператори відповідно першої та другої пам'яті розміром p елементів; $Ф_{П1}$ – оператор пам'яті першого елемента; $Ф_{П2}$ – оператор пам'яті другого елемента; $Ф_{Пор}$ – оператор

порівняння даних; $Ф_{Км}$ – оператор комутації (перестановки даних); $Ф_{П2р}$ – оператори пам’яті розміром $2р$ елементів.

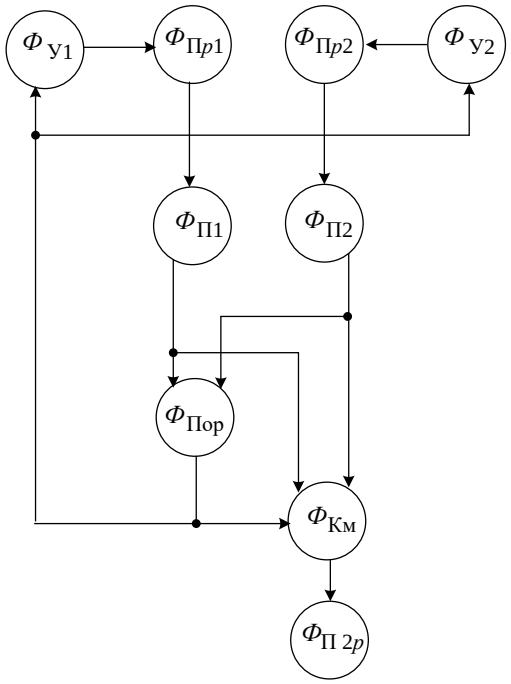


Рис.2.2. Граф схема виконання базової операції об’єднання двох упорядкованих масивів чисел з використанням одного обчислювального елемента

Приклад, об’єднання двох упорядкованих масивів із чотирьох чисел з використанням одного обчислювального елемента наведено в табл. 2.1.

Табл. 2.1

Об’єднання двох упорядкованих масивів із чотирьох чисел

$П_{Р1}$	$П_{Р2}$
10	9
7	4
5	2
3	1

$П_{2р}$
10
9
7

5
4
3
2

Об’єднання двох упорядкованих масивів обсягом p чисел з використанням одного обчислювального елемента виконується за $2p$ тактів.

Вдосконалена базова операція об’єднання двох упорядкованих масивів з використанням двох обчислювальних елементів. В основі вдосконаленого паралельного методу злиття, лежить ідея зменшення кількості тактів які необхідні для отримання відсортованого масиву $\{a_i\}_{i=1}^{2p}$ чисел із двох відсортованих масивів чисел $\{a_{1i}\}_{i=1}^{p}$ та $\{a_{2i}\}_{i=1}^{p}$. Для зменшення часу об’єднання двох упорядкованих масивів чисел $\{a_{1i}\}_{i=1}^{p}$ та $\{a_{2i}\}_{i=1}^{p}$ було запропоновано розпаралелити процес об’єднання підмасивів за рахунок одночасного злиття двох масивів з використанням двох обчислювальних елементів починаючи як з максимальних значень, так і мінімальних значень. Отримуємо відсортований масив із $2p$ чисел на двох виходах: на першому виході (p_{1max} найбільших чисел), на другому виході (p_{2min} найменших чисел). Кількість операцій попарного порівняння (тактів роботи), які необхідно виконати для отримання p_{1max} найбільших чисел і p_{2min} найменших чисел дорівнює p, що в порівнянні з існуючими алгоритмами об’єднання, є два рази менше.

У основі вдосконаленої базової операції об’єднання двох упорядкованих масивів чисел лежить використання елементарних операціях – попарного порівняння елементів та їх перестановка. Для об’єднання двох підмасивів чисел починаючи з максимальних використовуються наступні операції:

$$y_{\max i} = \begin{cases} 0, \textit{коли } a_{1\max i} \le a_{2\max i} \\ 1, \textit{коли } a_{1\max i} > a_{2\max i} \end{cases}, \tag{2.11}$$

$$a_{C\max i} = \begin{cases} a_{1\max i}, \textit{коли } y_{\max i} = 1 \\ a_{2\max i}, \textit{коли } y_{\max i} = 0 \end{cases}, \tag{2.12}$$

а для об’єднання підмасивів чисел починаючи з мінімальних наступні:

$$y_{\min i} = \begin{cases} 0, \textit{коли } a_{1\min i} \le a_{2\min i} \\ 1, \textit{коли } a_{1\min i} > a_{2\min i} \end{cases}, \tag{2.13}$$

$$a_{C\min i} = \begin{cases} a_{1\min i}, \textit{коли } y_{\min i} = 0 \\ a_{2\min i}, \textit{коли } y_{\min i} = 1 \end{cases}, \tag{2.14}$$

де $y_{\max i}$, $y_{\min i}$ – результати попарних порівнянь елементів максимальних і мінімальних масивів; $a_{1\max i}$, $a_{2\max i}$ – i-і елементи порівнянь максимальних значень з відповідного першого та другого підмасивів; $a_{1\min i}$, $a_{2\min i}$ – i-і елементи порівнянь мінімальних значень відповідного першого та другого підмасивів; $a_{C\max i}$, $a_{C\min i}$ – i-і відсортовані елементи масивів максимальних чисел і мінімальних чисел.

Для виконання вдосконаленої базової операції об’єднання двох упорядкованих масивів розміром p чисел у один масив, де p старших чисел отримуємо на першому виході, а p молодших чисел отримуємо на другому виході, необхідно використати два процесори (процесорні ядра) та чотири блоки пом’яті.

Граф схема виконання базової операції об’єднання двох упорядкованих масивів чисел з використанням двох обчислювальних елементів наведена на рис.2.3, де $Ф_{У1}$, $Ф_{У2}$, $Ф_{У3}$, $Ф_{У4}$ – оператори управління процесом сортування відповідно для першого, другого, третього та четвертого масивів; $Ф_{Пр1}$, $Ф_{Пр2}$, $Ф_{Пр3}$, $Ф_{Пр4}$ – оператори відповідно першої, другої, третьої та четвертої пам’яті розміром p елементів; $Ф_{П1}$, $Ф_{П2}$, $Ф_{П3}$, $Ф_{П4}$ – оператори пам’яті відповідно першого, другого, третього та четвертого елементів; $Ф_{Км1}$, $Ф_{Км2}$ – перший та другий оператори комутації; $Ф_{Пр\ max}$ – оператор пам’яті для p максимальних чисел; $Ф_{Пр\ max}$ – оператор пам’яті для p мінімальних чисел.

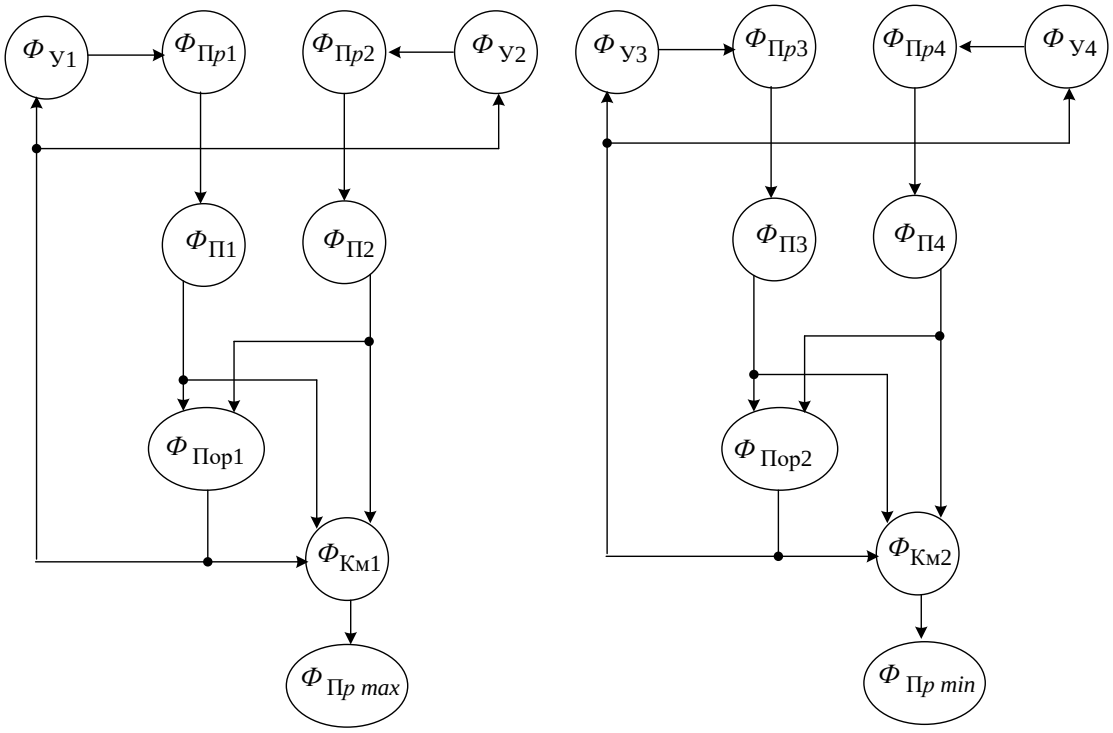


Рис.2.3. Граф схема виконання базової операції об’єднання двох упорядкованих масивів чисел з використанням двох обчислювальних елементів

Алгоритм виконання вдосконаленої базової операції об’єднання двох упорядкованих масивів реалізується в два етапи. На першому етапі виконується запис першого p_1 відсортованого масиву паралельно у перший і третій блоки пам’яті, а другого p_2 відсортованого масиву - у другий і четвертий боки пам’яті.

На другому етапі виконання вдосконаленої базової операції здійснюється злиття масивів даних записаних у першому та другому блоках пам’яті за спаданням, а в третьому та четвертому блоках пам’яті за зростанням.

Блок схема алгоритму виконання вдосконаленої базової операції об’єднання двох упорядкованих масивів чисел наведена на рис.2.4, де БП – блок пам’яті; Рг – регістри; Лч$_ц$ - лічильник циклів; Вх*ТІ* – вхід тактових імпульсів.

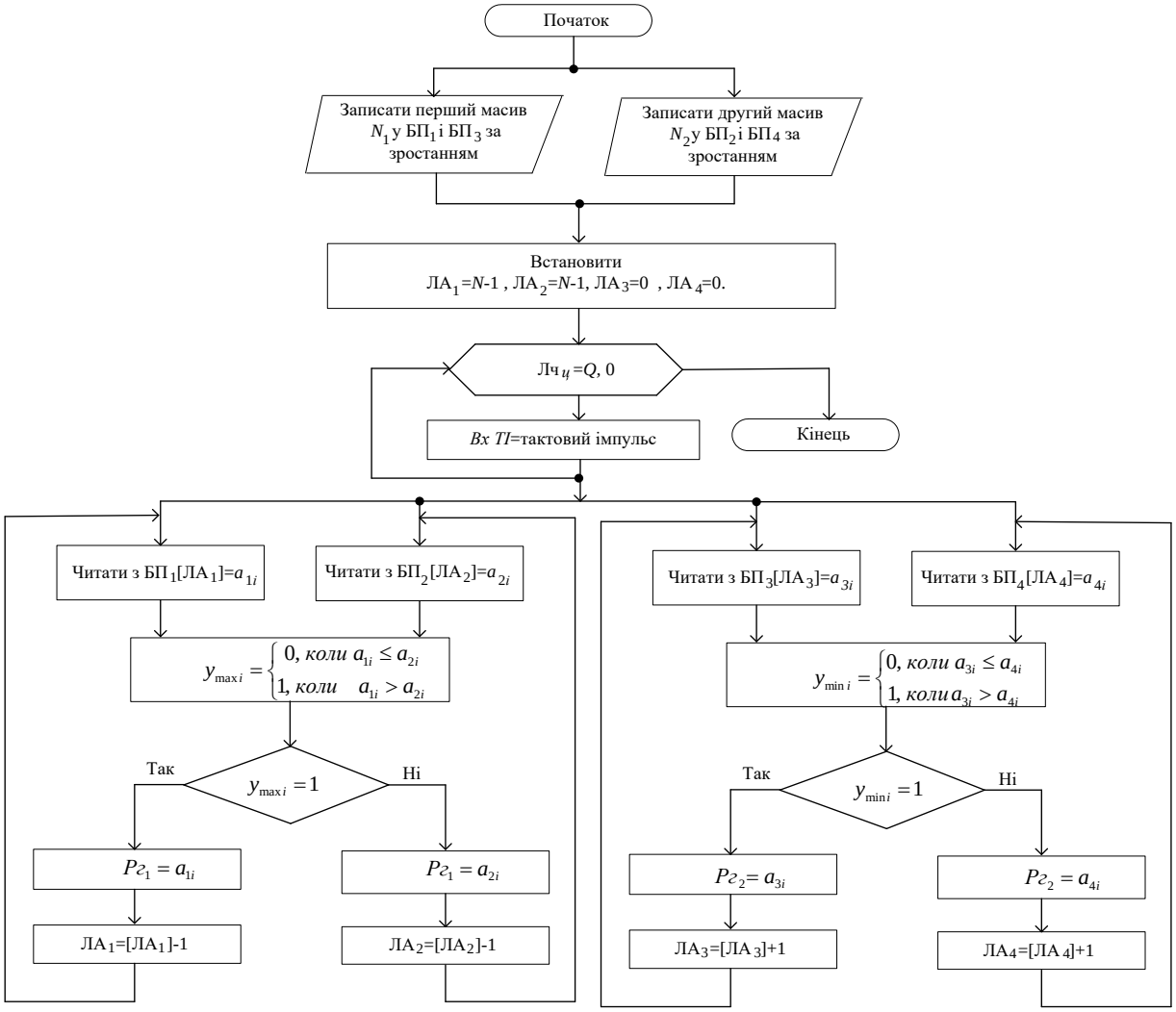


Рис.2.4. Блок схема алгоритму виконання вдосконаленої базової операції об’єднання двох упорядкованих масивів чисел

Приклад, об’єднання двох упорядкованих масивів із чотирьох чисел з використанням двох обчислювальних елементів наведено в табл. 2.2.

Табл. 2.2

Об’єднання двох упорядкованих масивів із чотирьох чисел

$П_{Р1}$	$П_{Р2}$	$П_{Р3}$	$П_{Р4}$
10	9	10	9
7	4	7	4
5	2	5	2
3	1	3	1

$П_{P\ max}$	$П_{P\ min}$
10	1
9	2
7	3
5	4

Розроблена вдосконалена базова операція об’єднання двох упорядкованих масивів чисел використовується для реалізації паралельного сортування методом злиття m упорядкованих масивів чисел довжиною p чисел. Особливість виконання даної базової операції є об’єднання в часі процесу запису їх у наступні блоки пам’яті з виводом відсортованих чисел. Результатом паралельного сортування m упорядкованих масивів чисел довжиною p чисел є відсортований масив розміром $m\times p$ чисел, найбільші p числа отримуємо при першому вході послідовно, а найменші N числа - на m виході.

2.4. Методи паралельного та паралельно-потокового сортування масивів чисел вставкою

Сортування масиву чисел $\{a_i\}_1^m$ методом вставки полягає в отримані нового масиву чисел $\{b_i\}_1^m$, який складаються із чисел a_i, переставлених в необхідному порядку. Сортування масиву чисел $\{a_i\}_1^m$ методом вставки полягає в тому, що масив розбивається на дві частини: відсортовану та невідсортовану. Спочатку відсортована частина містить тільки перше число a_1, яке само по собі є впорядкований. На кожному j кроці сортування беремо a_{i+1} число з невідсортованої частини та вставляємо його до відсортованої частини так, щоб вона не втратила впорядкованості [5].

Базова операція Φ_{ji} алгоритму сортування масиву чисел $\{a_i\}_1^m$ методом вставки зводиться до виконання двох елементарних операцій: попарного порівняння числа

a_{i+1} з i-м числом b_{ji} відсортованої частини та перестановки чисел. Попарне порівняння чисел здійснюється так [5]:

$$y_{ji} = \begin{cases} 0, при & a_{i+1} > b_{ji} \\ 1, при & a_{i+1} \le b_{ji} \end{cases} . \qquad (2.15)$$

Результати попарного порівняння числа y_{ji} використовуються для перестановки чисел і формування виходу базової операції у відповідності з наступним виразом:

$$b_{ji} = \begin{cases} b_{(j-1)i-1,} \text{ коли } y_{j(i-1)} = y_{ji} = 0 \\ a_{i+1}, \text{ коли } y_{j(i-1)} = 1, y_{ji} = 0 \\ b_{(j-1)k}, \text{ коли } y_{j(i-1)} = y_{ji} = 1 \end{cases}, \qquad (2.16)$$

де $b_{(j\text{-}1)i\text{-}1}$, $b_{(j\text{-}1)i}$ – числа з виходів відповідно $\Phi_{(j\text{-}1)i\text{-}1}$, $\Phi_{(j\text{-}1)i}$; $y_{j(i\text{-}1)}$ – результат порівняння з виходів $\Phi_{j(i\text{-}1)}$.

Потоковий граф алгоритму паралельного сортування методом вставки, наведений на рис.2.5. де Φ_{ji} – функціональний оператор попарного порівняння та перестановки даних.

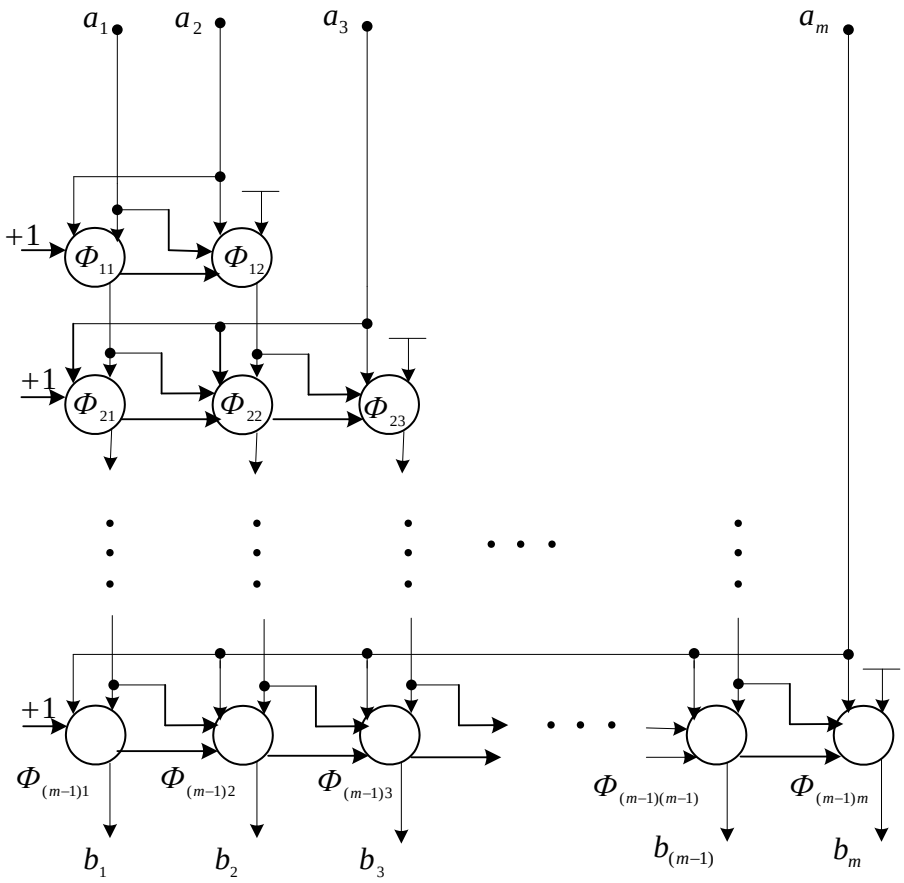


Рис.2.5. Потоковий граф алгоритму паралельного сортування чисел методом вставки

Потоковий граф алгоритму паралельного сортування методом вставки масиву із m чисел має висоту $h=m$-1, ширину $L=m$. Складність паралельного алгоритму сортування чисел методом вставки рівна $R=(\frac{m^2+m}{2}-1)$ базових операцій [5].

Проекція потокового графу алгоритму паралельного сортування методом вставки масиву на горизонтальну вісь X для m чисел, наведена на рис.2.6., де $Ф_i$ – функціональний оператор перестановки даних та попарного порівняння; $Ф_{МЗП}$ – макрооператор перестановки та затримки; $Ф_{МУ}$ – макрооператор управління.

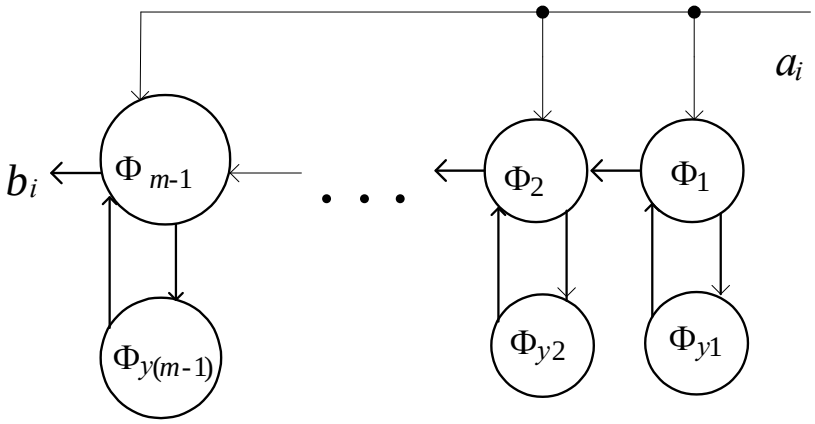


Рис.2.6. Лінійна проекція графа алгоритму сортування чисел методом вставки на горизонтальну вісь X

Складність даного алгоритму рівна $R=(m)$ базових операцій. Інтенсивність сортування масивів даних при лінійній проекції на горизонтальну вісь X дорівнює

$$D_c=\frac{n}{T_к}, \tag{2.17}$$

де n – розрядність чисел сортування.

Другим варіантом отримання потокового графу сортування даних методом вставки є його лінійна проекція на вертикальну вісь Y. При даному варіанті укрупнення операцій виконується шляхом об’єднання у межах ярусу функціональних операторів і каналів передачі даних. При такому укрупнені операцій, отримуємо конкретизований потоковий граф з одним каналом передачі чисел [5].

Проекція потокового графу алгоритму паралельного сортування методом вставки на вертикальну вісь *Y* основі, наведена на рис.2.7., де $Ф_1$ – $Ф_{m\text{-}1}$ – базові операції; $Ф_{y1}$ – $Ф_{y(m\text{-}1)}$ – оператори управління.

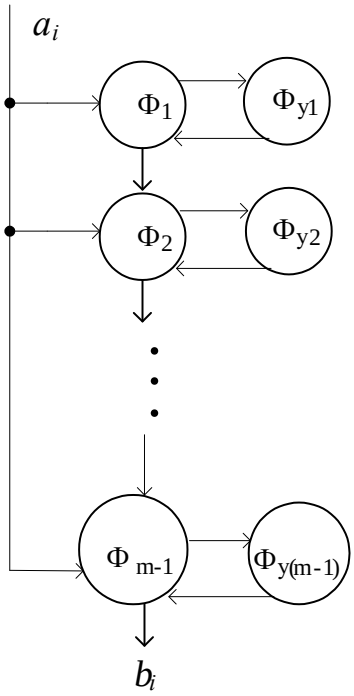


Рис.2.7. Лінійна проекція графа сортування чисел на вертикальну вісь *Y*

Складність даного алгоритму є рівною складності алгоритму наведеному на рис.2.6. Реалізація таких алгоритмів забезпечує однакову інтенсивність сортування.

Третій варіант отримання конкретизованого графу сортування чисел методом вставки зв'язаний з підвищенням інтенсивності сортування масивів чисел, яке досягається збільшенням кількості каналів надходження чисел. Даний варіант конкретизованого графу є паралельно-потоковим і отримується шляхом лінійної проекції на вертикальну вісь *Y*. Паралельно-потоковий граф сортування масиву чисел наведений на рис.2.8, де $Ф_c$ – оператор сортування; $Ф_1$ – $Ф_m$ – оператори порівняння; $Ф_{y1}$ – $Ф_{ym}$ – оператори управління; $Ф_{к1}$ – $Ф_{км}$ – оператори комутації.

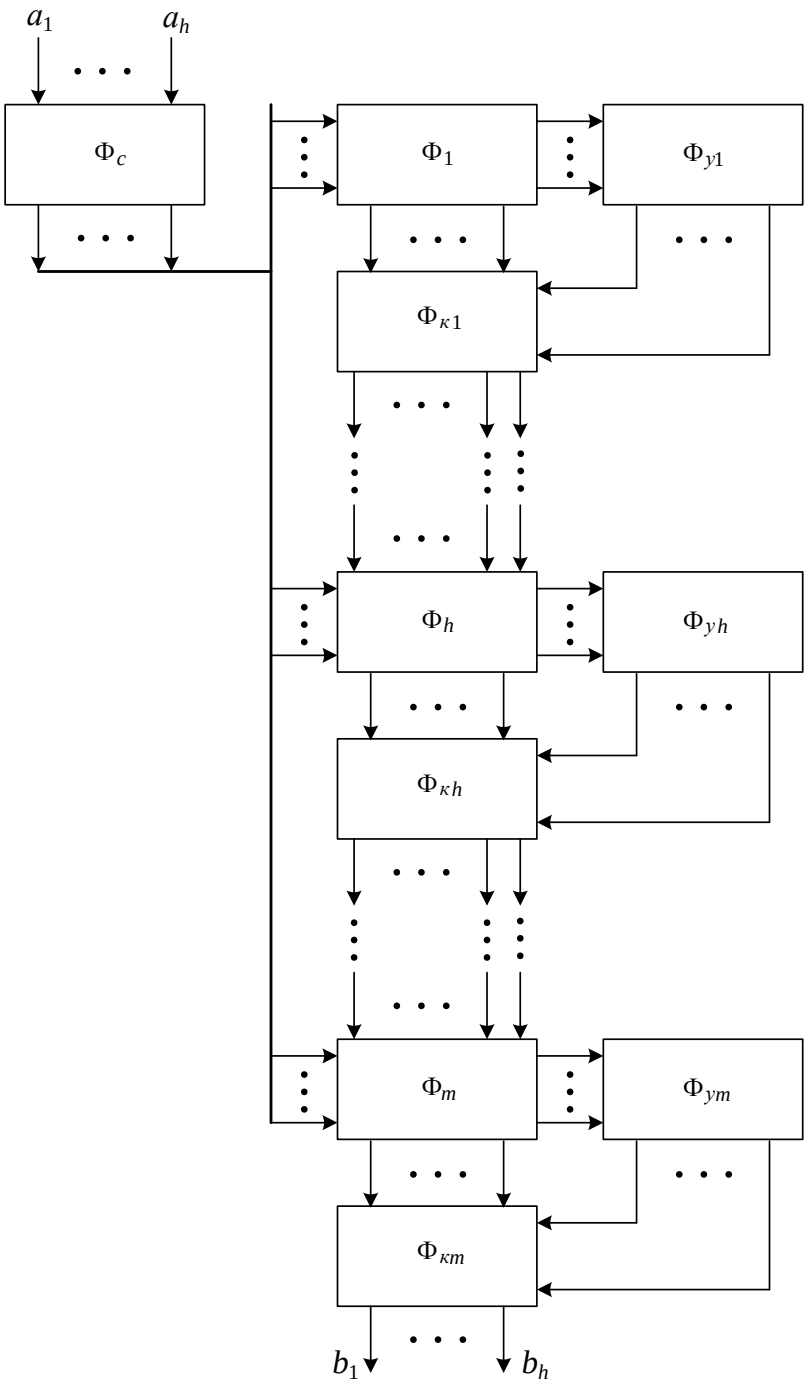


Рис.2.8. Паралельно-потоковий граф сортування масиву чисел

Складність паралельно-потокового графу алгоритму сортування масиву чисел рівна $R = (\mathrm{mh})$ базових операцій. Оціночна інтенсивність паралельно-потокового сортування масиву чисел дорівнює:

$$D_c = \frac{hn}{T_к}. \qquad (2.18)$$

Основним шляхом підвищення інтенсивності паралельно-потокового сортування масиву чисел є збільшення кількості каналів [5].

2.5. Метод паралельного сортування даних злиттям

Існуючі паралельні алгоритми сортування методом злиття базуються на операціях перестановки чисел та попарного порівняння, тобто базовою операцією є сортування двох чисел r=2. Критерієм оцінки їхньої ефективності є час виконання та обчислювальна складність. В основі алгоритмів сортування методом злиття з базовою операцією r=2 лежить макрооперація об'єднання двох упорядкованих підмасивів в один впорядкований масив. На початку сортування вхідний масив чисел $\{a_j\}_{j=1}^{m}$ розбивається на $m/2$ груп чисел, в яких виконується об'єднання двох упорядкованих масивів довжиною одиниця (макрооперація першого типу) в один упорядкований масив розміром 2.

Приклад потокового графу алгоритму паралельного сортування методом злиття масиву із 16-и чисел для r=2, наведений на рис.2.9 де Φ_{ji} – функціональний оператор попарного порівняння та перестановки даних.

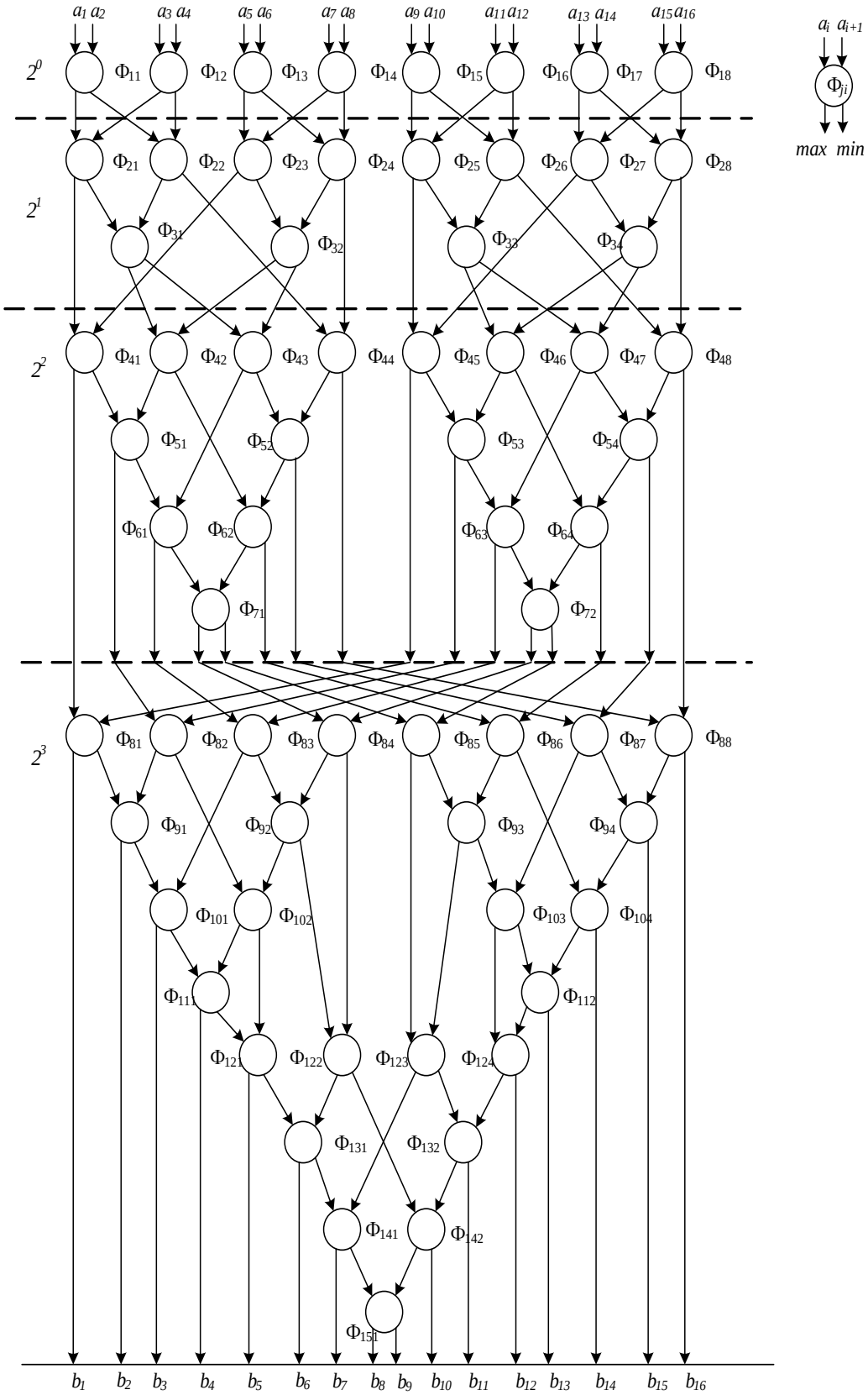


Рис.2.9. Потоковий граф алгоритму паралельного сортування масиву із 16-и чисел методом злиття для r=2

У результаті виконання першого об’єднання формуються $m/2$ груп упорядкованих масивів довжиною два. Кількість об’єднань залежить від розміру масиву m та типу базової операції, тобто від значення r. При використанні базової операції з $r=2$ кількість об’єднань визначається так:

$$L=\log_2 m. \qquad (2.19)$$

Орієнтація алгоритмів сортування на НВІС-реалізацію вимагає зменшення числа виводів інтерфейсу та розрядності з’єднань між функціональними операторами попарного порівняння та перестановки даних Φ_{ji}. Забезпечити ці вимоги можна за рахунок використання методів паралельно-вертикального сортування даних, при яких надходження даних, сортування та видача результатів здійснюється розрядними зрізами [3]. Детальніше вдосконалення та орієнтація методу злиття на паралельно-вертикальне сортування у реальному часі буде описано в розділі 3.1., але для наочності різниці потокових графів методу сортування злиттям та методу злиття при паралельно-вертикальній реалізації було наведено нижче. Приклад, потокового графу алгоритму паралельного сортування методом злиття масиву із 16-и чисел для $r=4$, наведений на рис.2.10 де Φ_{ji} – функціональний оператор паралельно-вертикального сортування чотирьох чисел.

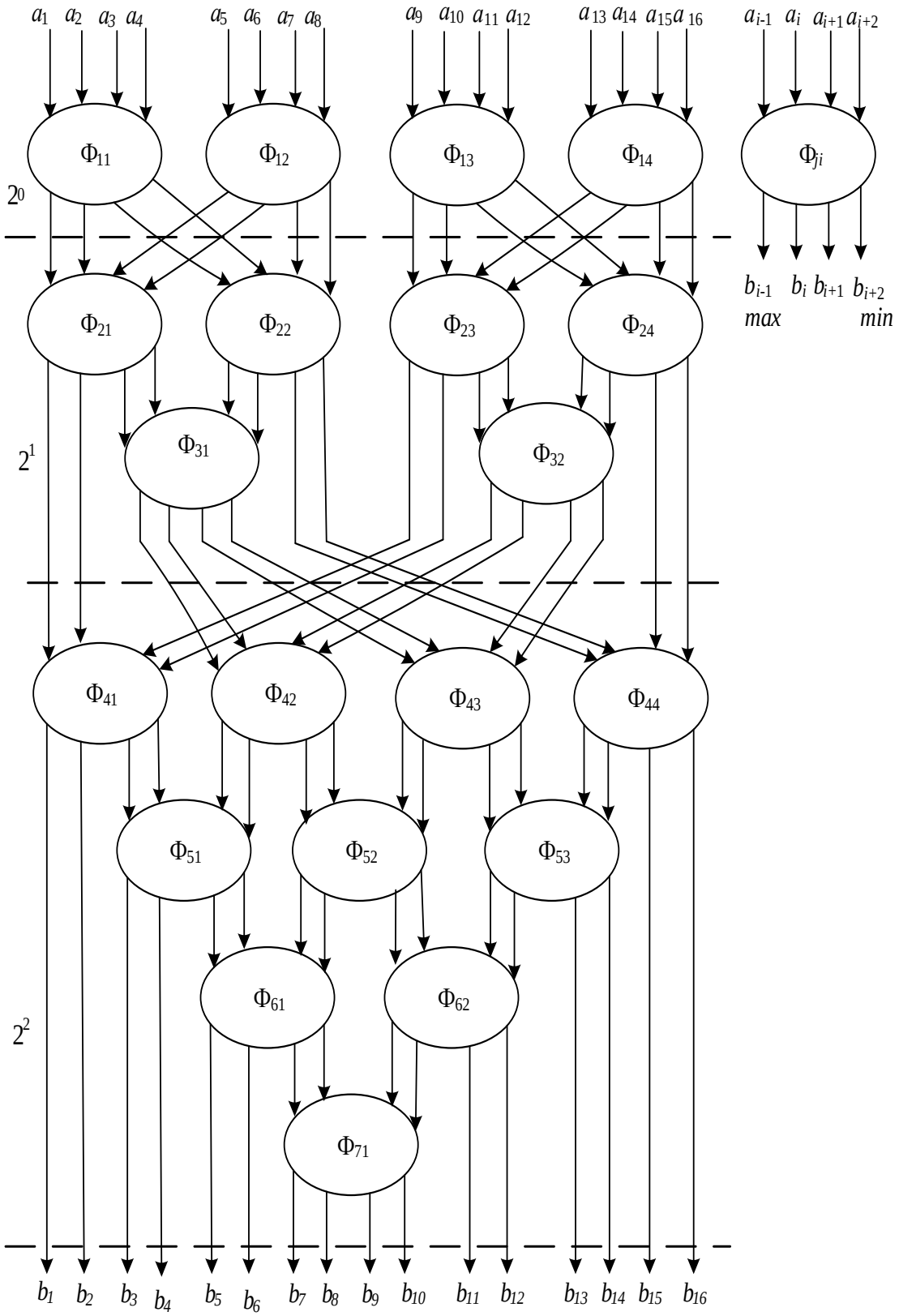


Рис.2.10. Потоковий граф алгоритму паралельного сортування масиву із 16-и чисел методом злиття для r=4

У результаті виконання першого об'єднання формуються $m/4$ груп упорядкованих масивів довжиною чотири. Кількість об'єднань при використанні базової операції з $r=4$ визначається так:

$$L=\log_2 m-\log_2 r/2. \quad (2.20)$$

Паралельне сортування масиву із m чисел методом злиття на основі базової операції з $r=4$ вимагає послідовного проходження даних через $k=\frac{2^{L+1}}{\mathrm{r}}-1=7$. Це приблизно у два рази зменшено затримку проходження даних з входу на вихід.

2.6. Удосконалення методу сортування даних злиттям

Для сортування масиву із $m\times N$ чисел пропонується використати метод сортування злиттям. На початку сортування вхідний масив із $m\times N$ чисел розбивається на m масивів розміром N чисел. Для сортування масиву із N чисел методом злиття кожне число подається, як упорядкований масив розміром одиниця. Сортування такого масиву методом злиття ґрунтується на операціях послідовного об'єднання упорядкованих масивів. У результаті виконання першого етапу об'єднання формуються $N/2$ впорядкованих масивів довжиною два. Кількість етапів об'єднання для отримання упорядкованих масивів довжиною N визначається так:

$$p=\lceil \log_2 N \rceil \quad (2.21)$$

де $\lceil\ \rceil$ - більше ціле число.

Сортування m упорядкованих масивів довжиною N будемо виконувати паралельним методом злиття, в основі якого лежить базова операція об'єднання двох упорядкованих масивів $\{a_{1i}\}_{i=1}^{N}$ та $\{a_{2i}\}_{i=1}^{N}$ у один упорядкований масив $\{b_{1i}\}_{i=1}^{2N}$. На першому етапі сортування вхідний масив чисел $\{a_j\}_{j=1}^{mN}$ розбивається на $m/2$ пар упорядкованих масивів довжиною N, які попарно об'єднуються. Таке об'єднання реалізується на основі першого типу макрооперації (базової операції). У результаті виконання першого етапу формуються $m/4$ впорядкованих масивів довжиною $2N$. Для виконання другого етапу використовується макрооперація другого типу – об'єднання

двох упорядкованих масивів $\{a_{1i}\}_{i=1}^{2N}$ та $\{a_{2i}\}_{i=1}^{2N}$ у один упорядкований масив $\{b_{1i}\}_{i=1}^{4N}$. Така макрооперація виконується на базі трьох макроопераций першого типу.

Кількість етапів необхідних для сортування масиву із m масивів чисел визначається за формулою:

$$k = \lceil log_2 m \rceil \qquad (2.22).$$

Кожний s-й етап, де s=1,…,k, реалізується на основі макроопераций s-го типу, кожна з яких виконує об'єднання двох упорядкованих масивів $\{a_{1i}\}_{i=1}^{2^{s-1}N}$ та $\{a_{2i}\}_{i=1}^{2^{s-1}N}$ у один упорядкований масив $\{b_{1i}\}_{i=1}^{2^{s}N}$. Макрооперація s-го типу ґрунтується на трьох макрооперациях (s-1)-го типу. Основою всіх макроопераций є базова операція об'єднання двох упорядкованих масивів $\{a_{1i}\}_{i=1}^{N}$ та $\{a_{2i}\}_{i=1}^{N}$ у один упорядкований масив $\{b_{1i}\}_{i=1}^{2N}$. Кількість базових операцій необхідних для виконання макрооперації s-го типу дорівнює 3^{s-1}.

Алгоритми паралельного сортування масиву із $m \times N$ чисел методом злиття відображається у вигляді потокового графу $F=(\Phi,\Gamma)$, де $\Phi=\{\Phi_1,\Phi_2,\ldots,\Phi_n\}$ – функціональні оператори, які виконують операцію сортування масивів довжиною N і базові операції об'єднання двох упорядкованих масивів $\{a_{1i}\}_{i=1}^{N}$ та $\{a_{2i}\}_{i=1}^{N}$ у один упорядкований масив $\{b_{1i}\}_{i=1}^{2N}$; Γ – закон відображення зв'язків між операторами. Такі потокові графи забезпечують знаходження оптимальних просторово-часових рішень. Графічно потоковий граф алгоритму паралельного сортування методом злиття масиву із $m \times N$ чисел відображається у вигляді вершин, що відповідають операторам алгоритму Φ_i (сортування масивів довжиною N і об'єднання двох упорядкованих масивів довжиною N у один упорядкований масив) та дуг, які відображають зв'язки між операторами. Кожний функціональний оператор об'єднання двох упорядкованих масивів довжиною N у один упорядкований масив має два входи та два виходи. На кожній вхід, послідовно число за числом поступають упорядковані масиви розміром N чисел. На першому та другому виходах функціонального оператора об'єднання

послідовно число за числом формується відповідно N максимальних і N мінімальних чисел об’єднаного упорядкованого масиву.

Потоковий граф алгоритму паралельного сортування масиву із $m \times N$ чисел методом злиття наведений на рис.2.11, де a_{1i}-a_{mi} – m входи даних; Φ_{01}-Φ_{0m} – функціональні оператори сортування масивів чисел довжиною N; $\Phi_{s1} - \Phi_{s(2m/2^s)}$ – макрооперації s-го типу, які виконують злиття двох упорядкованих підмасивів довжиною $2^{s-1}N$ чисел у один довжиною 2^sN чисел.

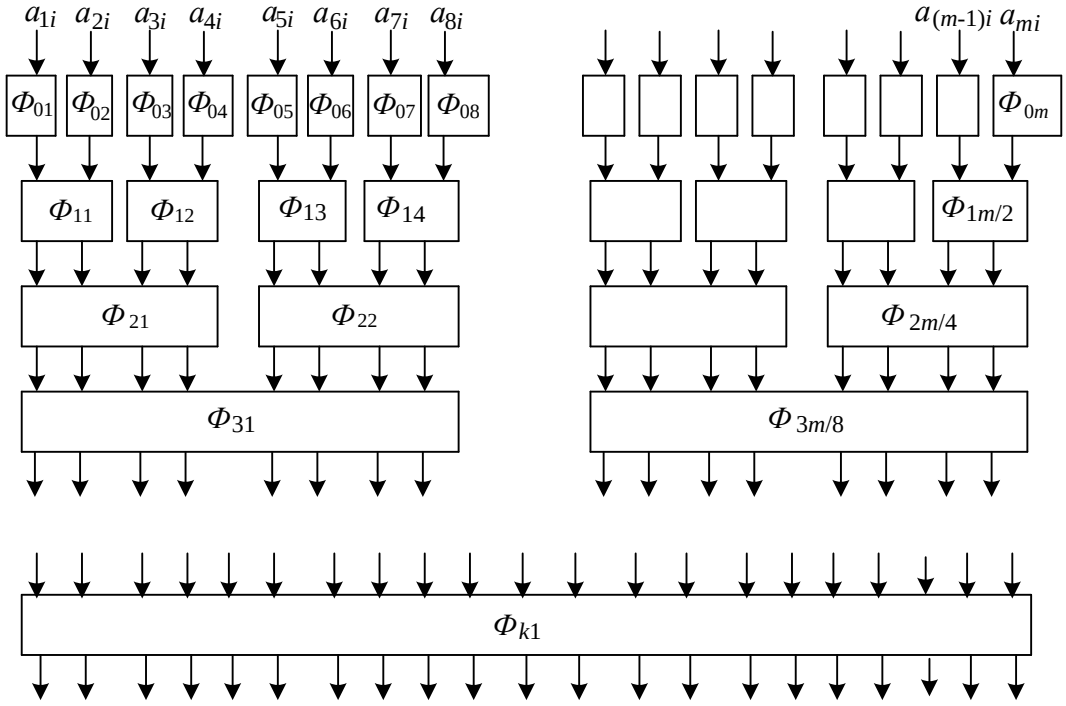


Рис.2.11. Структура потокового графу алгоритму паралельного сортування масиву із $m \times N$ чисел методом злиття

Потоковий граф алгоритму паралельного сортування масиву із $m \times N$ чисел методом злиття складається із (k+1) ярусів. У нульовому ярусі виконуються оператори сортування масивів довжиною N чисел, а у s-х ярусах – макрооперація злиття двох упорядкованих підмасивів довжиною $2^{s-1}N$ чисел у один довжиною 2^sN чисел. У s-х ярусах виконується об’єднання двох упорядкованих масивів довжиною $2^{s-1}N$ чисел у один довжиною 2^sN чисел, яке реалізується на основні макрооперації s-го типу. Кожна макрооперація s-го типу реалізується на трьох макрооперація (s-1)-го типу (рис.2.12).

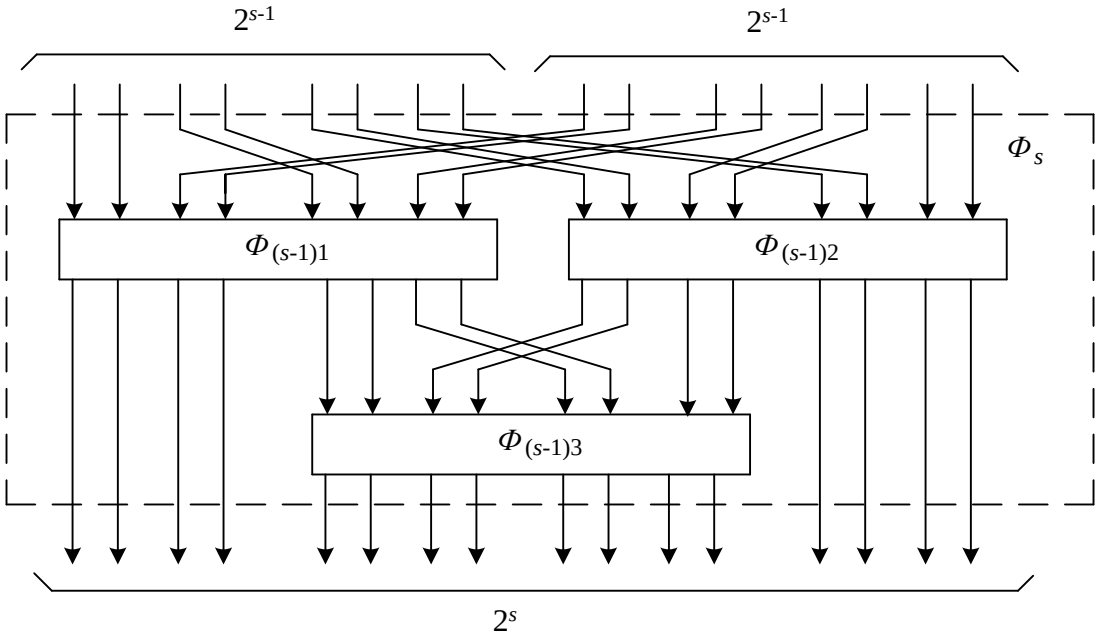


Рис.2.12 Структура реалізації макрооперації s-го типу

Всі макрооперації s-го типу, що використовуються у сортуванні масиву методом злиття виконуються на основі базової операції об'єднання двох упорядкованих підмасивів $\{a_{1i}\}_{i=1}^{N}$ та $\{a_{2i}\}_{i=1}^{N}$ у один упорядкований масив $\{b_{1i}\}_{i=1}^{2N}$. Кількість базових операцій R, яка використовується для реалізації макрооперації s-го типу, визначається так:

$$R = 3^s \tag{2.23}$$

Орієнтований на графічний процесор конкретизований потоковий граф, отримуємо шляхом лінійної проекції потокового графу на горизонтальну вісь X. Проекція конкретизованого графу алгоритму паралельного сортування масиву із $m \times N$ чисел методом злиття наведена на рис.2.13, де Φ_{cl} – функціональний оператор сортування масиву довжиною N методом злиття (l=1,.., m); $\Phi_{БОh}$ – функціональний оператор базової операції об'єднання (h=1,.., m/2) двох упорядкованих підмасивів довжиною N у один впорядкований масив $2N$; $\Phi_{МЗП}$ – макрооператор перестановки та затримки; $\Phi_{МУ}$ – макрооператор управління.

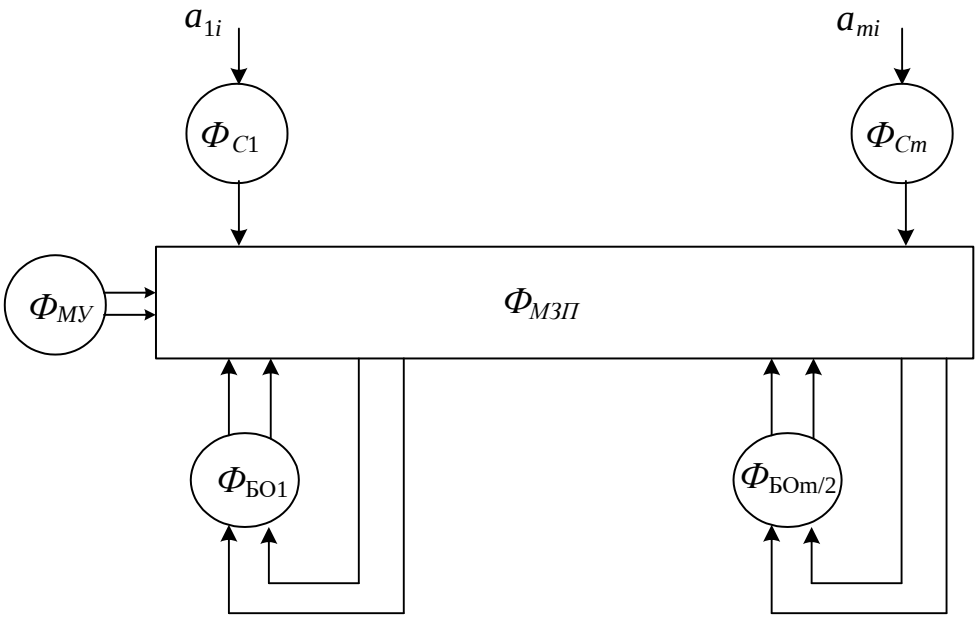


Рис.2.13. Лінійна проекція потокового графу алгоритму сортування злиттям на горизонтальну вісь X

В результаті отримано лінійну проекцію потокового графу алгоритму сортування злиттям на горизонтальну вісь X, яка орієнтована на архітектуру графічного процесора GPU. З допомогою макрооператора управління $Ф_{МУ}$ імакрооператора перестановки та затримки $Ф_{МЗП}$, обчислюється величина перестановки даних, затримки, і здійснюється планування процесу сортування.

2.7. Розробка функціональних моделей паралельного сортування даних

Процес розробки паралельно-потокових графів алгоритмів пошуку та сортування масивів даних можна розбити на такі чотири етапи [3]:

1) декомпозиція алгоритму сортування масиву даних (пошуку);
2) проектування комунікацій між функціональними операторами;
3) укрупнення функціональних операторів;
4) планування процесу сортування масиву даних (пошуку).

При етапі декомпозиції алгоритм сортування масивів даних $Ф$ розділяється на функціональні оператори $Ф_{ji,}$ попарного порівняння та перестановки даних між, якими створюються зв’язки, які відповідають вказаному алгоритму сортування.

Необхідно зауважити, що переважна більшість методів і алгоритмів сортування масивів даних ґрунтуються на використанні базової операції – попарного порівняння та перестановки даних. Алгоритми сортування масивів даних, які ґрунтуються на базовій операції попарного порівняння та перестановки даних, відрізняються між собою способами вибору пари даних і їх перестановки у масиві.

Базова операція попарного порівняння та перестановки даних також використовується для випадку коли дані, які необхідно відсортувати, розділені на m частин (боків) однакового розміру. Кількість блоків, на які розбивається масив даних, визначається архітектурою багатопроцесорної багатоядерної системи. Величина кожного боку, який необхідно відсортувати дорівнює $N=Q/m$ даних, де Q – розмір масиву даних. Сортування отриманих m блоків даних доцільно виконувати паралельно. Для подальшого паралельного сортування даних пропонується використовувати, як базову операцію сортування об'єднання двох упорядкованих масивів $\{a_{1i}\}_{i=1}^{N}$ та $\{a_{2i}\}_{i=1}^{N}$ у один упорядкований масив $\{b_{1i}\}_{i=1}^{2N}$ та його перестановки. Алгоритми сортування на основі базової операції об'єднання двох масивів в один та його перестановки використовуються для паралельного сортування масивів даних.

Результат декомпозиції алгоритму залежить від степеню деталізації алгоритму, чим більша степінь, тим простіше і легше здійснюється адаптація алгоритму до вимог конкретного застосування. Одним зі шляхів здійснення декомпозиції є використання методу функціональної декомпозиції. Використання методу функціональної декомпозиції дає можливість отримання просторово-часового відображення структури алгоритму сортування даних, як на основі базових операцій попарного порівняння та перестановки даних, так і на основі операції об'єднання двох упорядкованих масивів. Час і спосіб виконання таких операцій є одними із основних параметрів при визначені конвеєрного такту $T_к$ сортування даних. Результатом першого етапу розробки є граф-схеми паралельного алгоритму, визначення кількості блоків m та складності базової операції сортування даних.

На етапі проектування комунікацій визначається структура каналів для обміну даними між функціональними операторами $Ф_{ji}$, яка відповідає паралельно-

потоковому алгоритму сортування масиву даних, який розбитий на m блоків. Для чого виконується перехід від граф схеми паралельного алгоритму до його потокового графу, в якому виконується просторово-часове розміщення та закріплення функціональних операторів $Ф_i$ за ярусами. Структура зв'язків у паралельно-потоковому графі між функціональними операторами $Ф_{ji}$ сусідніх ярусів визначається кількістю каналів надходження даних.

Для варіанту програмної реалізації алгоритму сортування масиву даних на багатопроцесорній багатоядерній системі за результатами перших двох етапів розробки граф схеми можна визначити:

- кількість і розмір блоків, на які розбивається масив даних;
- складність базової операції алгоритму сортування даних, яка визначає макроконвеєрний такт сортування $T_{мк}$ даних;
- кількість каналів передачі даних між ярусами графа алгоритму сортування даних;
- структуру передачі даних між функціональними операторами ярусів графа алгоритму.

Результатом перших двох етапів розробки для варіанту апаратної реалізації потокового графу сортування є оцінка інтенсивності сортування масивів даних D_c. Вихідними даними для визначення інтенсивністю сортування $D_c = \frac{sn_s}{T_к}$ масивів даних є [3]:

- кількість каналів сортування даних s і їх розрядність n_s;
- складність функціональних операторів $Ф_{ji}$, яка визначає час їх реалізації та конвеєрний такт сортування $T_к$;
- швидкодія елементної бази, яка враховується при визначені конвеєрного такту сортування $T_к$.

При оцінювані узгодженості інтенсивності надходження даних P_d із обчислювальної здатності D_c вводиться коефіцієнт узгодженості, який визначається так:

$$L = \left\lceil \frac{P_d}{D_c} \right\rceil, \tag{2.24}$$

де ⌈ ⌉ - знак округлення до більшого цілого. $L=P_d/D_c$

Коефіцієнт узгодженості L може бути $L = 1$, $L > 1$ та $L < 1$. При $L = 1$, створений граф сортування даних є узгодженим і він забезпечує високу ефективність використання обладнання при його апаратній реалізації.

У випадку коли $L>1$, граф сортування є неузгодженим і необхідно збільшувати інтенсивність сортування D_c для його узгодження. Підвищення інтенсивності сортування D_c досягається шляхом збільшення кількісті каналів для сортування даних s та їх розрядность n_s або за рахунок зменшення складності функціональних операторів Φ_{ji}. В ситуації коли не вдається отримати необхідну інтенсивність сортування D_c за рахунок зміни перерахованих параметрів, тоді підвищення інтенсивності сортування D_c отримується за рахунок паралельного виконання L графів.

Третій етап проектування, зводиться до укрупнення операцій шляхом об’єднання каналів передачі даних і функціональних операторів Φ_{jk} як у межах одного ярусу, так і між різними ярусами. При програмній реалізації даний етап розробки паралельного алгоритму сортування даних орієнтований на конкретну реалізацію, яка враховує архітектуру багатопроцесорної багатоядерної системи. Для апаратної реалізації алгоритму сортування масиву даних у реальному часі враховується інтенсивність надходження даних.

У результаті такого об'єднання отримаємо потоковий граф сортування, який назиється конкретизованим потоковим графом. Укрупнення пов’язаного з етапом планування процесу сортування.

Четвертий етап планування процесу сортування для програмної реалізації на багатопроцесорній багатоядерній системі зводиться до визначення розміру блоків (масивів) даних, які будуть об’єднуватися, процесорів для виконання такої базової

операції. Основним критерієм ефективності виконання даного етапу для програмної реалізації на багатопроцесорній багатоядерній системі є мінімізація часу виконання програми сортування масиву із Q даних.

Для варіанту апаратної реалізації критерієм ефективності реалізації сортування масиву із Q даних є забезпечення реального часу при мінімальних апаратних затратах.

На цьому етапі визначається величина затримок, перестановки даних і виконується планування процесу сортування. Для відтворення операції сортування даних у конкретизований потоковий граф вводяться оператори затримки, управління та перестановки даних.

2.8. Висновки до розділу 2

1. Сформульовано вимоги до засобів інформаційних технологій паралельного сортування та пошуку даних, основною з яких є забезпечення реального часу.

2. Сформульовано вимоги до алгоритмів паралельного сортування та пошуку даних у реальному часі. Визначено, що алгоритми паралельного сортування та пошуку даних у реальному часі повинні бути: добре структурованими з детермінованим переміщенням даних; ґрунтуватися на однотипних операціях попарного порівняння та перестановки даних з регулярними та локальними зв’язками; орієнтованими на реалізацію на множині взаємозв’язаних процесорних ядер; використовувати конвеєризацію та просторовий паралелізм.

3. Вдосконалено та орієнтовано метод сортування масиву чисел вставкою на паралельне та паралельно-потокове сортування одновимірного масиву $\{a_i\}_1{}^m$. Вдосконалення досягнуто завдяки зміні розрядності каналів надходження та сортування даних.

4. Запропоновано зменшити час сортування методом вставки шляхом збільшення кількості чисел з невідсортованої частини, що вставляються до відсортованої частини так, щоб вона не втратила впорядкованості.

5. Вдосконалено та орієнтовано метод злиття на паралельне та паралельно-потокове сортування масивів даних.

6. Розроблені функціональні моделі алгоритмів паралельного сортування потоків даних у реальному часі. Вхідними даними таких моделей є інтенсивності надходження даних, розміри масиву, методи сортування та засоби реалізації. Функціональні моделі алгоритмів сортування завдяки знаходженню просторово-часових співвідношень забезпечують отримання паралельного конкретизованого графа для виконання сортування потоків даних у реальному часі. Процес розроблення функціональної моделі паралельного сортування масиву розбивається на такі чотири етапи: декомпозиція алгоритму сортування масиву; проектування комунікацій між функціональними операторами; укрупнення функціональних операторів; планування процесу сортування даних. Розроблені функціональні моделі паралельного сортування даних методом злиття та вставки, які внаслідок використання механізму управління паралелізмом забезпечують отримання конкретизованого графу алгоритму, орієнтованого на сортування у реальному часі на заданих засобах.

РОЗДІЛ 3. РОЗРОБЛЕННЯ МЕТОДІВ ТА ФУНКЦІОНАЛЬНИХ МОДЕЛЕЙ ПАРАЛЕЛЬНОГО-ВЕРТИКАЛЬНОГО СОРТУВАННЯ І ПОШУКУ ДАНИХ

У розділі розроблено та вдосконалено методи, алгоритми та функціональні моделі паралельно-вертикального сортування даних у реальному часі. Розроблено метод та функціональні моделі паралельно-вертикального пошуку у реальному часі максимальних і мінімальних чисел у масивах з інтенсивним надходженням даних.

3.1. Вдосконалення та орієнтація методу злиття на паралельно-вертикальне сортування даних

Як вже було сказано в розділі 2.5, орієнтація алгоритмів сортування на НВІС-реалізацію вимагає зменшення числа виводів інтерфейсу та розрядності з'єднань між функціональними операторами попарного порівняння та перестановки даних Φ_{ji}. Забезпечити ці вимоги можна використанням паралельно-вертикальних методів сортування даних, при яких надходження даних, сортування та видача результатів здійснюється розрядними зрізами [3].

Розглянемо паралельно-вертикальну реалізацію сортування даних злиттям з використанням базової операції з r=2. Дана базова операція ґрунтуються на операціях попарного порозрядного порівняння числами, які поступають в старшими розрядами вперед. В кожному i-у такті (i=1,…,n, n – розрядність чисел) виконується порозрядне попарне порівняння чисел (визначення максимального D_{max} та мінімального D_{min}).

Підвищити швидкодію сортування методом злиття можна шляхом використання базової операції, в якій $r \geq 3$. При використанні такої базової операції на початку сортування вхідний масив чисел $\{a_j\}_{j=1}^{m}$ розбивається на m/r груп чисел, в яких виконується об'єднання r упорядкованих масивів довжиною одиниця (базова операція) в один упорядкований масив довжиною r. У результаті виконання першого об'єднання формуються $m/2r$ груп упорядкованих масивів довжиною r. В кожній групі на основі базової операції $r \geq 3$ виконується об'єднання двох масивів розміром r

в один масив розміром $2r$ (макрооперація першого типу). Апаратно базова операція виконується ПЕ.

Наступні макрооперації виконуються аналогічно макрооперації першого типу. При цьому кількість об'єднань зменшиться на число p, яке рівне $p=\log 2\ r$.

Розглянемо паралельно-вертикальну реалізацію сортування даних злиттям на базі підрахунку (базова операція з $r=4$). Дана базова операція ґрунтуються на операціях попарного порозрядного порівняння кожного числа x_{ji} масиву $\{x_j\}_{j=1}^{4}$ з іншими числами, які поступають в пристрій старшими розрядами вперед. В кожному i-у такті ($i=1,\ldots,n$, n – розрядність чисел) виконується операція порозрядного попарного порівняння чисел відповідно до формули:

$$y_{j1i}y_{j2i}=\begin{cases} 00, \quad коли \quad (a_{1i}=a_{2i})\wedge(y_{1(i-1)}=y_{2(i-1)}=0) \\ 01, \quad коли \quad [(a_{1i}>a_{2i})\wedge(y_{1(i-1)}=y_{2(i-1)}=0)]\vee(y_{1(i-1)}=0, y_{2(i-1)}=1) \\ 10, \quad коли \quad [(a_{1i}<a_{2i})\wedge(y_{1(i-1)}=y_{2(i-1)}=0)]\vee(y_{1(i-1)}=1, y_{2(i-1)}=0) \end{cases} \qquad (3.1)$$

Структура вузла попарного порозрядного порівняння, який реалізує форм. 3.1, наведена на рис.3.1, де Км – комутатор; Тг – тригер; R – вхід скиду в нуль, TI – вхід тактових імпульсів.

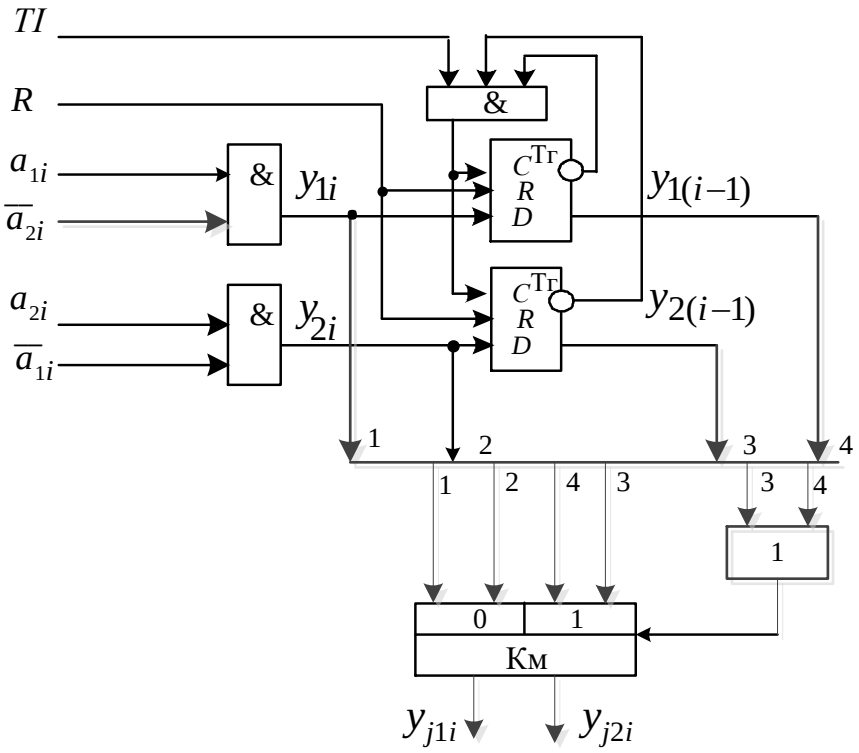


Рис.3.1. Структура вузла попарного порозрядного порівняння

На початку сортування тригери у вузлах попарного порівняння встановлюються в нуль. У кожному наступному такті роботи визначаються кількість чисел більших $Q_{jб}$ і менших $Q_{jм}$ числа $x_{j.}$ Таке визначення здійснюється за наступною формулою:

$$Q_{jб} = \sum_{k=1}^{m-1} y_{j1i}, \qquad Q_{jм} = \sum_{k=1}^{m-1} y_{j2i}\,, \tag{3.2}$$

де y_{j1i}, y_{j2i} – перший та другий розряди результату попарного порівняння числа x_j в i-у такті. Для кожного числа x_j масиву, визначається кількість однакових чисел та місце положення кожного числа у відсортованому масиві. Кількість однакових чисел у масиві визначається за формулою:

$$g_j = m-(Q_{jб}+Q_{jм}). \tag{3.3}$$

Структура ПЕ, який реалізує базову операцію паралельно-вертикального сортування для r=4, наведена на рис.3.2, де a_{1i}, a_{2i}, a_{3i}, a_{4i} – вхідні числа; b_{1i}, b_{2i}, b_{3i}, b_{4i} – відсортовані числа.

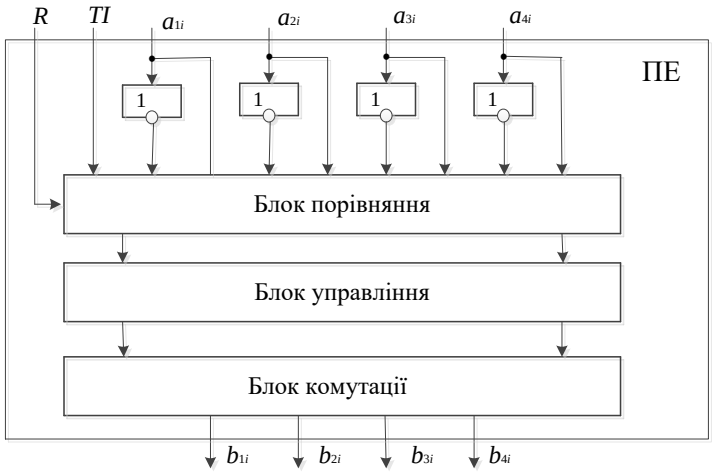


Рис.3.2. Структура ПЕ паралельно-вертикального сортування для r=4

У такому ПЕ на основі попарного порозрядного порівняння здійснюється паралельно-вертикальне сортування чотирьох чисел, яке є базовою операцією сортування методом злиття. Час виконання такої базової операції визначається так:

$$t_{БО} = t_{БП} + t_{БУ} + t_{БКм}\,, \tag{3.4}$$

де $t_{БП}$, $t_{БУ}$, $t_{БКм}$ – час відповідно порівняння, формування сигналів управління та комутації даних.

Макрооперація першого типу реалізується на трьох ПЕ об’єднаних у блок сортування $БС_1$ у відповідності з схемою рис.3.3.

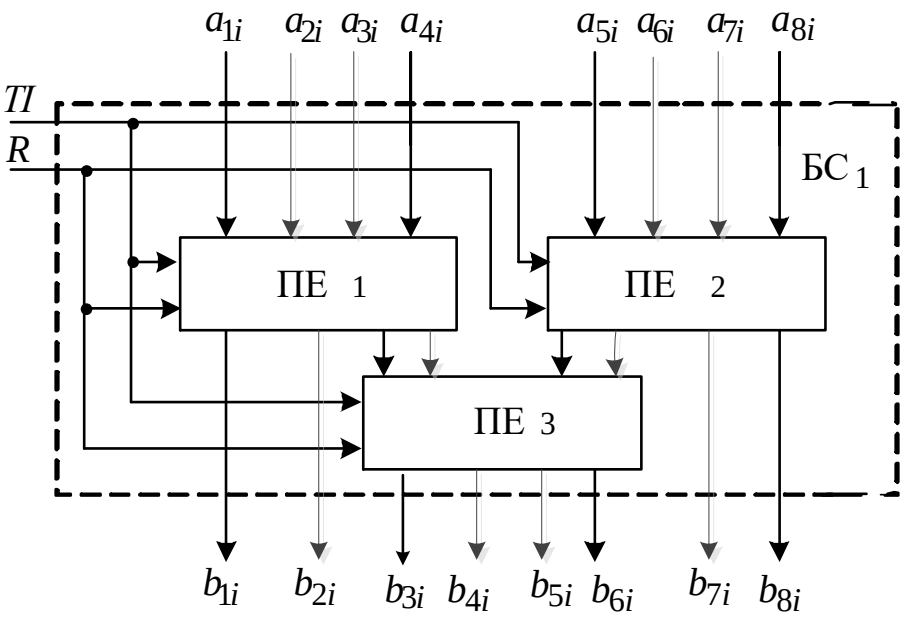


Рис.3.3. Схема блоку сортування першого типу

Кількість типів блоків сортування для сортування масиву з m чисел визначається за формулою:

$$K = log_2 m - 2. \quad (3.5)$$

Блок сортування s-го типу ($БС_s$), де s=1,…,K, реалізуються на основі трьох блоків типу $БС_{s-1}$. В кожному блоці $БС_s$ із двох упорядкованих підмасивів розміром 2^{s+1} методом злиття утворюється впорядкований масив розміром 2^{s+2}. Схема пристрою вертикального сортування методом злиття наведена на рис.3.4.

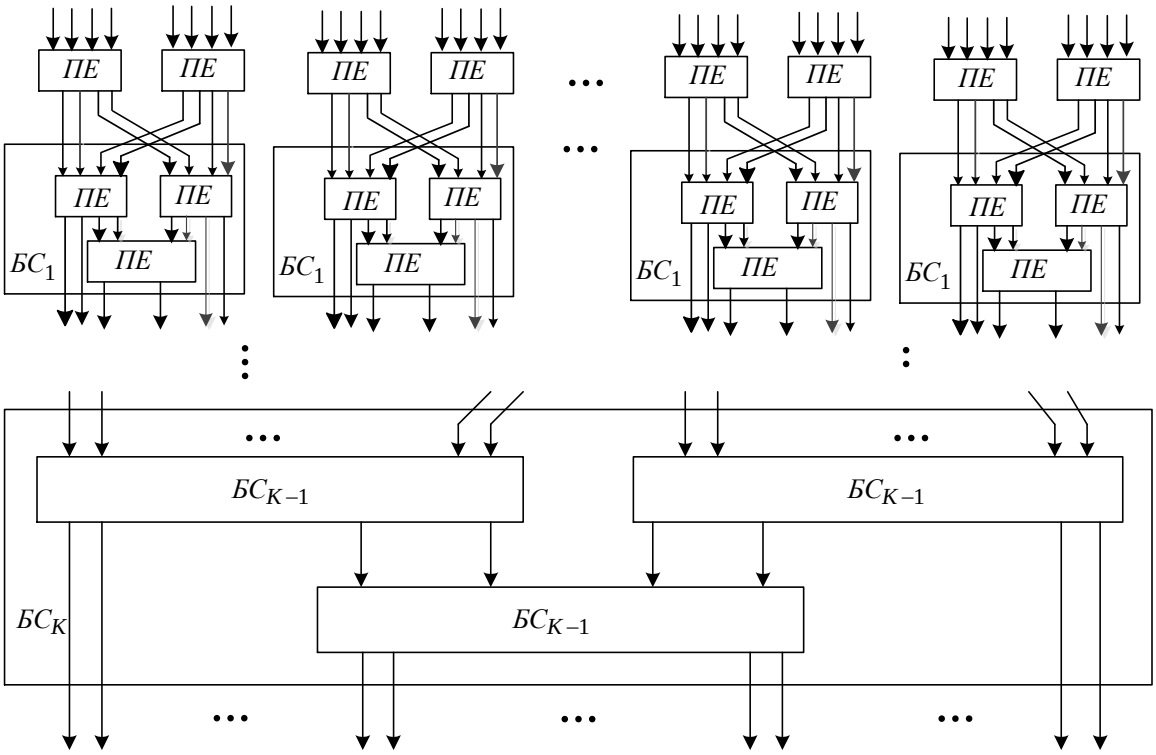


Рис.3.4. Схема пристрою паралельно-вертикального сортування методом злиття

Кількість ПЕ, які необхідні для сортування масиву з m чисел, визначається за формулою:

$$L = \frac{m}{r}\sum_{s=1}^{\kappa} 3^s 2^{K-s} . \tag{3.6}$$

Сортування даних у такому пристрої здійснюється із тактом:

$$T_{TI} = t_{БО}(1+\sum_{s=1}^{K} 2^s . \tag{3.7}$$

Час сортування в у такому пристрою визначається за формулою:

$$t_c = nT_{TI} , \tag{3.8}$$

де n – розрядність чисел сортування.

3.2. Вдосконалення та орієнтація методу вставки на паралельно-вертикальне сортування даних

При паралельно-вертикальному сортування масиву чисел $\{a_i\}_1^m$ методом вставки вхідні числа поступають паралельно розрядними зрізами починаючи з

старших розрядів. Граф алгоритму паралельно-вертикального сортування масиву чисел $\{a_i\}_1^m$ методом вставки відповідає графу паралельного сортування чисел методом вставки. Базова операція Φ_{ji} алгоритму паралельно-вертикального сортування масиву чисел $\{a_i\}_1^m$ методом вставки аналогічно як у паралельному алгоритмі зводиться до виконання двох елементарних операцій: попарного порівняння числа a_{i+1} з i-м числом b_{ji} відсортованої частини та перестановки чисел. Попарне порівняння чисел здійснюється так:

$$y_{ji} = \begin{cases} 0, при \quad a_{i+1} > b_{ji} \\ 1, при \quad a_{i+1} \le b_{ji} \end{cases} . \tag{3.9}$$

Результати попарного порівняння числа y_{ji} використовуються для перестановки чисел і формування виходу базової операції у відповідності з наступним виразом:

$$b_{ji} = \begin{cases} b_{(j-1)i-1,} \ коли \ y_{j(i-1)} = y_{ji} = 0 \\ a_{i+1}, коли \ y_{j(i-1)} = 1, y_{ji} = 0 \\ b_{(j-1)k}, коли \ y_{j(i-1)} = y_{ji} = 1 \end{cases}, \tag{3.10}$$

де $b_{(j\text{-}1)i\text{-}1}$, $b_{(j\text{-}1)i}$ – числа з виходів відповідно $\Phi_{(j\text{-}1)i\text{-}1}$, $\Phi_{(j\text{-}1)i}$; $y_{j(i\text{-}1)}$ – результат порівняння з виходів $\Phi_{j(i\text{-}1)}$. Особливістю базової операції паралельно-вертикального сортування масиву чисел $\{a_i\}_1^m$ методом вставки є те, що числа порівнюються і переставляться порозрядно.

Структура ПЕ, який реалізує базову операцію паралельно-вертикального сортування масиву чисел $\{a_i\}_1^m$ методом вставки наведена рис. 3.5, де Тг – тригер; *TI* – вхід тактових імпульсів; Км – комутатор; СП – схема порівняння; *R* – вхід скиду в нуль.

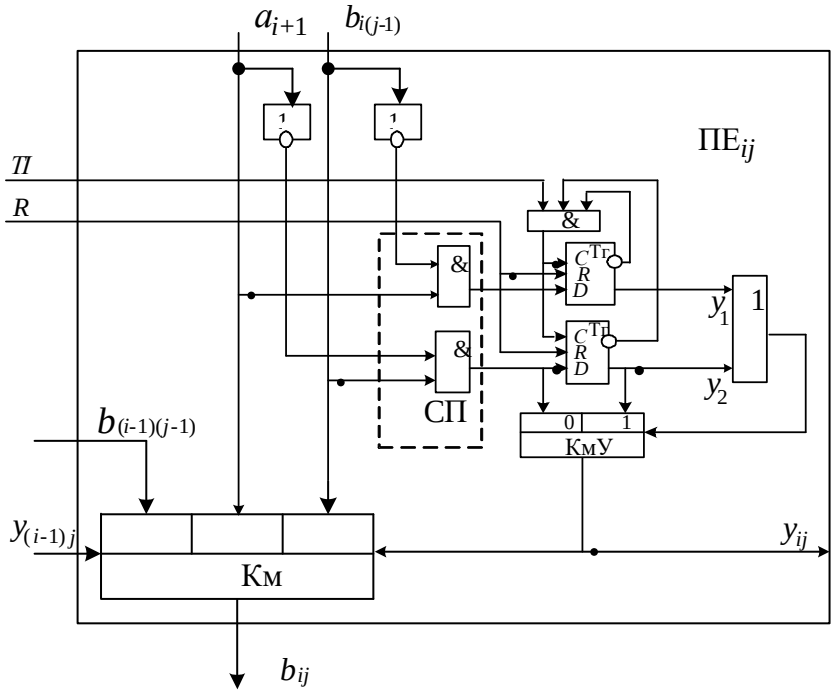


Рис.3.5. Структура ПЕ для паралельно-вертикального сортування чисел методом вставки

У ПЕ для порівняння необхідно виконати n тактів, де n – розрядність чисел. Формування сигналу порозрядного порівняння y_{ij} в ПЕ виконується СП, яка реалізована на основі двох елементах *І*. Результат виконання попереднього порівняння y_1 і y_2 записується в тригерах, запис в які блокується лог.0 з інверсних виходів цих тригерів. На початку сортування нового масиву чисел ці тригери встановлюються в нуль.

Для зміни інтенсивності сортування пропонується використовувати вертикально-групового надходження і порівняння чисел, розрядність яких може бути $1<p<n$.

3.3. Метод паралельно-вертикального сортування масивів чисел з використанням підрахунку одиниць у розрядному зрізі

Даний метод сортування передбачає паралельне надходження N чисел розрядними зрізами старшими розрядами уперед і паралельне формування розрядних зрізів відсортованих чисел. Сортування одновимірного масиву чисел $\{D_k\}_{k=1}^{N}$ здійснюється за n макротактів. При виконанні кожного i-го (i=1,…,n) макротакту

сортування масиву чисел одночасно виконується N базових операцій, які забезпечують формування i-х розрядів N відсортованих чисел. На першому макротакті сортування здійснюється формування старших розрядів всіх чисел. Даний макротакт зводиться до таких операцій:

1) підрахунку кількості одиниць у розрядному зрізі

$$S_1 = \sum_{k=1}^{N} D_{1k} \wedge y_{1k}, \tag{3.11}$$

де y_{1k} – значення k-го розряду1-го слова управління, яке дорівнює y_{1k}=1;

2) формування для $D_1^*,...,D_{s_1}^*$ виходів значення 1-го розрядного зрізу P_1за формулою:

$$P_1 = \bigvee_{k=1}^{N} D_{k1} \wedge y_{1k}, \tag{3.12}$$

де D_{k1} – значення 1-го розряду k-го числа масиву;

3) формування для $D_{S_1+1}^*,...,D_N^*$ виходів значення 1-го розрядного зрізу P_1за формулою:

$$P_1 = \bigvee_{k=1}^{N} D_{k1} \wedge \bar{y}_{1k}, \tag{3.13}$$

де $\bar{y}_{1k}$ – інверсне значення k-го розряду1-го слова управління;

4) визначення 1-х (старших) розрядів для виходів $D_{11}^*,..., D_{N1}^*$ за виразом:

$$D_{k1}^* = \begin{cases} 0, & коли \quad P_1 = 0 \\ 1, & коли \quad P_1 = 1 \end{cases} \tag{3.14}$$

5) формування 2-го слова управління для виходів $D_1^*,...,D_{s_1}^*$ за формулою:

$$y_{21},...,y_{2S_1} = \begin{cases} 0, & коли \quad P_1 = 1, \quad D_{k1} = 0 \\ 1, & коли \quad P_1 = 1, \quad D_{k1} = 1 \end{cases}. \tag{3.15}$$

6) формування 2-го слова управління для виходів $D_{S_1+1}^*,...,D_N^*$ за формулою:

$$y_{2(S_1+1)},...,y_{2N} = \begin{cases} 0, & коли \quad P_1 = 0, \quad D_{k1} = 1 \\ 1, & коли \quad P_1 = 0, \quad D_{k1} = 0 \end{cases}. \tag{3.16}$$

Наступні макротакти сортування для кожної групи розрядів $(D_1^*,...,D_{s_1}^*)$ виконуються незалежно та аналогічно до першого макротакту.

Сортування двовимірного масиву чисел $\{D_{kj}\}_{k=1;\,j=1}^{N;M}$ з використанням засобів паралельно-вертикального сортування одновимірних масивів чисел передбачає, що вхідні дані надходять паралельно N каналами. Таке сортування доцільно виконувати методом витіснення, базовою макрооперацією якого є паралельно-вертикальне сортування одновимірного масиву із $2N$ чисел. В кожному макротакті реалізації даного метода виконується сортування $2N$ чисел, з яких N є вхідними числами, а M-більшими числами з попереднього макротакту роботи. Відсортовані M більших чисел залишаються для виконання наступного макротакту роботи, а N менших записуються у пам'ять. Кількість макротактів, необхідних для сортування двовимірного масиву чисел $\{D_{kj}\}_{k=1;\,j=1}^{N;M}$, визначається за формулою:

$$p = \sum_{j=1}^{M} (M - j). \tag{3.17}$$

Одним із шляхів підвищення швидкодії сортування двовимірного масиву чисел є збільшення кількості базових макрооперацій, які виконуються паралельно.

3.4. Функціональна модель паралельно-вертикального сортування одновимірних масивів чисел з використанням підрахунку одиниць у розрядному зрізі

Функціональна модель паралельно-вертикального сортування одновимірних масивів чисел забезпечує просторово-часове відображення алгоритму паралельно-вертикального сортування одновимірного масиву $\{D_k\}_{k=1}^{N}$ чисел. Особливістю паралельно-вертикального сортування одновимірного масиву чисел є паралельне порозрядне надходження N вхідних чисел (розрядного зрізу) старшими розрядами вперед і паралельне порозрядне формування N відсортованих чисел. Потоковий граф алгоритму паралельно-вертикального сортування одновимірного масиву чисел наведений на рис.3.6, де Φ_1 і Φ_y – відповідно функціональний та управляючий оператори.

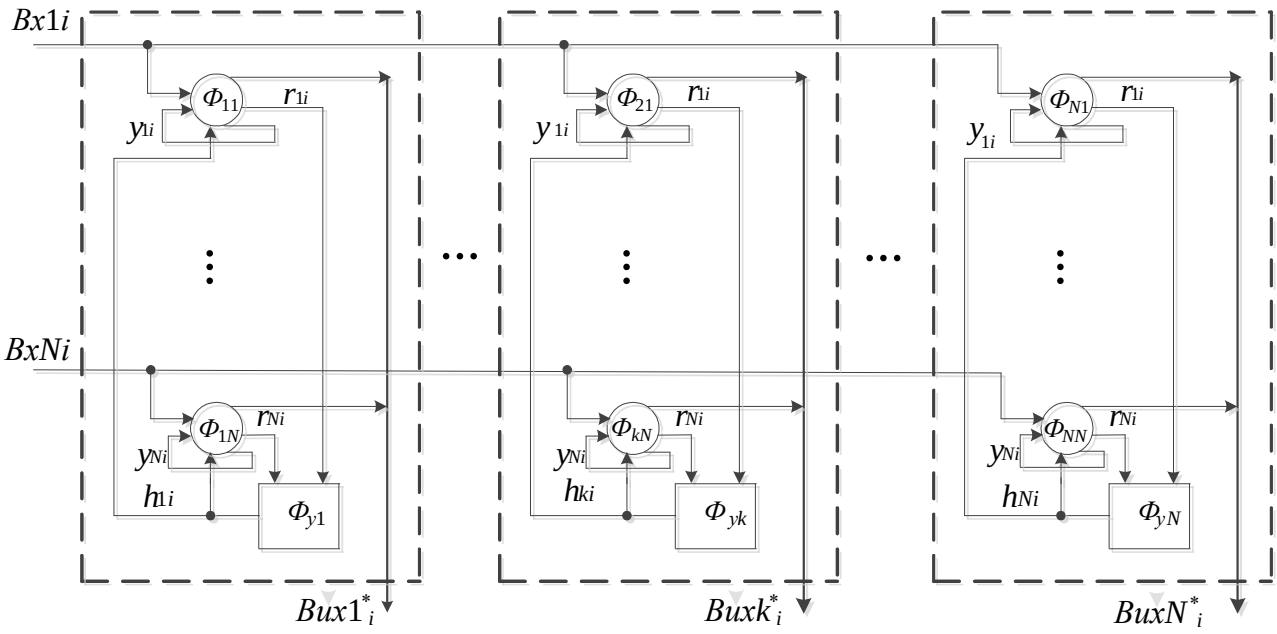


Рис.3.6 Функціональна модель паралельно-вертикального сортування одновимірних масивів чисел

Особливістю функціональної моделі алгоритму паралельно-вертикального сортування одновимірного масиву чисел є те, що з кожного k входу одночасно надходить на всі ПЕ. Кількість ПЕ визначається розмірністю масиву, який сортується. В кожному i-му такті роботи в ПЕ$_k$ формується i-й розряд відсортованого k-го числа. Найбільше відсортоване число формується на виході ПЕ$_1$, а найменше – на виході ПЕ$_N$. Кожний ПЕ реалізується на основі N функціональних операторів Φ_1 і одного функціонального оператора Φ_y. У ПЕ$_k$ функціональний оператор Φ_{1k} у кожному i-му такті роботи забезпечує виконання таких операцій:

1) обчислення значення i-го розряду для k-го входу за формулою:

$$P_{ki} = \overline{D_{ki} \wedge y_{ki}}\,, \tag{3.18}$$

2) формування сигналу для підрахунку кількості одиниць за формулою:

$$r_{ki} = (\overline{D}^*_{ki} \vee D_{ki}) \wedge y_i \tag{3.19}$$

3) визначення k розряду $(i+1)$-го слова управління за формулою:

$$y_{k(i+1)} = r_{ki} \oplus h_{ki}, \tag{3.20}$$

де h_{ki} – сигнал управління передачею формування (i+1)-го слова управління; $\overline{D}^*_{ki}$ – інверсне значення i-го розряду k-го відсортованого числа, яке обчислюється в ПЕ$_k$ за формулою:

$$\overline{D}^*_{ki} = \bigvee_{k=1}^{N} P_{ki} \ . \tag{3.21}$$

У ПЕ$_k$ функціональний оператор управління $Ф_{yk}$ забезпечує виконання таких операцій:

1) підрахунку кількості одиниць за формулою:

$$R_{ki} = \sum_{k=1}^{N} r_{ki} \tag{3.22}$$

2) визначення h_{ki} за формулою:

$$h_{ki} = \begin{cases} 0, & коли \quad \sum_k \geq Рг_{ki} \\ 1, & коли \quad \sum_k < Рг_{ki} \end{cases} \tag{3.23}$$

де значення регістра $Рг_{ki}$ в i-у такті.

На початку роботи у всіх ПЕ всі розряди слів управління установлюються в одиницю, а в регістр $Рг_k$ кожного k-го ПЕ$_k$ записується значення k.

3.5. Метод паралельно-вертикального пошуку максимальних і мінімальних чисел у одновимірному масиві даних

Метод паралельно-вертикального пошуку максимального D_{max} і мінімального D_{min} чисел у одновимірному $\{D_k\}_{k=1}^{N}$ та двовимірному $\{D_{kj}\}_{k=1;\,j=1}^{N;M}$ масивах передбачає паралельне надходження N чисел розрядними зрізами старшими розрядами уперед [2]. Визначення максимального D_{max} і мінімального $D_{\min}$ чисел в одновимірному масиві $\{D_k\}_{k=1}^{N}$ за даним методом ґрунтується на виконанні n однотипних базових макрооперацій, де n – розрядність чисел.

Базова макрооперація обчислення максимального числа $D_{\max}$ у одновимірному масиві $\{D_k\}_{k=1}^{N}$ реалізується за три етапи та використовує такі операції:

1) формування значення i-го розрядного (i=1,…,n) зрізу P_i за формулою:

$$P_i = \bigvee_{k=1}^{N} D_{ik} \wedge y_{ik}, \tag{3.24}$$

де D_{ik} – значення i-го розряду k-го числа масиву, y_{ik} – значення k-го розряду i-го слова управління, значення першого слова управління дорівнює $y_{11}=y_{12}=\ldots=y_{1k}=\ldots=y_{1N}=1$;

2) визначення i-го розряду максимального числа $D_{\max i}$ за виразом:

$$D_{\max i} = \begin{cases} 0, & коли \quad P_i = 0 \\ 1, & коли \quad P_i = 1 \end{cases}; \tag{3.25}$$

3) формування k розрядів $(i+1)$-го слова управління за формулою:

$$y_{(i+1)k} = \begin{cases} 0, & коли \quad P_i = 1, \quad D_{ki} \neq y_{ik} \\ 1, & коли \quad P_i = D_{ki} = y_{ki} = 1 \\ y_{ki}, & коли \quad P_i = 0 \end{cases}. \tag{3.26}$$

Обчислення мінімального числа $D_{\min}$ у одновимірному масиві $\{D_k\}_{k=1}^{N}$ ґрунтується на базовій макрооперації, яка реалізується за такі три етапи:

1) формування значення i-го розрядного зрізу P_i, яке виконується за формулою:

$$P_i = \bigvee_{k=1}^{N} \overline{D}_{ik} \wedge y_{ik}, \tag{3.27}$$

де $\overline{D}_{ik}$ – інверсне значення i-го розряду k-го числа масиву; y_{ik} – значення k-го розряду i-го слова управління, значення 1-го слова управління дорівнює $y_{11}=y_{12}=\ldots=y_{1k}=\ldots=y_{1N}=1$;

2) визначення i-го розряду мінімального числа $D_{\min i}$ за виразом:

$$D_{\min i} = \begin{cases} 0, & коли \quad P_i = 1 \\ 1, & коли \quad P_i = 0 \end{cases}; \tag{3.28}$$

3) формування k розрядів $(i+1)$-го слова управління, здійснюється за допомогою наступного виразу:

$$y_{(i+1)k} = \begin{cases} 0, & коли \quad P_i = 1, \quad \overline{D}_{ki} \neq y_{ik} \\ 1, & коли \quad P_i = \overline{D}_{ki} = y_{ki} = 1 \\ y_{ki}, & коли \quad P_i = 0 \end{cases}. \tag{3.29}$$

Особливістю розглянутого методу паралельно-вертикального обрахунку максимальних і мінімальних чисел у числових масивах є [2]:

- використання однієї макрооперації;
- можливість використання розпаралелення та конвеєризації обчислень;
- можливість одночасного опрацювання N розрядних зрізів;
- час обчислення в основному визначається не кількістю N чисел, а їх розрядністю.

3.6. Функціональна модель паралельно-вертикального пошуку максимальних і мінімальних чисел у одновимірному масиві

Для управління просторово-часовим розпаралленням алгоритмів паралельно-вертикального обчислення максимальних і мінімальних чисел з метою забезпечення реального часу при мінімальних затратах обладнання розроблена функціональна модель алгоритму $F=(\Phi,\Gamma)$, де $\Phi=\{\Phi_1,\Phi_2,\ldots,\Phi_n\}$ – набір функціональних операторів, які відповідають операціям обчислення максимального (3.24) – (3.26) (мінімального (3.27) – (3.29)) числа, Γ – зв'язки між функціональними операторами [5]. Графічно функціональна модель алгоритму паралельно-вертикального обчислення максимального числа відображається у вигляді вершин, що відповідають операціям обчислення (3.24) – (3.26) та дуг, які відображають зв'язки між ними. Виявити паралелізм і знайти оптимальні просторово-часові рішення для реалізації алгоритму паралельно-вертикального обчислення максимального (мінімального) числа забезпечується поданням його в ярусно-паралельній формі у вигляді потокового графу. Потоковий граф алгоритму паралельно-вертикального обчислення максимального числа наведений на рис.3.7, де y_{ik} – значення k-го розряду i-го слова управління; D_{ki} – i-й розряд k-го числа; Φ_1, Φ_2, Φ_3 – функціональні оператори, які виконують операції відповідно $P_{ik} = D_{ik} \wedge y_{ik}$, $y_{(i+1)k} = (P_i \wedge D_{ki} \wedge y_{ik}) \vee (\overline{P_i} \vee y_{sk})$, $P_i = \bigvee_{k=1}^{N} P_{ik}$.

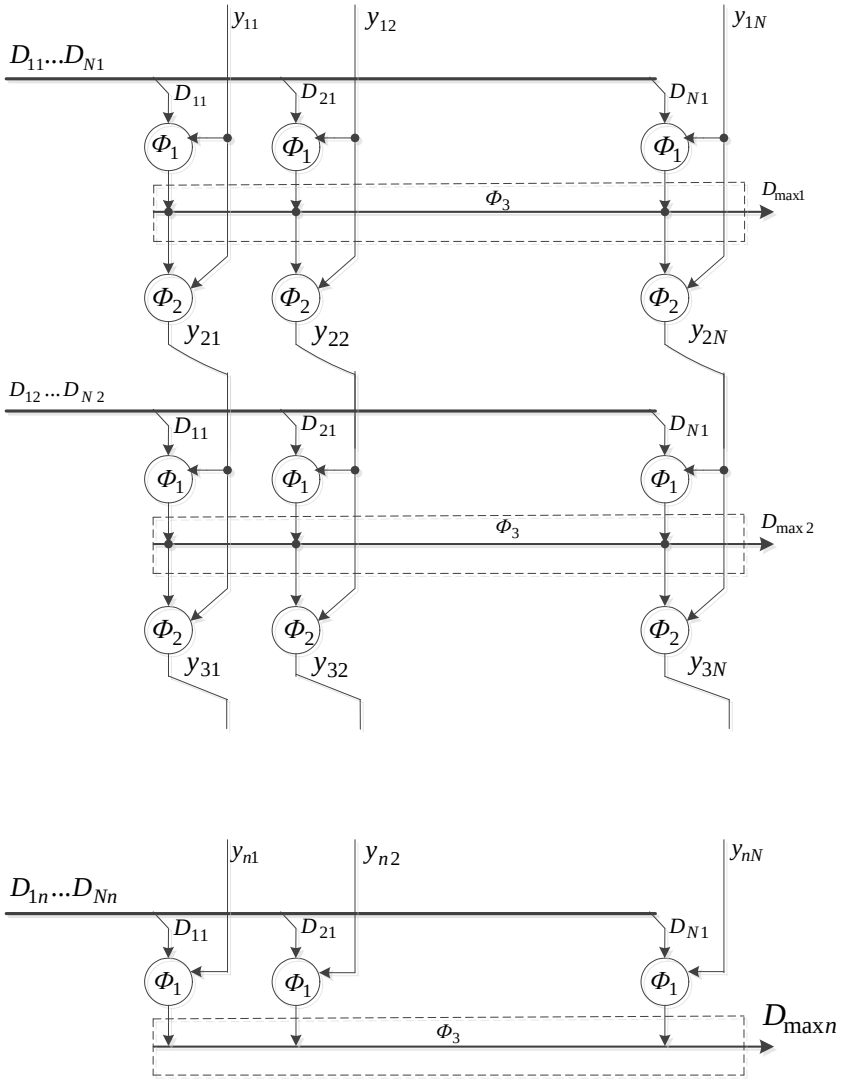


Рис.3.7. Функціональна модель паралельно-вертикального пошуку максимального числа

Розроблена функціональна модель забезпечує просторово-часове відображення процесу паралельно-вертикального обчислення максимального числа в одновимірному масиві для випадку коли всі N числа надходять одночасно. Аналогічну структуру має потоковий граф алгоритму паралельно-вертикального обчислення мінімального числа, який відрізняться тільки операціями, що реалізуються функціональним оператором $\Phi_1 = \overline{D}_{ik} \wedge y_{ik}$.

У результаті апаратного відображення функціональної моделі (рис.3.7) отримаємо матричну структуру, яка забезпечить інтенсивність обчислення максимального (мінімального) числа, що визначається за формулою:

$$D_{M\max} = \frac{Nn}{T_к}, \tag{3.30}$$

де $T_к$ – конвеєрний такт, який визначається складністю функціональних операторів $Ф_1$, $Ф_2$ і $Ф_3$.

Зменшити інтенсивність обчислення максимального (мінімального) числа та адаптувати граф алгоритму (рис.3.7) до ітераційної структури можна шляхом лінійної проекції його на горизонтальну вісь X. Такий граф алгоритму обчислення максимального (мінімального) числа наведений на рис.3.8, де $Ф_{МУ}$ – макрооператор управління, який забезпечує збереження інформації про структуру потокового графа обчислення.

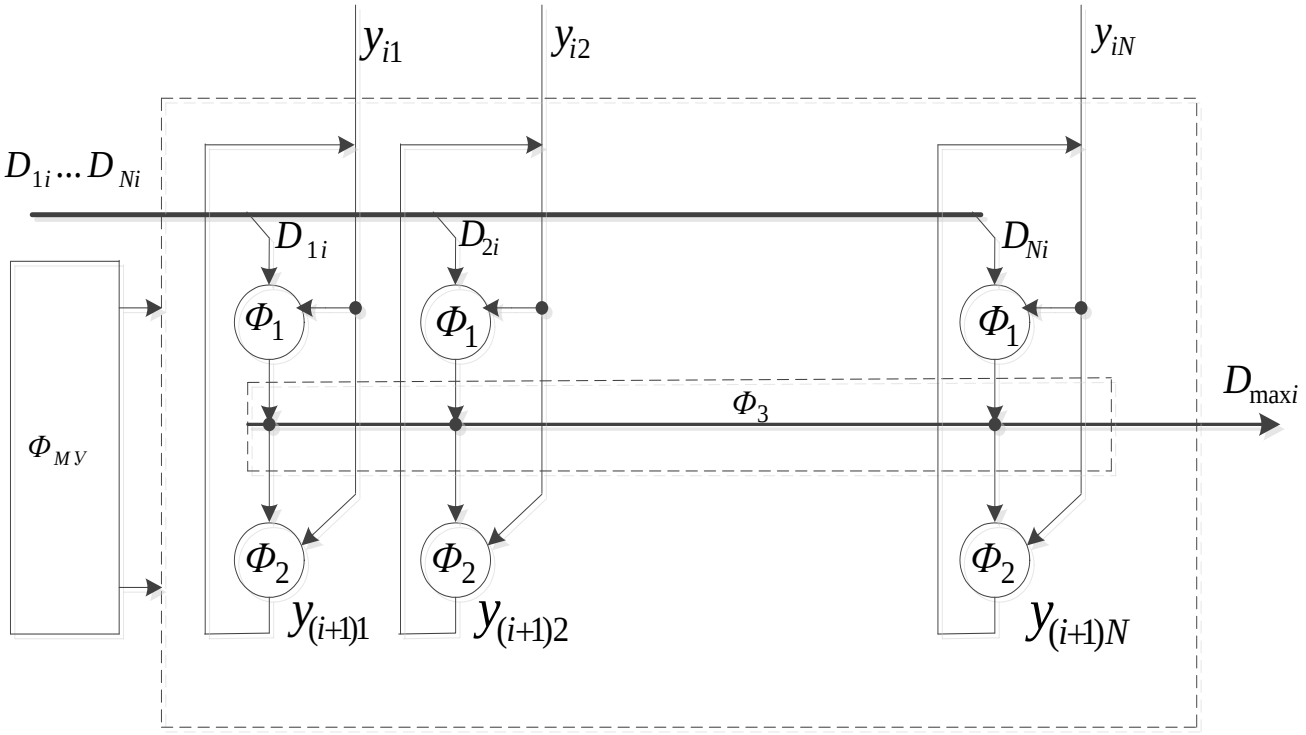


Рис.3.8. Лінійна проекція графової моделі обчислення максимального (мінімального) числа на горизонтальну вісь X

Графа модель (рис.3.8) передбачає надходження даних розрядними зрізами. Інтенсивність обчислення максимального (мінімального) числа, яку можна досягнути при реалізації такого графа алгоритму, визначається за формулою:

$$D_{1\max} = \frac{N}{T_к}. \qquad (3.31)$$

Зменшити час обчислення максимального (мінімального) числа можна шляхом збільшення розрядності каналів надходження чисел $n_к$. Збільшення $n_к \geq 2$ веде відповідно до ускладнення графа алгоритму обчислення максимального (мінімального) числа у порівнянні з графом (рис.3.8) та збільшення кількості розрядів максимального (мінімального) числа, які одночасно обчислюються в одному такті.

3.7. Вдосконалення методу пошуку максимальних (мінімальних) чисел у двовимірних масивах

Обчислення максимального D_{max} та мінімального D_{min} чисел в двовимірному масиві $\{D_{kj}\}_{k=1;\,j=1}^{N;M}$ виконується на основі базових макрооперацій, які виконуються за відповідними формулами (3.24)-(3.26) та (3.27)-(3.29). Різниця визначення максимального D_{max} (мінімального D_{min}) чисел в двовимірному масиві $\{D_{kj}\}_{k=1;\,j=1}^{N;M}$ полягає у тому, що після виконання кожних n базових операцій над одновимірним масивом із (N+1) до обчислення долучається максимальне (мінімальне) число нового j-го одновимірного масиву та здійснюється запис одиниць у всі розряди регістра правління. Обчислення максимального (мінімального) числа в двовимірному масиві $\{D_{kj}\}_{k=1;\,j=1}^{N;M}$ вимагає виконання $M \times n$ базових операцій [2].

Лінійна проекція на горизонтальну вісь X графу алгоритму обчислення максимального D_{max} (мінімального D_{min}) чисел у двовимірному масиві $\{D_{kj}\}_{k=1;\,j=1}^{N;M}$), наведена на рис.3.9, де $Ф_{MУ}$ – макрооператор управління, який забезпечує збереження інформації про структуру потокового графа обчислення максимального D_{max} (мінімального D_{min}) чисел у двовимірному масиві; $Ф_1$, $Ф_2$, $Ф_3$ – функціональні оператори, які виконують операції відповідно $P_{ik} = D_{ik} \wedge y_{ik}$,

$y_{(i+1)k} = (P_i \wedge D_{ki} \wedge y_{ik}) \vee (\overline{P_i} \vee y_{sk})$, $P_i = \bigvee_{k=1}^{N} P_{ik}$.

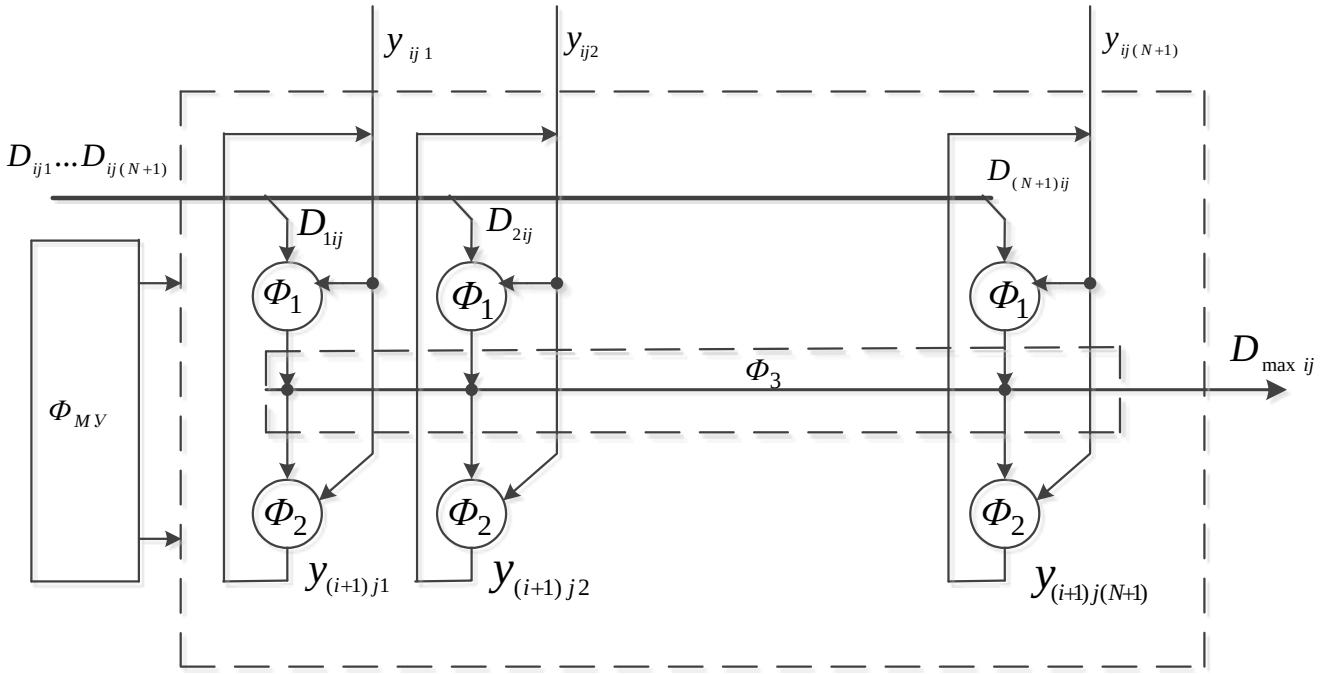


Рис.3.9. Лінійна проекція на горизонтальну вісь *X* графової моделі обчислення максимального (мінімального) числа у двовимірному масиві

Даний граф орієнтований на ітераційне обчислення максимального (мінімального) числа у двовимірному масиві для випадку паралельного надходження *N* чисел розрядними зрізами старшими розрядами уперед.

Збільшити швидкодію можна шляхом одночасного опрацювання двох і більше стрічок з двовимірного масиву. Граф такого алгоритму, наведений на рис.3.10, де *p* – кількість стрічок двовимірного масиву, які опрацьовуються одночасно.

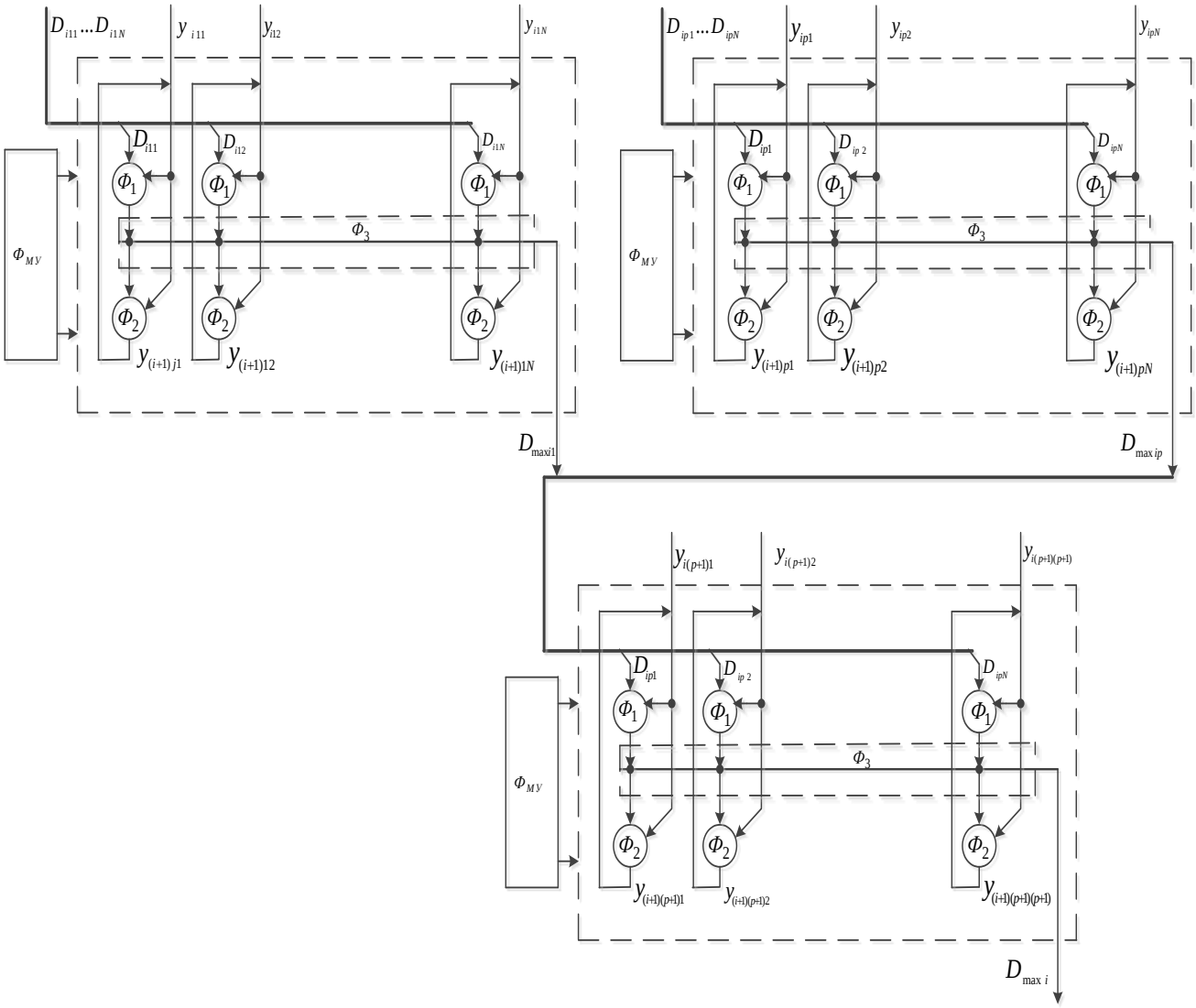


Рис.3.10. Графова модель паралельно-групового обчислення максимального (мінімального) числа у двовимірному масиві

Обчислення максимального (мінімального) числа за таким графом здійснюється за $(n+1)k$ тактів, де n – розрядність чисел, $k=M/p$.

Паралельно-паралельне обчислення максимального (мінімального) числа у двовимірному масиві вимагає одночасного опрацювання всіх M стрічок. Граф алгоритму такого опрацювання складається двох сходинок. Перша обчислює M максимальних (мінімальних) чисел двовимірного масиву, друга – обчислює максимальне (мінімальне) число з M чисел, які надходять з першої сходинки. Час обчислення максимального (мінімального) числа у двовимірному масиві за таким графом алгоритмом рівний $(n+1)$ тактів.

3.8. Паралельні структури пристроїв обчислення максимальних і мінімальних значень із масиву чисел

Аналіз методів і алгоритмів пошуку максимальних і мінімальних чисел із масиву даних показав, що найефективнішими є алгоритмами для реалізації їх у вигляді НВІС, які базуються на методі порозрядного порівняння [2, 5]. Пошук максимального A_{max} і мінімального A_{min} чисел із групи чисел A_1, A_2,…, A_j,…, A_m за допомогою даного методу виконується послідовним порівнянням розрядів всіх чисел починаючи зі старшого розряду. При кожному такому порівнянні отримується i-і розряди максимального і мінімального чисел, пошук яких здійснюється за наступними формулами [2]:

$$\overline{A_{i\max}} = \bigwedge_{j=1}^{m} \overline{a_{ji} \wedge y_{ij}}, \quad y_{1j} = 1; \tag{3.32}$$

$$A_{i\min} = \bigwedge_{j=1}^{m} \overline{a_{ji} \wedge z_{ij}}, \quad z_{1j} = 1. \tag{3.33}$$

де y_{ij}, z_{ij} - i-і розряди j-х слів управління; a_{ji}- i-й розряд j-го числа; m – кількість чисел у групі.

Формування $(i+1)$-х розрядів j-х слів управління виконується за формулами:

$$y_{(i+1)j} = (\overline{A}_{i\max} \vee x_{ji}) \wedge y_{ij}, \tag{3.34}$$

$$z_{(i+1)j} = (A_{i\min} \vee x_{ji}) \wedge z_{ij}. \tag{3.35}$$

Синтез НВІС-структур для паралельного пошуку максимальних і мінімальних чисел зводиться до наступних етапів [2]:

- виділення базової операції;
- просторово-часове відображення алгоритму;
- розробка схеми ПЕ, для реалізації базової операції алгоритму;
- синтез НВІС-структур на базі ПЕ;
- організація інтерфейсу НВІС-структури.

При аналізі алгоритмів паралельного обчислення максимальних і мінімальних значень із групи чисел на основі методу порозрядного порівняння дозволяється виділити базову операцію для НВІС-реалізації [2]. Базова операція складається з

формування розрядів слів управління використовуючи формулами (3.34) і (3.35) та виконання наступних логічних обчислень:

$$\overline{A_{ji\max}} = \overline{a_{ji} \wedge y_{ij}}\ , \tag{3.36}$$

$$A_{ji\min} = \overline{\overline{a_{ji}} \wedge z_{ij}}\ . \tag{3.37}$$

Виділена базова операція створюється у вигляді ПЕ, схеми яких наведені на рис.3.11, де а – ПЕ для однотактного пристрою; б – ПЕ для конвеєрного пристрою; в – ПЕ для пристрою з вертикальним опрацюванням вхідних чисел.

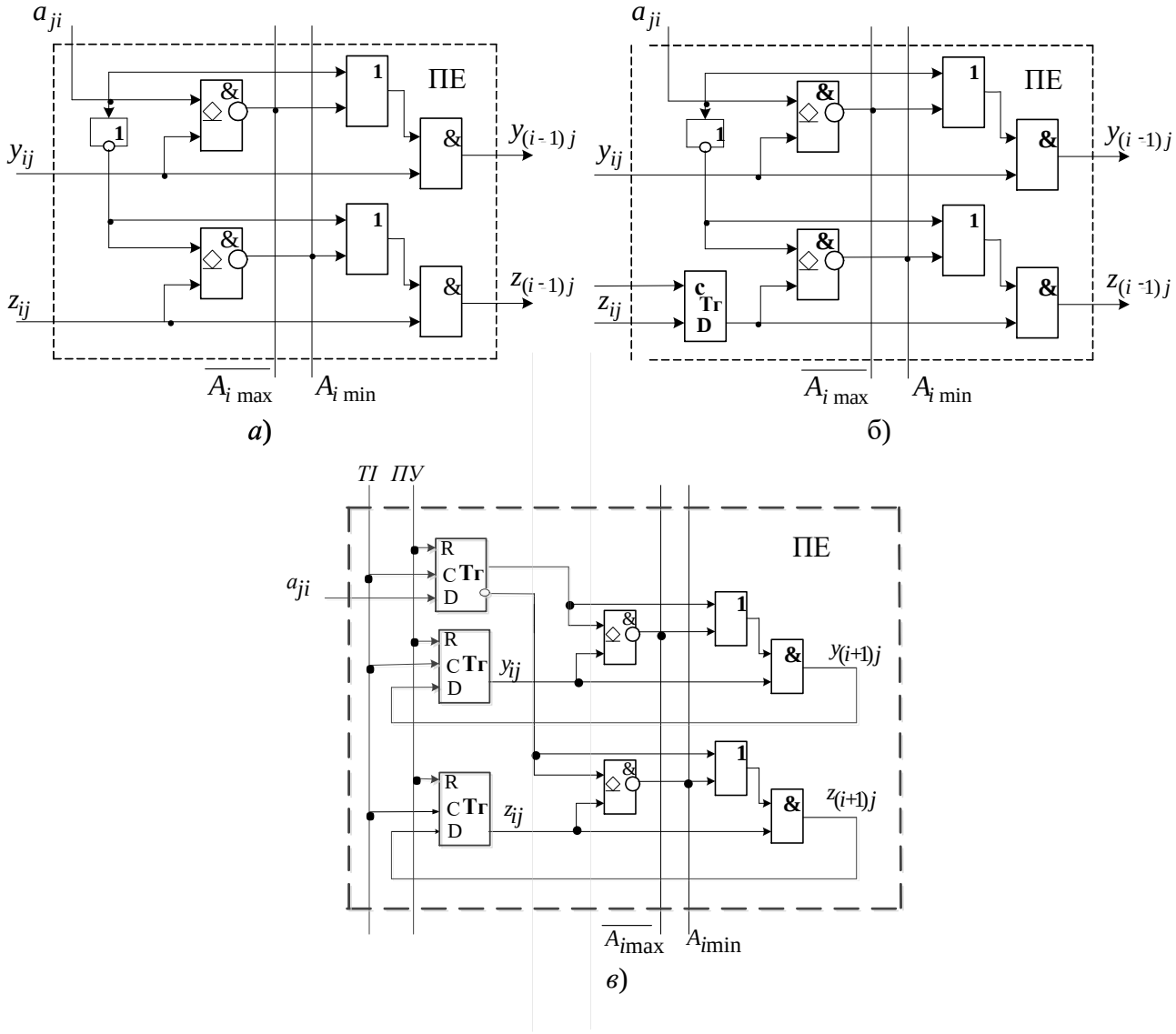


Рис. 3.11. Схеми ПЕ: *а* - однотактного пристрою; *б* – конвеєрного пристрою; *в* – пристрою з вертикальним опрацюванням

Слід визначити наступні особливості розроблених ПЕ, а саме використання спільних шин для результатів виконання обчислення, підключення до яких виконується за допомогою логічних елементів 2*I-HE* з відкритим колектором. Збільшення розміру масиву чисел, в якому обчислюються максимальні і мінімальні значення забезпечується за рахунок вертикального підключення до спільних шин результатів. Висока швидкодія забезпечується також за рахунок використання спільних шин результатів, також швидкодія не залежить від кількості чисел у масиві, а залежить тільки від часу затримки на процесорному елементі.

Вартість НВІС-пристроїв для обчислення максимальних і мінімальних значень із групи чисел залежить в основному від площі кристала, яка в свою чергу визначається як кількість транзисторів, так і кількістю зовнішніх виводів, число яких обмежене рівнем технології та розміром кристалу. НВІС-реалізація для структур сортування вимагає зменшення числа виводів інтерфейсу та кількості з'єднань між ПЕ. Забезпечення цих вимог можна за рахунок використання паралельно-вертикального алгоритму обчислення максимальних і мінімальних значень із групи чисел, при якому надходження чисел і видача результатів здійснюється розрядними зрізами [2,5].

Структура паралельно-вертикального НВІС-пристрою обчислення максимальних і мінімальних значень із групи чисел наведена на рис.3.12, де ТІ – тактовий вхід; ПУ – вхід початкової установки тригерів; $a_1,\ldots,a_m$ – однорозрядні інформаційні входи; де m – кількість чисел, що порівнюються; $ПЕ_1,\ldots$, $ПЕ_m$ – ПЕ пристрою з вертикальним опрацюванням вхідних чисел; $Тг_1$ і $Тг_2$ – D-тригери; $\overline{A_{max}}$ і A_{min} – порозрядний вихід відповідно інверсного максимального та мінімального чисел [2,5].

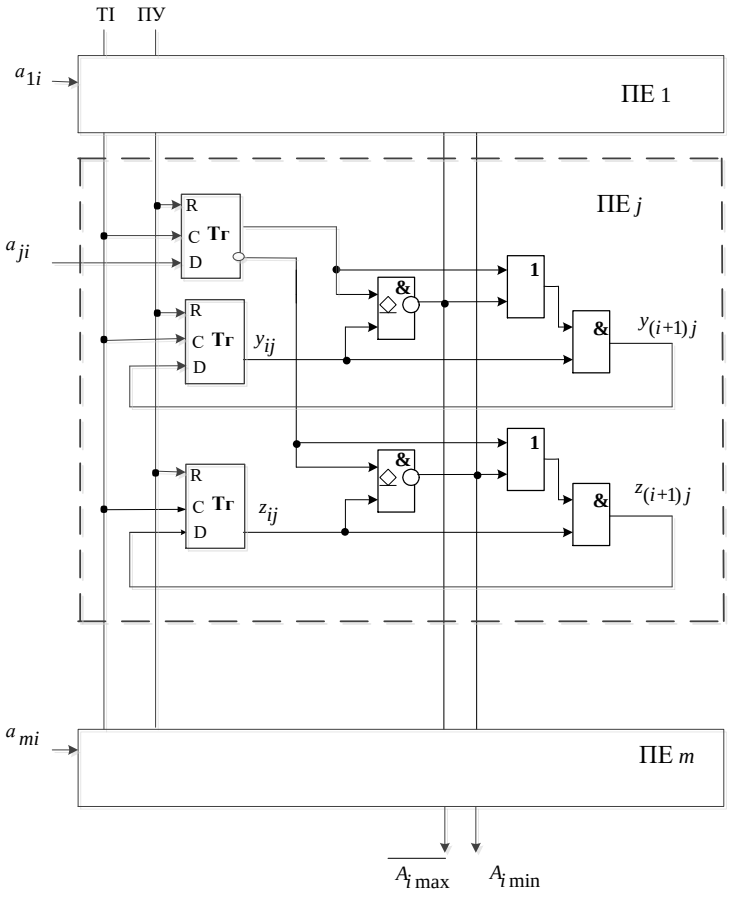


Рис.3.12. Схема паралельно-вертикального НВІС-пристрою обчислення максимальних і мінімальних значень

Виконання паралельно-вертикального НВІС-пристрою пошуку максимальних і мінімальних значень зводиться до наступних кроків. Перед початком роботи всі тригери пристрою, встановлюються у лог.0 за допомогою імпульсу початкової установки, який надходить із входу ПУ. При одночасному обчисленні максимального та мінімального чисел в одновимірному масиві із m чисел дані на інформаційні входи надходить порозрядно старшими розрядами вперед. Так у i-у такті роботи пристрою в тригери даних з інформаційних входів записуються i-і розряди чисел, в тригери управління – i-і розряди y_{ij}, z_{ij} слів управління. За n тактів, де n – розрядність чисел, на виході $\overline{A_{\max}}$ отримаємо інверсне значення максимального числа, а на виході A_{min} – мінімальне значення числа із одновимірного масиву з m чисел [2,5].

Час який необхідний для обчислення максимального та мінімального чисел із масиву з m визначається за [2]:

$$t_m = (t_{Tг} + 3t_I)n, \tag{3.38}$$

де $t_{Tг}$ – час запису інформації у тригер; t_I – час затримки інформації при проходженні через логічні елементи типу *АБО*, *І*, *І-НЕ.*

Затрати обладнання на реалізацію даного пристрою рівні:

$$W_m = (3W_{Tг} + 6W_I)m, \tag{3.39}$$

де $W_{Tг}$ – затрати обладнання на реалізацію *D*-тригера, W_I – затрати обладнання на реалізацію логічних елементів типу *АБО*, *І*, *І-НЕ*.

Для роботи з інтенсивнішими потоками даних вимагається збільшення обчислювальної здатності, яка досягається за рахунок збільшення розрядності каналів надходження даних, яка може бути два і більше. При розрядності каналів надходження даних яка дорівнює n, тобто коли одночасно опрацьовуються всі розряди чисел досягається максимальна швидкодія досягається коли [2].

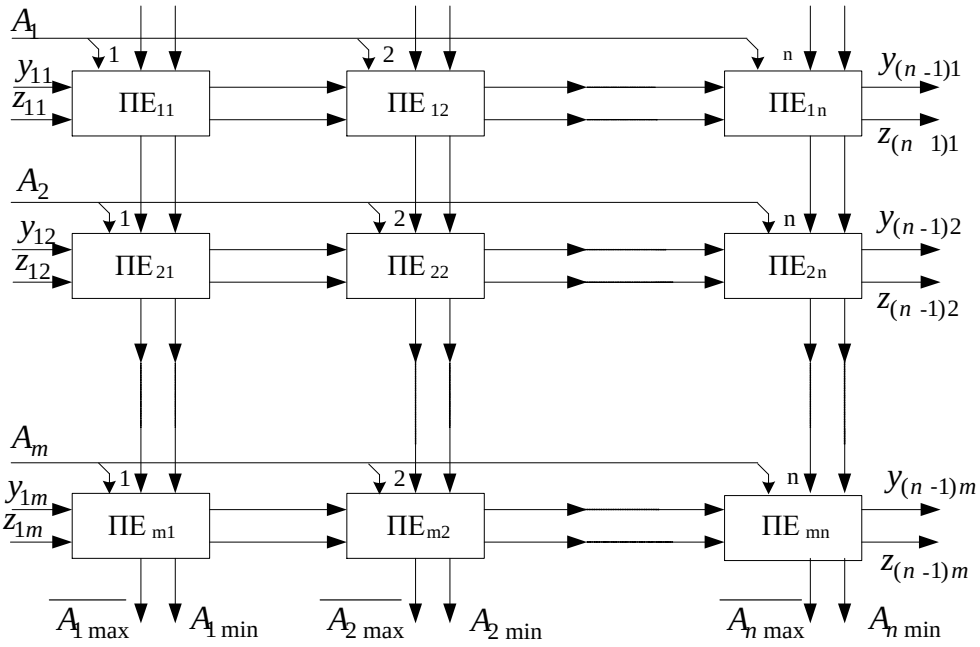


Рис.3.13 Матрична однотактна НВІС-структура пристрою обчислення максимальних і мінімальних значень із групи m чисел

Використовуючи однотактного і конвеєрного ПЕ створено відповідні матричні однотактні (рис.3.13) та конвеєрні (рис.3.14) паралельні НВІС-структури пристрою обчислення максимальних і мінімальних значень із групи чисел. Кожна із структур складається із матриці (m×n) ПЕ, в якій $ПЕ_{1i}$-$ПЕ_{mi}$ i-го стовпчика обчислюють

відповідно до формул (3.32) і (3.33) значення *i*-х розрядів максимального $\overline{A_{i\max}}$ і мінімального $A_{i\min}$ чисел та формують у відповідності з формулами (3.34) і (3.35) значення (i+1)- х слів управління $y_{(i+1)}$ і $z_{(i+1)}$.

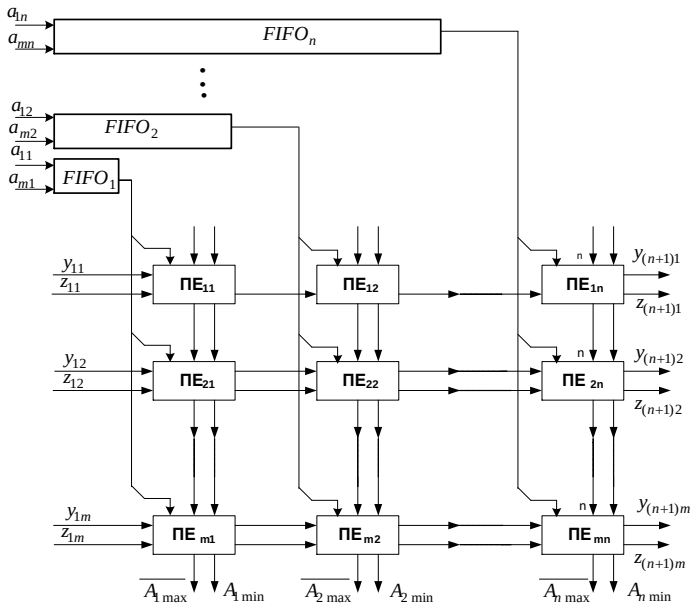


Рис.3.14. Матрична конвеєрна НВІС-структура пристрою обчислення максимальних і мінімальних значень із групи *m* чисел

Час пошуку максимальних і мінімальних значень у однотактному пристрої визначається за наступною формулою [2]:

$$T_o = 4nt_i, \tag{3.40}$$

де t_i - час спрацювання логічного елемента “І”; *n* – розрядність чисел.

Апаратні витрати на реалізацію однотактного пристрою дорівнюють:

$$W_o = 7NnW_i, \tag{3.41}$$

де W_i – апаратні витрати на логічні елементи типу І, АБО, І-НЕ.

Особливістю конвеєрної НВІС-структури (рис.3.14) є введення у ПЕ тригерів та використання *n* блоків пам’яті типу FIFO. Кожний *i*-ий блок $FIFO_i$ забезпечує затримку інформації на *i* тактів. Тривалість виконання конвеєрного такту в даному пристрої визначається за формулою:

$$T_к = t_{Tг} + 4nt_i, \tag{3.42}$$

де $t_{Tг}$ – час спрацювання тригера.

Апаратні витрати на створення конвеєрного пристрою обчислення максимальних і мінімальних значень із групи m чисел дорівнюють:

$$W_к = W_{FIFO} + mn(W_{Tг} + 7W_i), \quad (3.43)$$

де W_{FIFO} і $W_{Tг}$ – апаратні затрати відповідно на пам’ять типу FIFO і тригер.

3.9. Висновки до розділу 3

1. Вдосконалено та орієнтовано метод злиття на паралельно-вертикальне сортування даних у реальному часі. Вдосконалення досягнуто завдяки використанню базової операції, яка ґрунтується на попарному порозрядному порівнянні та перестановці трьох і більше чисел $r \geq 3$, які поступають старшими розрядами вперед. Використання такої базової операції забезпечує зменшення часу сортування.

2. Вдосконалено та орієнтовано метод вставки на паралельно-вертикальне сортування даних у реальному часі. При виконанні такого сортування вхідні дані надходять паралельно розрядними зрізами, починаючи зі старших розрядів. Для зміни інтенсивності сортування пропонується використовувати базову операцію з вертикально-груповим порівнянням та перестановкою чисел.

3. Розроблено метод паралельно-вертикального сортування даних, який завдяки підрахунку одиниць у i-му вхідному розрядному зрізі та паралельному формуванню i-го розрядного зрізу відсортованого масиву чисел забезпечує зменшення часу сортування

4. Розроблено функціональну модель паралельно-вертикального сортування одновимірного масиву чисел.

5. Розроблено метод паралельно-вертикального пошуку максимальних і мінімальних чисел у масивах, який внаслідок паралельного опрацювання i-го розрядного зрізу масиву чисел і паралельного формування слів управління забезпечує зменшення часу пошуку, який визначається в основному розрядністю чисел.

6. Розроблено функціональну моделі паралельно-вертикального пошуку максимальних (мінімальних) чисел.

РОЗДІЛ 4. РОЗРОБЛЕННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЗАСОБІВ ПАРАЛЕЛЬНОГО СОРТУВАННЯ І ПОШУКУ ДАНИХ

У даному розділі розроблено інформаційну технологію паралельного сортування даних та інформаційну технологію паралельного пошуку даних. Реалізовано з використанням технології CUDA програмні засоби паралельного сортування масивів даних вдосконаленим методом злиття та методом паралельно-вертикального сортування масивів даних. Розроблено на базі ПЛІС фірми Altera та середовища проєктування Quartus II апаратні засоби паралельно-вертикального сортування та пошуку максимального числа в одновимірному масиві даних.

4.1. Розроблення інформаційної технології паралельного сортування

Розроблення інформаційної технології паралельного сортування даних, ґрунтується на інтегрованому підході, який охопив: дослідження, розроблення методів і алгоритмів паралельного сортування масивів даних; розроблення функціональних моделей алгоритмів паралельного сортування потоків даних у реальному часі; сучасну елементну базу (GPU, ПЛІС) та засоби автоматизованого проектування; нові архітектурні рішення, орієнтовані на технології надвеликих інтегральних схем. Розроблену схему інформаційну технологію паралельного сортування даних наведено на рис.4.1., де P_d – інтенсивність надходження даних; P_c – інтенсивність сортування даних; s – кількість трактів сортування; n_s – розрядність трактів сортування; $T_к$ – такт сортування; L – коефіцієнт узгодженості; v – коефіцієнт врахування особливостей засобів реалізації; $t_м$ – час накопичення масиву даних; $R(O)$ – складність алгоритму сортування у кількості операцій попарного порівняння та перестановки даних; E_o – ефективність використання обладнання.

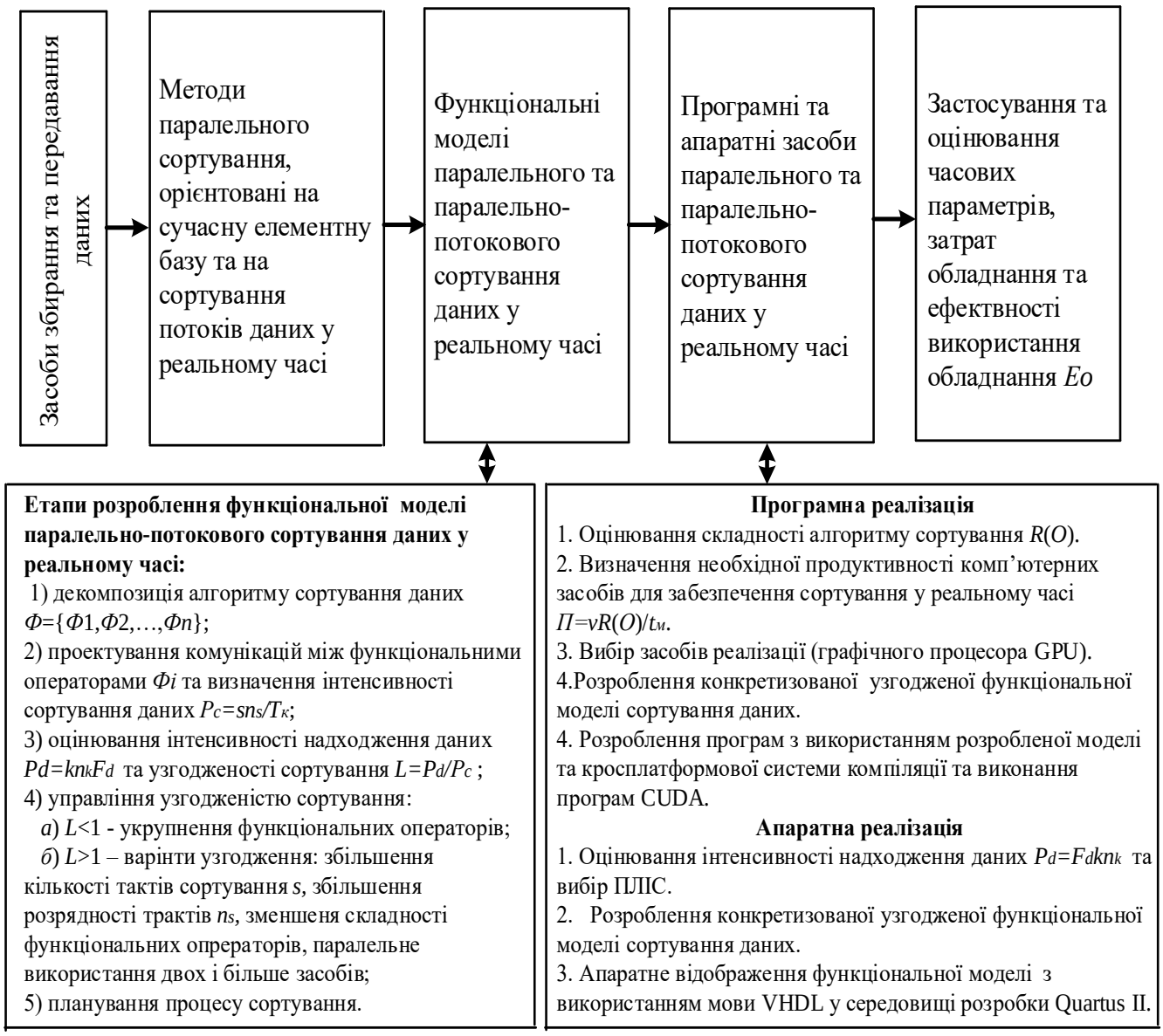


Рис 4.1. Схема інформаційної технології паралельного сортування даних

Реалізована інформаційна технологія паралельного сортування даних ґрунтується на розроблених і вдосконалених методах сортування даних, функціональних моделях і враховує інтенсивність надходження даних, розміри масивів даних, особливості засобів реалізації та забезпечує сортування даних у реальному часі з високою ефективністю використання обладнання.

4.2. Розроблення інформаційної технології паралельного пошуку даних

Також розроблено інформаційну технологію паралельного пошуку даних (рис.4.2), основними компонентами якої є: засоби збирання та передавання даних

розрядними зрізами; розроблені методи паралельно-вертикального пошуку максимальних і мінімальних чисел; функціональні моделі пошуку у реальному часі максимальних і мінімальних чисел у масивах; засоби автоматизованого проєктування апаратного і програмного забезпечення; програмні та апаратні засоби пошуку даних у реальному часі з високою ефективністю використання обладнання.

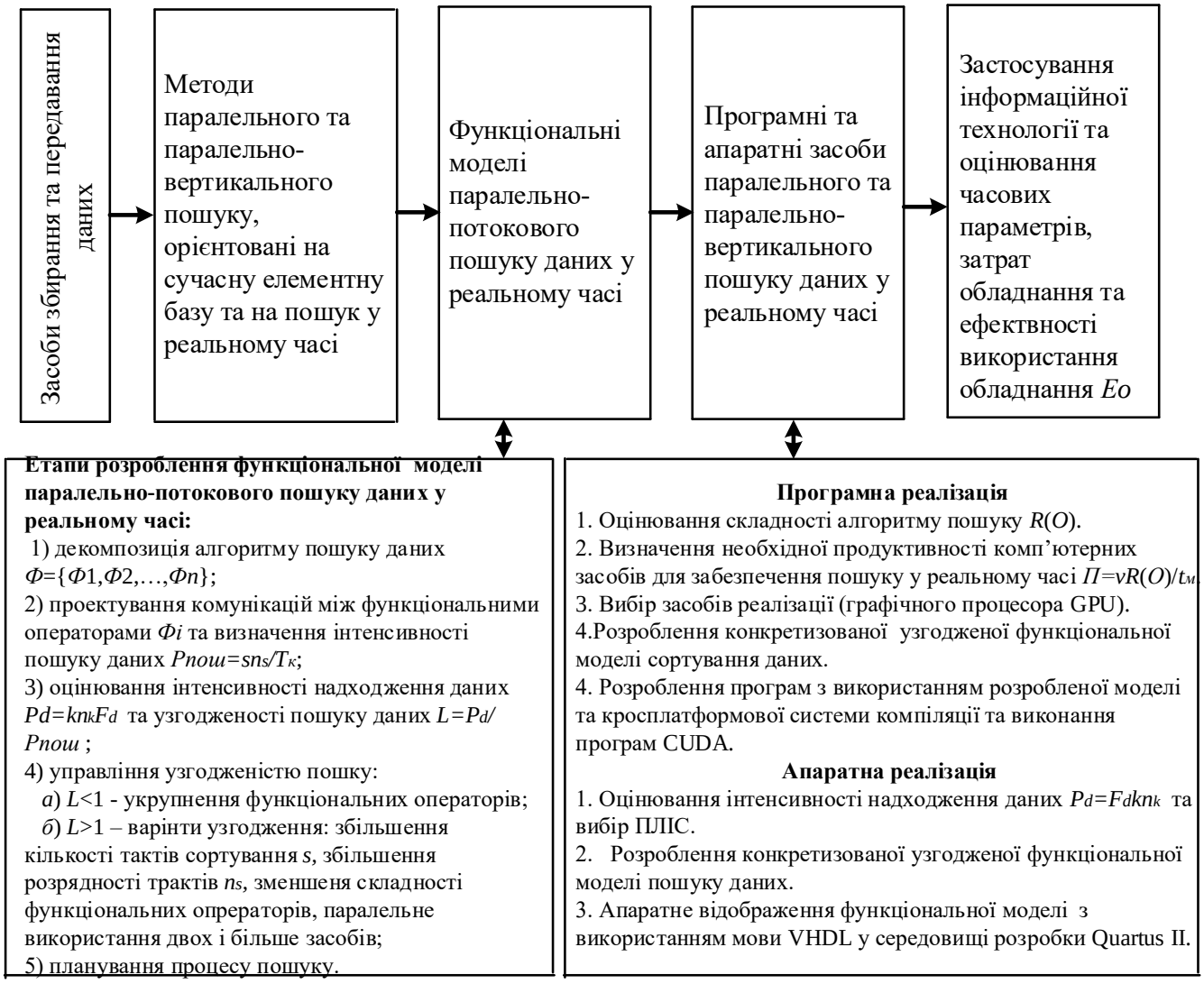


Рис 4.2. Схема інформаційної технології паралельного пошуку даних

4.3. Розроблення програмних засобів для паралельно-вертикального сортування масивів даних на графічному процесорі

Розробка програмних засобів інформаційних технологій паралельного сортування даних базується на використанні комплексного підходу, який охоплює:

дослідження, вдосконалення та розробку методів і алгоритмів паралельно-вертикального сортування масивів даних; використання потокових графів алгоритмів паралельно-вертикального сортування; архітектуру графічних процесорів GPU та програмну модель паралельних обчислень CUDA.

Для реалізації засобів ІТ паралельного сортування даних та ІТ паралельного пошуку даних було використано багатоядерний графічний процесор nVidia GeForce RTX 2060 який включає 1920 ядр та підтримує архітектуру і програмну модель CUDA. Технічні характеристики графічного процесору вказано в таблиці 4.1 [94].

Таб. 4.1.

Характеристики відеокарти nVidia GeForce RTX 2060

Параметр	Опис
Графічний процесор	Розмір: 12 нм Кількість транзисторів: 10, 800 млн Розмір матриці: 445 мм^2
Частота	Базова: 1365 МГц Підсилена: 1680 МГц Пам'ять: 1750 МГц
Пам'ять	Розмір: 6 Гб Тип: GDDR6 Шина: 192 Біт Пропускна здатність: 336.0 Гб/с
Підтримка програмних засобів:	DirectX - 12 Ultimate (12_2) OpenGL - 4.6 OpenCL - 1.2 Vulkan - 1.2 CUDA - 7.5 Shader Model - 6.5
Кількість ядр	1920

Ціна	350$

Оскільки CUDA це програмна модель, орієнтована на багатоядерну архітектору для ефективного використання обладнання необхідно використовувати підходи та принципи паралельного програмування. Основна відмінність програмної реалізації алгоритмів на CUDA(GPU) в порівнянні з програмуванням на центральному процесорі, це використання потоків, блоків та сіток. Потік у графічному процесорі це основний елемент даних, що підлягають обробці. На відміну від потоків процесора, потоки CUDA надзвичайно "легкі", що в свою чергу забезпечує, невеликі затрати на операції обміну між потоками. В свою чергу блоки, це об’єднання від 64 до 512 потоків. В графічних процесорах блоки складаються в сіток. Такий підхід до групування надає перевагу в тому, що кількість блоків, які одночасно виконуються графічним процесором напряму пов’язані з апаратними ресурсами, що в свою чергу призводять до високого рівня масштабування програми. Оскільки ядро програми за один виклик виконується для всіх потоків сітки і не звертає уваги на наявні ресурси графічного процесора. Якщо обладнання має недостатньо ресурсів, воно виконує блоки послідовно; і навпаки, якщо ресурсів є достатня кількість, опрацювання блоків відбувається паралельно. Це означає, що однаковий код програми ефективно виконується як на відносно простих, так і на складніших графічних процесорах [41].

Загальна схема виконання алгоритму сортування методом злиття, де базові операції розподіленими між блоками і потоками для масиву з 16-и чисел наведено на рис.4.3.

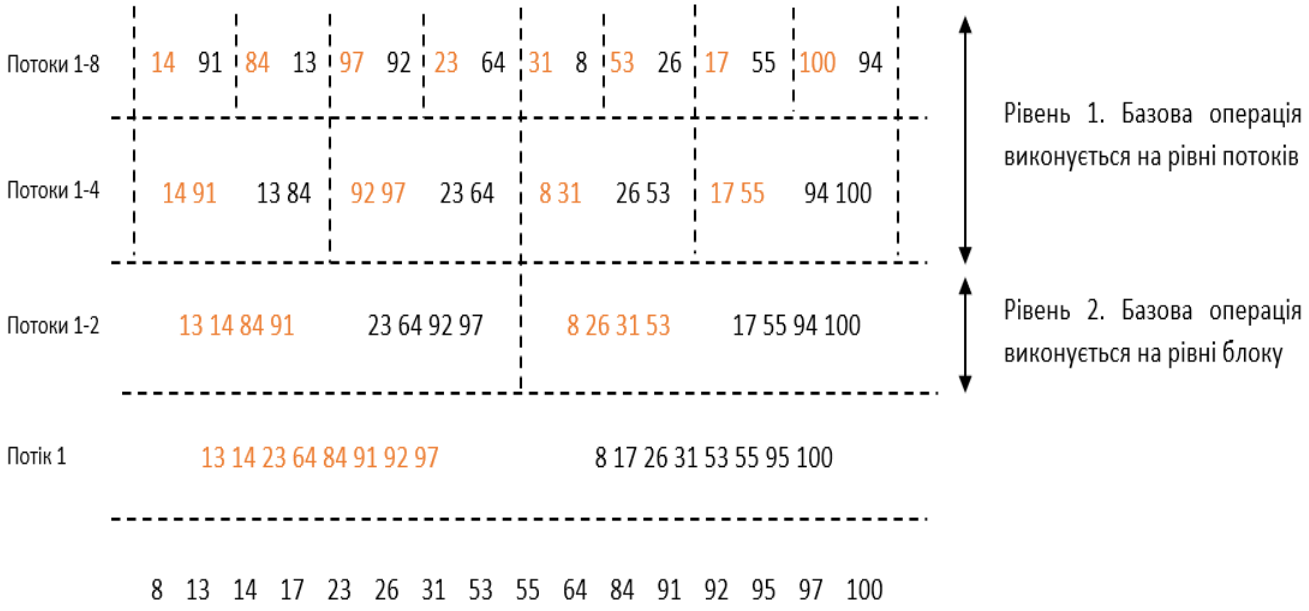


Рис 4.3. Схема алгоритму сортування методом злиття на графічному процесорі

Для ефективного використання обкладення, стандартизації і зменшення ціни програмних засобів інформаційних технології паралельного сортування та пошуку даних було оправно стек програмних засобів від компанії Microsoft. А саме наступні компоненти:

- Сервер застосувань. Для реалізації було вибрано Windows Server. Він дозволяє швидко розгортати програмне забезпечення та маєш підтримку великого спектру додаткових апаратних засобів. На ньому розташовані програмні засоби розроблені на основі методів з розділу 2 і 3 дисертаційної роботи.

- Сервер бази даних. Для збереження і опрацювання даних обрано MS SQL Server. Перевагою цієї бази даних є те, що вона дозволяє зберігати і легко опрацьовувати великі масиви даних. Крім цього MS SQL Server, це комплексна система, яка в себе включає і засоби СППР, такі як Reporting services і Analysis services.

- Сервер безперервної інтеграції (CI) і безперервного розгортання (CD). Для інтеграції обрано сервер – Jenkins і систему централізованого збереження коду Git [100, 101], що забезпечує легкість здійснення розгортання нових версій програмного комплексу.

Загальна структура інформаційних технологіх паралельного сортування та пошуку даних зображена на рис. 4.4.

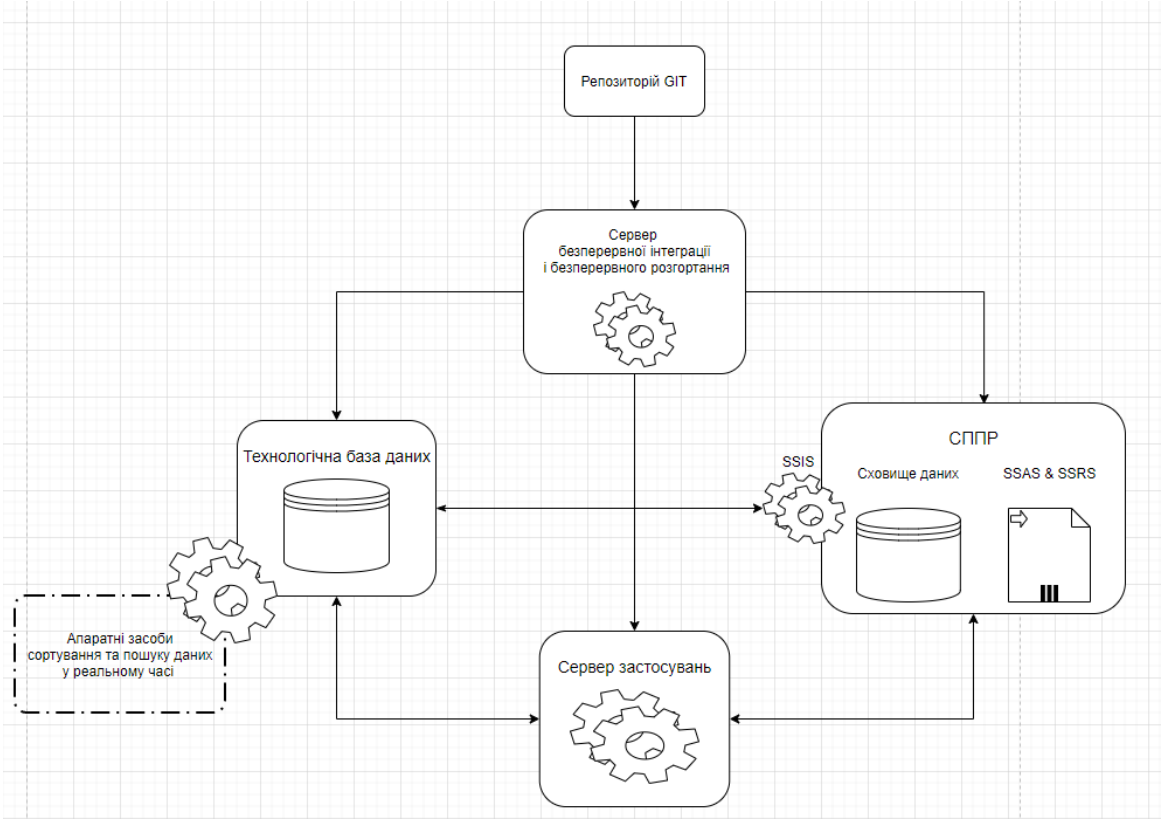


Рис 4.4. Загальна структура програмних засобів інформаційних технологій паралельного сортування та пошуку даних

Відображена на рис 4.4. структура програмних засобів є модульною. Кожний сервер конфігурується окремо і не залежний від інших.

Опис основних модулів програмного засобу. Програмний засіб паралельного сортування та пошуку даних було розділено на сім модулів (рис. 4.5) в залежності від функцій, що вони виконують.

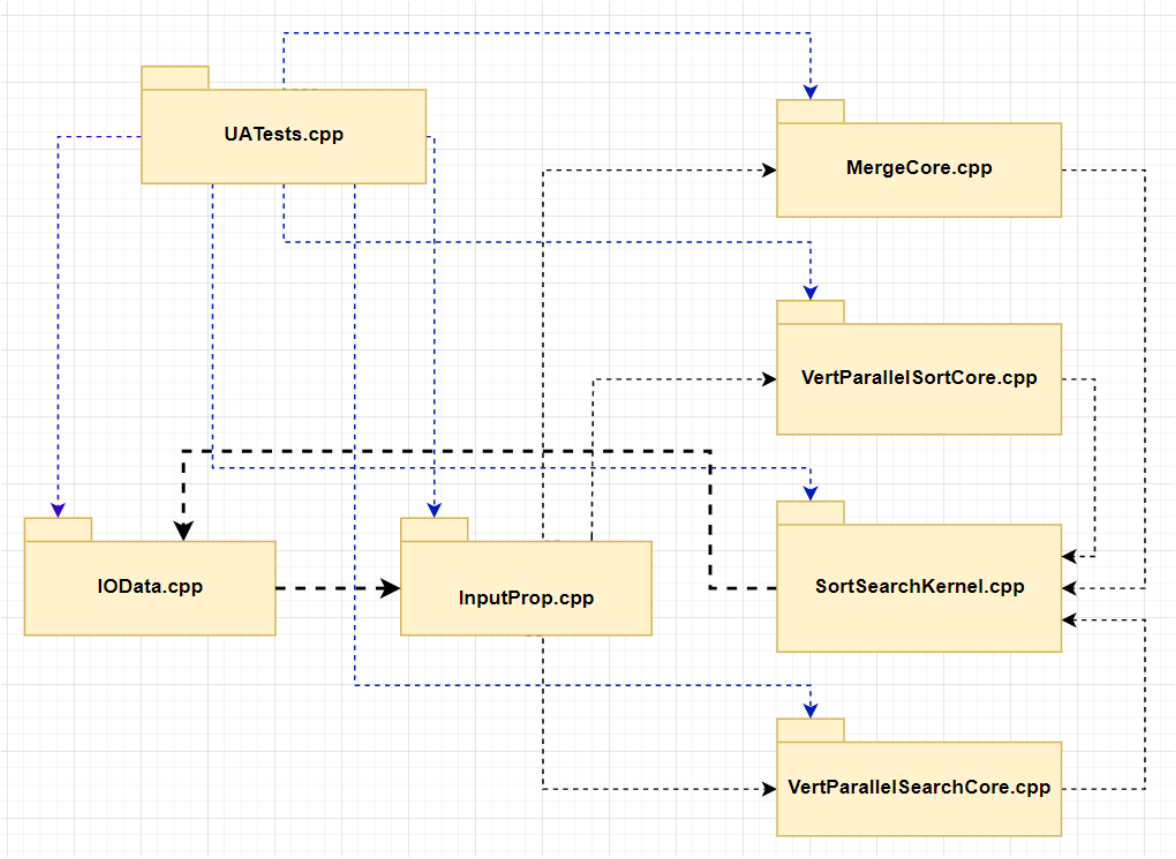


Рис 4.5. Діаграма модулів програмних засобів паралельного сортування та пошуку даних

До цих модулів відносяться:

- *IOData.cpp* – модуль для роботи з базами даних, використовується для вичитки і запису даних;
- *InputProp.cpp* – модуль, який на основі розміру масивів та параметрів засобів їх реалізації (багатопроцесорні системи та багатоядерні процесори) вибирає методи сортування та пошуку даних;
- *SortSearchKernel.cpp* – основний модуль, який виконує сортування чи пошуку даних на основі вибраної базової операції;
- *MergeCore.cpp* – модуль, який включає в себе базові операції паралельного сортування методом злиття;
- *VertParallelSortCore.cpp* – модуль, який включає в себе базові операції паралельно-вертикального сортування;

- *VertParallelSearchCore.cpp* – модуль, який включає в себе базові операції паралельно-вертикального пошуку мінімальних і максимальних чисел;
- *UATests.cpp* - містить юніт тести та інтеграційні тести модулів, що входять до програмного засобу ІТ.

Структури бази даних. Основним джерелом даних для програмних засобів паралельного сортування та пошуку даних служать бази та сховища даних. Так як програмні засоби розроблялись у межах державних програм, було вирішено розробити два типи баз даних для багаторівневого управління енергоефективністю регіону. Для реалізації баз даних вибрано платформу MS SQL Server. Перша база – це технологічна база даних. Тобто це OLTP база даних в якій надходження і зміна даних відбувається в реальному часі. Дані в такій базі зберігаються не тривалий період часу, так як відбувається стрімкий ріст об’єму даних за рахунок надходження показників з великої кількості давачів, що в свою чергу призводить до стрімкої втрати продуктивності. Вона використаються для спостереження стану системи в режимі реального часу. Структура бази даних зображена на рис.4.6. Другий тип – це сховище даних у вигляді «зірки». Сховище даних місить великі масиви історичних даних. Дані використовуються для проведення аналізу. Структура сховища даних зображена на рис.4.7. Міграція даних між OLTP і СД відбувається за рахунок використання пакетів інтеграції даних SSIS. СППР побудована на вбудованих засобах MS SQL Server-а, а саме на SSRS і SSAR.

Для роботи з базами даних використовується модуль IOData.cpp з діаграми 4.5.

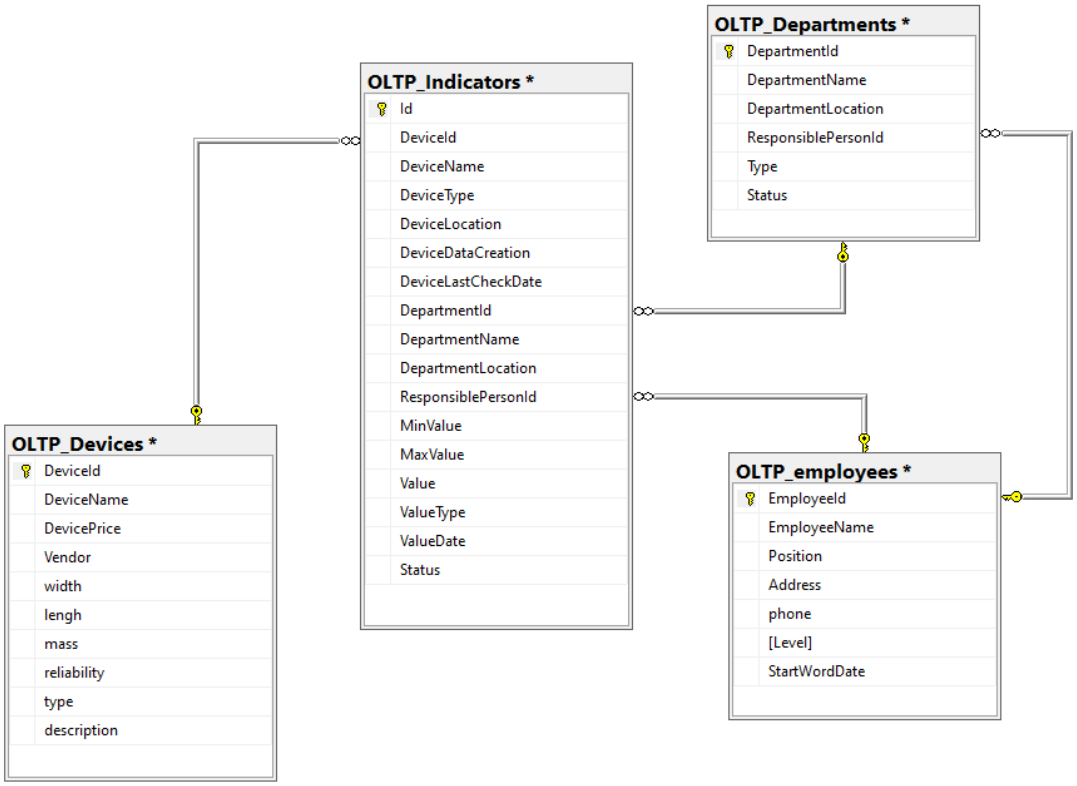


Рис. 4.6. Діаграма технологічної бази даних

Основною таблицею OLTP системи є dbo.OLTP_Indicators. В цю таблицю надходять показники зі всіх давачів підприємства. Показники діляться по типу і локації – департаменту (цеху і тп). В таблицях dbo.OLTP_Devices, dbo.OLTP_Departments, dbo.OLTP_employees міститься додаткова інформація про прилади, департаменти (цехи) і працівників.

Як вже було сказано, міграція дані між системами реалізована на основі служб MS SQL Server-а, а саме SSIS. За допомогою пакетів, відбувається очищення даних, їх групування по сутностях і приведення до одного стандарту. В результаті міграції ми отримуємо дані, які відповідають структурі сховища даних, яке зображено на рис. 4.7.

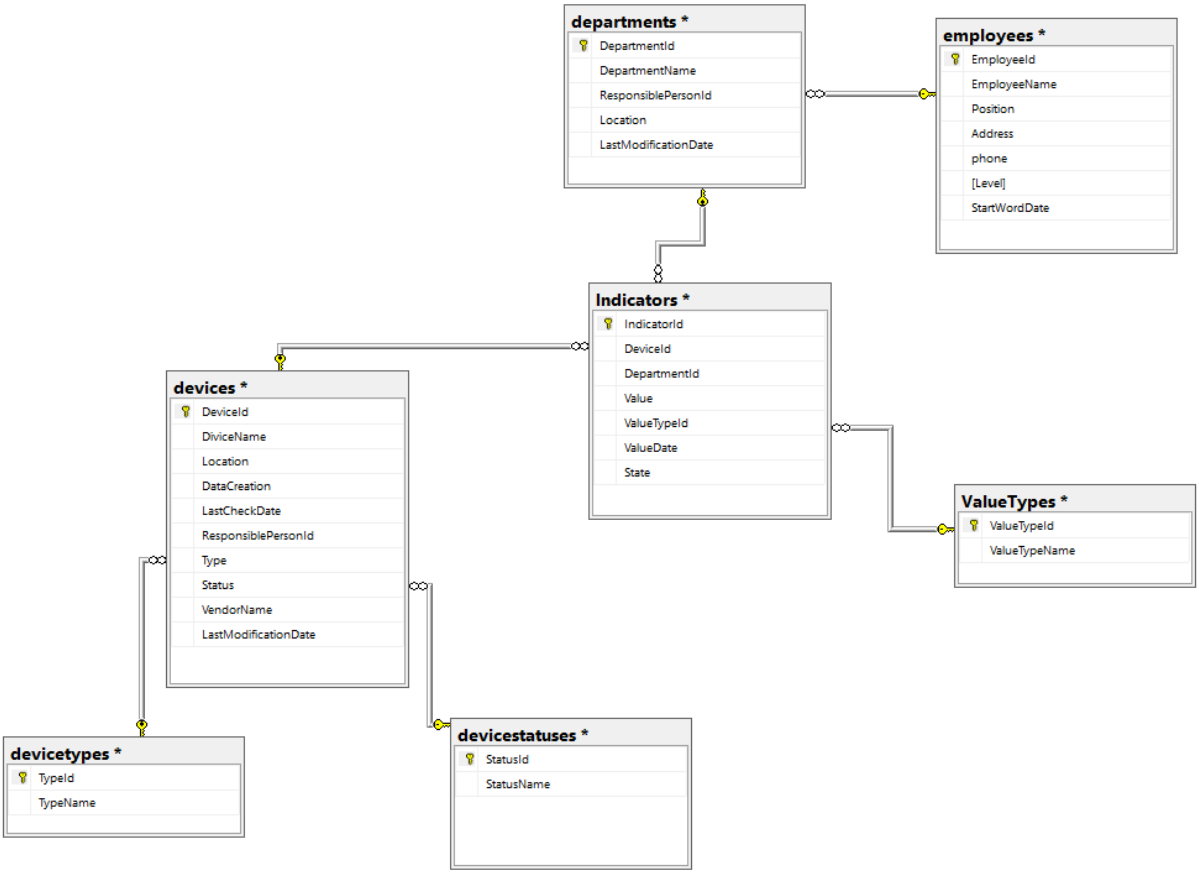


Рис. 4.7. Діаграма сховища даних

Сховище даних побудовано на основі архітектури «зірка» [100]. Для кожної сутності було створено окремі таблиці-виміри: dbo.devices, dbo.departments, dbo.employees та таблиці-довідники: dbo.devicetypes, dbo.devicestatuses, dbo.ValueTypes. Основною таблицею, являється таблиця фактів - dbo.Indicators. Дані які не були потрібні для аналізу були видалені на етапі міграції.

Аналіз проводиться на основі служб MS SQL Server-а, а саме SSRS – для побудови звітів, SSAS – для побудови багатовимірних кубів, та мови роботи з даними T-SQL.

Програмна реалізація методу паралельно-вертикального сортування даних виконувалась в межах розробки модуля *VertParallelSortCore.cpp* зображеного на рис. 4.5. Для ефективної реалізації методу П-ВС було розроблено блок-схему алгоритму для центрального процесору, яка наведена на рис.4.8. Перетворення даних у цій послідовності означає, трансформацію даних спочатку у бінарне представлення та

наступне виділення із цих даних порозрядних зрізів. Утворення порозрядних зрізів здійснюється транспонуванням матриці утвореної із бінарних представлень вхідних даних.

До прикладу, представимо, що у нас є матриця A, стрічками якої є двійкові представлення чисел починаючи із старшого розряду. Тоді A^T і буде шуканою нами матрицею, стовпцями якої стануть порозрядні представлення кожного числа, а стрічками відповідні розрядні зрізи починаючи від старшого розряду кожного числа. Розмірність такої матриці буде рівна $n * N$, тобто n стрічок (розрядів) та N стовпців (чисел). Відповідно у глобальній пам'яті GPU будуть такі затрати на пам'ять як $O(2*n*N)$ для вхідної матриці та вихідної сумарно. Такі затрати на пам'ять необхідні для забезпечення повністю незалежної обробки даних кожним потоком GPU. Тобто читання даних із одної комірки даних може виконуватися кількома потоками, але при цьому запис не буде пересікатися між потоками.

Окрім цього, максимальними затратами локальної пам'яті в кожному потоці GPU буде O(N) для збереження діапазону роботи (визначення безпосередніх номерів вхідних чисел над якими проводиться обробка). Використання такої додаткової інформації здійснюється для уникнення блокувань чи бар'єрів між потоками, що значно б сповільнило роботу та зробило неможливим масово паралельний підхід виконання сортування.

Рис.4.8. Блок-схема алгоритму паралельно-вертикального сортування для CPU

Кількість потоків GPU (CUDA) на яких виконуватиметься обробка рівна кількості чисел у матриці, а саме N. Проте вибір кількості блоків із потоками відбувається за наступною формулою:

$$N_{block} = ceil\left(\frac{N}{N_{thread}}\right), \quad (4.1)$$

де N – кількість чисел у вхідному масиві; N_{thread} – максимально можлива характеристиками GPU кількість потоків у блоці, що є степенем 2.

Алгоритм роботи кожного потоку П-ВС (рис.4.9) складається з наступних кроків:

1) отримати посилання на глобальну матрицю вхідних даних;

2) присвоїти для поточного потоку діапазон обробки даних (діапазон в якому елемент здійснюватиме пошук числа кількості одиниць, що необхідне для формування вихідного значення);

3) порахувати кількість одиниць у зрізі для поточного діапазону;

4) якщо кількість одиниць менша за номер потоку то записати у вихід «1», в іншому випадку – «0»;

5) обрахувати новий діапазон значень, до якого увійдуть ті значення із попереднього діапазону, у яких елементи на відповідних позиціях вхідного масиву будуть рівні значенню виходу;

6) якщо поточний потік записав на вихід значення «0», то сформувати для нього нове значення номеру потоку за формулою:

$$n_i = n_i - R_i, \tag{4.2}$$

де n_i-номер і-го потоку, R_i-сумарна кількість одиниць у діапазоні і-го потоку.

7) якщо є наступний зріз у вхідній матриці, то перейти до кроку 3, в іншому випадку завершити виконання потоку.

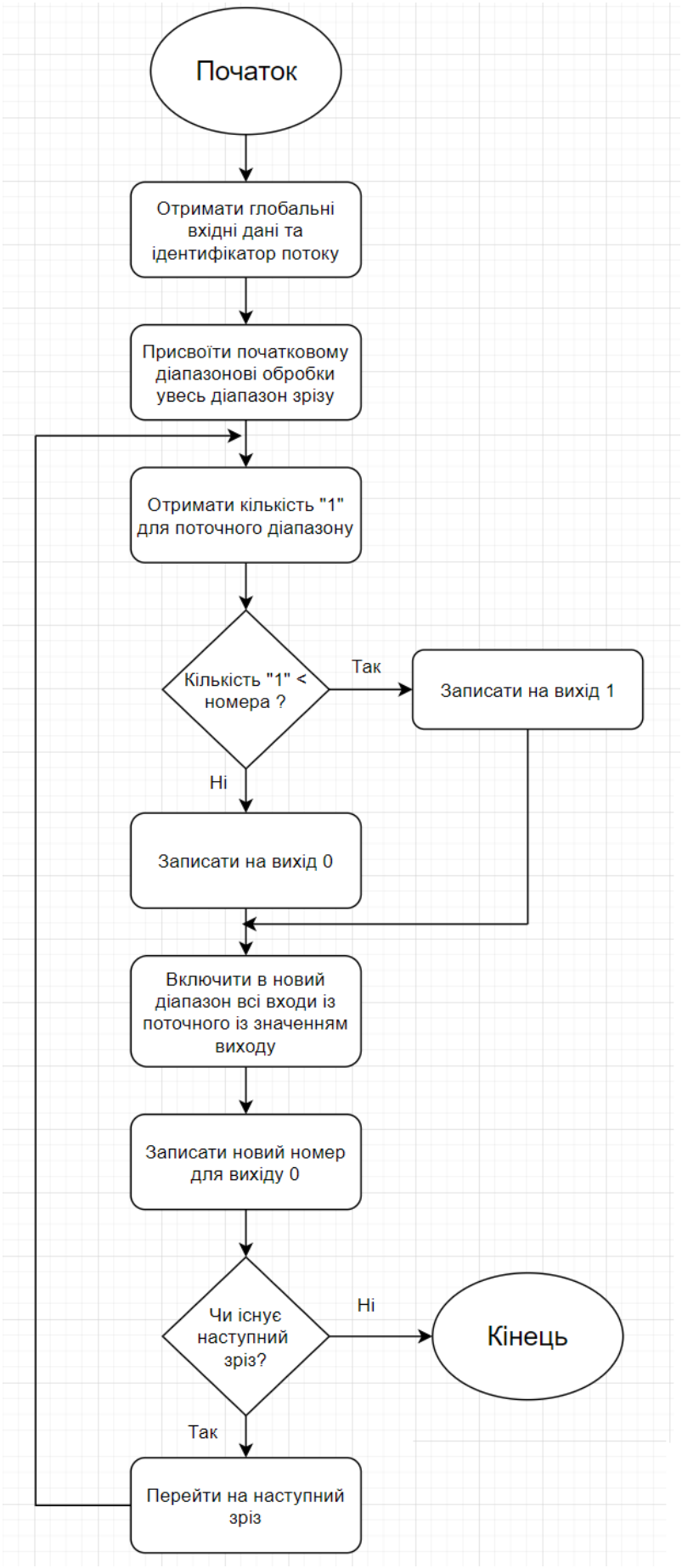


Рис.4.9. Блок-схема алгоритму роботи потоку.

Наступний псевдо-код на демонструє описаний вище алгоритм роботи кожного потоку CUDA:

```
@cuda.jit(«void(byte[:,:], byte[:,:])»)
def sort(sharedInput, out):
        id = cuda.grid(1)
        threadNumber = id

 workingRange = cuda.local.array(digits, numba.uint16)
for i in range(digits):
        workingRange[i] = i
                for position in range(positions):
                        numberOfOnes = 0
                        integers = sharedInput[position]
                                for i in workingRange:
                                  if (i > -1): numberOfOnes += integers[i]
                                  if (numberOfOnes == 0):
                                          out[id, position] = 0
                                  continue
                        if (threadNumber < numberOfOnes):
                                  output = 1
                        else:
                                  output = 0
                out[id, position] = output
                if (position != positions – 1):

                 # changeRange function
                 i = 0
                 integers = sharedInput[position]
                          for integer in workingRange:
                                  if (integers[integer] == output):
                                   workingRange[i] = integer
                                  i += 1
                                          for element in range(i, len(workingRange)):
                                                  workingRange[element] = -1
                                                  # changeThreadNumber function
                                                  if (i == 1):
                                                          threadNumber = 0
                                                          elif (output == 0):
                                                    threadNumber -= numberOfOnes
```

Параметрами анотації @cuda.jit є типи вхідних даних, в даному прикладі, це два двовимірних масиви типів байт, що відповідають вхідній та вихідній матриці. Код ядра викликається із переданими в нього кількістю блоків за формулою 4.1. та кількістю потоків у блоці, вхідною матрицею та вихідною матрицею.

Визначення глобальної позиції потоку здійснюється бібліотечною функцією cuda.grid(1), параметр якої визначає позицію потоку у одновимірному представленні всіх обчислювальних блоків CUDA.

Оцінка результатів виконання паралельно-вертикального сортування. У таблиці 4.2 представлено результати моделювання алгоритмів порозрядного сортування, паралельно-вертикального сортування на базі потоків CPU, паралельно-вертикального сортування на базі CUDA. Реалізація П-ВС на графічних процесорах

показує кращу швидкодію, ніж реалізація на CPU, але через неефективне управління ресурсами та обмежену у розмірі пам’ять відеоадаптера порівняно із ОП пристрою, виграш у швидкодії у найкращому випадку лише в 4 рази.

Таб.4.2.

Результати моделювання паралельно-вертикального сортування

Кількість чисел/розрядів	Порозрядне сортування (мс)	Паралельно-вертикальне сортування на CPU (мс)	Паралельно-вертикальне сортування на CUDA (мс)
10 / 10	1.21 ± 0.3	0.6 ± 0.04	0.4 ± 0.03
10 / 20	1.4 ± 0.15	0.69 ± 0.03	0.48 ± 0.02
1000 / 10	88.1 ± 2.4	40.478 ± 2.53	12.32 ± 0.81
1000 / 20	90.32 ± 4.191	48.75 ± 3.8	14.332 ± 1.4
5000 / 10	6031 ± 608.8	1602.7 ± 601.62	702.5 ± 20.311
5000 / 20	8119.3 ± 590	2071.2 ± 592	798.11 ± 53.8
10000 / 10	40201 ± 2901	12472 ± 1095.2	7098.3 ± 230.12
10000 / 20	45130 ± 3412	15001 ± 2031.4	8187.51 ± 580.3

Порівняльний графік зображено на рисунку 4.10.

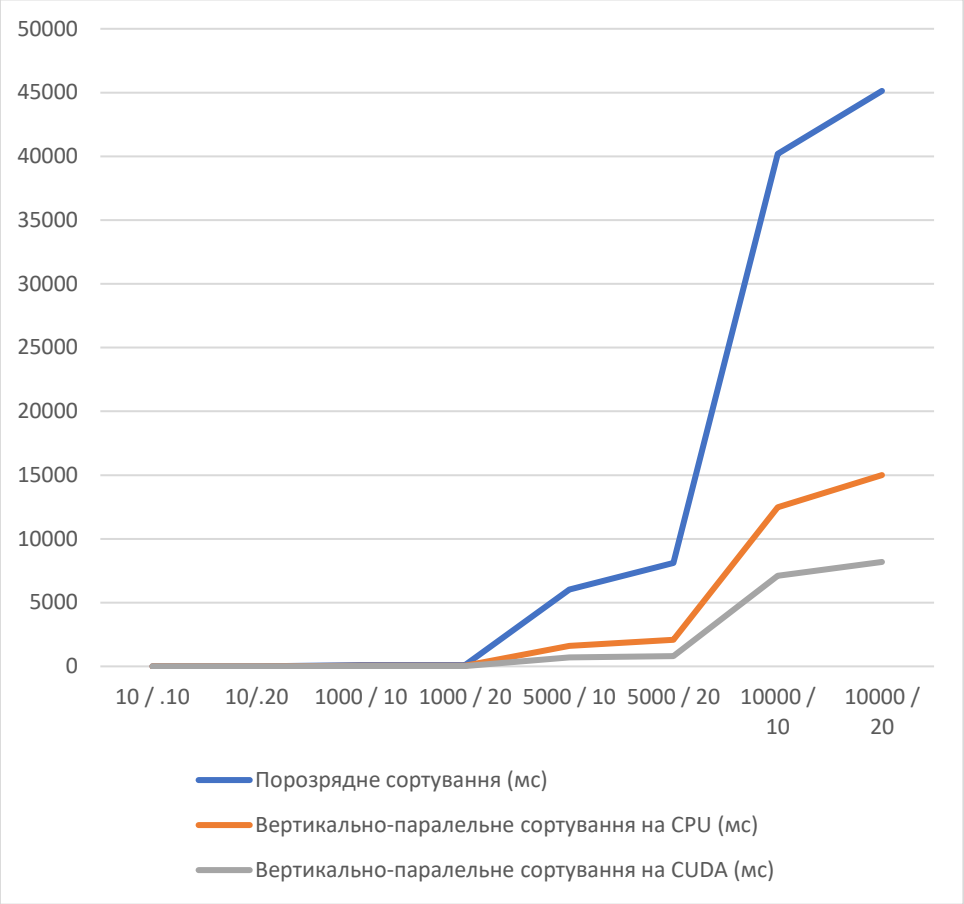


Рис 4.10. Графік порівняння результатів

4.4. Програмна реалізація паралельно-вертикального пошуку максимальних і мінімальних чисел

Програмна реалізація методу паралельно-вертикального пошуку максимальних і мінімальних чисел виконувалась в межах розробки модуля *VertParallelSearchCore.cpp* зображеного на рис. 4.5.

Метод паралельно-вертикального пошуку максимальних і мінімальних чисел у масивах на графічному процесорі з допомогою CUDA було реалізовано у 2 варіаціях:

1) із використанням локальних бар'єрів, з наступними особливостями:
   - затрати по глобальній пам'яті GPU O(n * N), де n – кількість розрядів чисел; N – кількість чисел у масиві;
   - затрати по локальній пам'яті GPU O(N + n);

- використання локальних бар’єрів, а отже кількість одночасно виконуваних потоків обмежена максимальною кількістю потоків у блоці– N_{thread};
- основні обрахунки згідно формул (3.24) – (3.29) виконуються на GPU;

2) з ітеративним процесом CPU + GPU, що має наступні особливості:

- затрати по глобальній пам’яті GPU O($5N$), де N – кількість чисел у масиві;
- локальна пам’ять GPU не використовується;
- кількість одночасно виконуваних потоків рівна кількості чисел N;
- основні обрахунки згідно формул (3.24) – (3.29) виконуються на GPU та редукція формул (3.24) і (3.27) на CPU в послідовному режимі.

На рисунку 4.11 наведено блок-схему алгоритму із використанням бар’єрів для пошуку максимального чи мінімального числа із застосуванням локальних бар’єрів, де необхідно виконати наступні кроки:

1) конвертувати масив чисел у представлення бінарної матриці розрядів цих чисел;

2) на хвостовій частині програми необхідно визначити кількість виконуваних потоків за формулою:

$$R = \min(N, N_{thread}), \tag{4.3}$$

де N – розмірність вхідного масиву чисел; N_{thread} – максимальна кількість потоків у групі (блоку) GPU;

3) на GPU визначити кількість опрацьовуваних елементів одним потоком за формулою:

$$N_e = ceil\left(\frac{N}{R}\right), \tag{4.4}$$

де N – кількість чисел у вхідному масиві; R – кількість потоків із формули (4.3);

4) на GPU заповнити слово управління значеннями 1, кожен потік заповнює діапазон $[N_e * i, Ne * (1 + i)\ ]$, де i – номер потоку у блоці;

5) перший потік у групі обраховує P і D згідно формул (3.24) – (3.26) чи (3.27) – (3.29);

6) ставиться локальний бар’єр для всіх потоків у групі (блоку);

7) кожен потік у групі обраховує наступне слово керування згідно формули (3.24) чи (3.27) у діапазоні $[N_e * i, Ne * (1 + i)$];

8) встановлюється локальний бар’єр для очікування завершення обрахунку слова керування усіма потоками;

9) якщо є наступний зріз то повторюються всі кроки GPU з 5 по 8.

Причому слово керування завжди зберігаються у локальній пам’яті групи потоків GPU як і значення бітів максимального / мінімального числа.

Такий підхід ефективний для невеликих масивів даних, які ми можемо повністю загрузити у пам’ять GPU і тим самим забезпечити ефективну обробку без постійного обміну із RAM пам’яттю хоста.

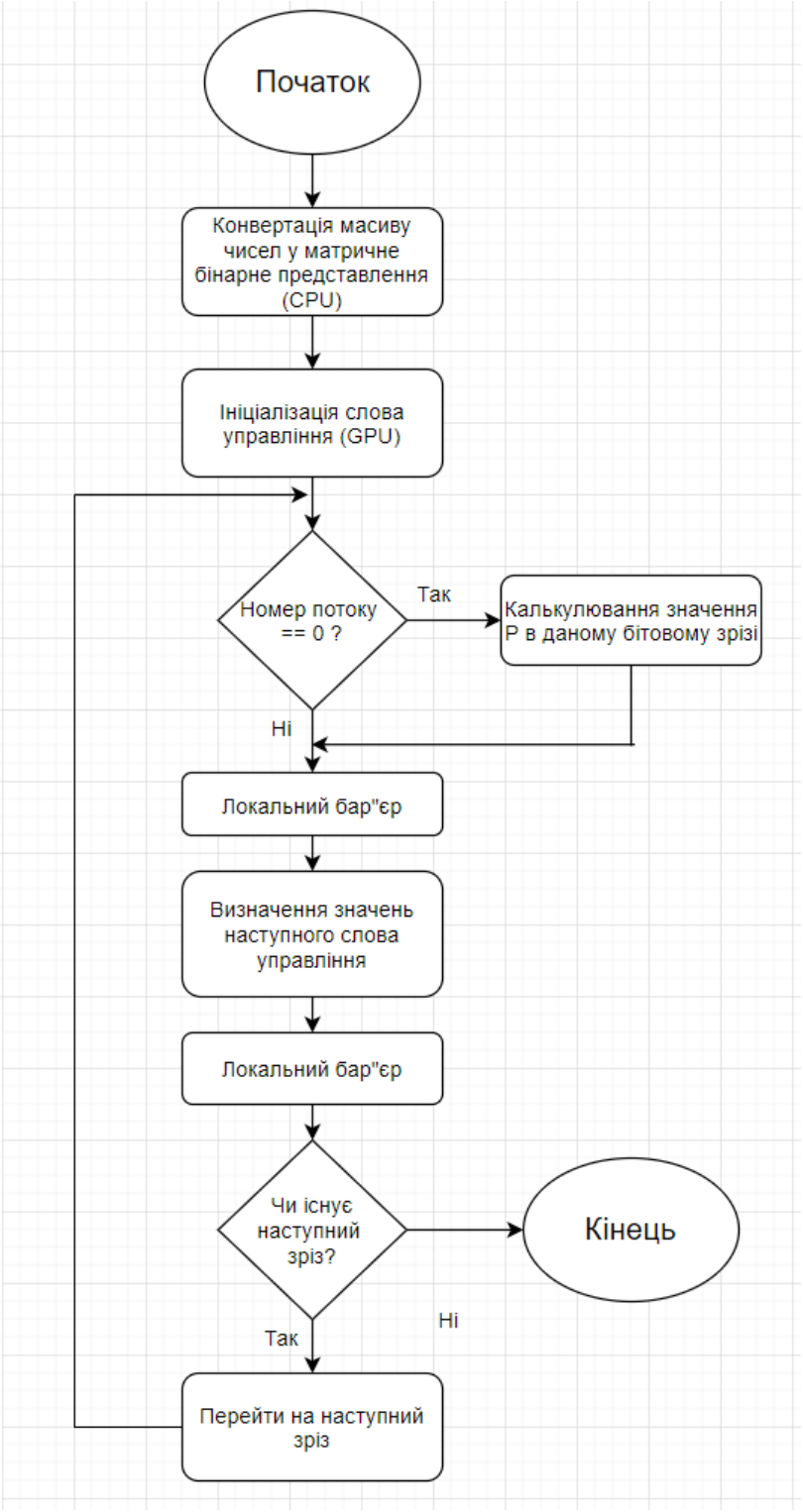


Рис.4.11 Блок-схема алгоритму пошуку максимального чи мінімального числа із використанням бар’єрів.

На рисунку 4.12 наведено блок схему алгоритму пошуку максимального чи мінімального числа із застосуванням ітеративного підходу комбінування CPU та GPU, при якому необхідно виконати наступні кроки:

1. конвертувати масив чисел у представлення бінарної матриці розрядів цих чисел;

2. на хостовій частині програми проініціалізувати слово керування одиницями;

3. на GPU визначити за формулами (3.24) чи (3.27) значення наступного слова керування;

4. на GPU визначити часткові значення P;

5. здійснити редукцію значень P на хостовій частині програми та визначити відповідно D;

6. якщо є наступний зріз то повторюються всі кроки 3 – 5 із переданими на GPU відповдіно поточним та попереднім побітовим зрізом, а також значенням слова керування знайденим у поточній ітерації.

Таким чином, всі дані, що зберігаються на GPU повинні бути доступні у глобальній пам’яті, а саме два побітових зріза, поточне та обраховане наступне слово керування, часткові значення P.

Такий підхід дозволяє обробляти великі масиви даних, за рахунок збереження у пам’яті GPU невеликої кількості даних, порівняно із попереднім підходом, що використовував локальні бар’єри, цей підхід вимагає постійного обміну і синхронізації даних між GPU та CPU, що спричиняє затрати на передачу даних між ними, а також додає виконання послідовного коду з сторони CPU.

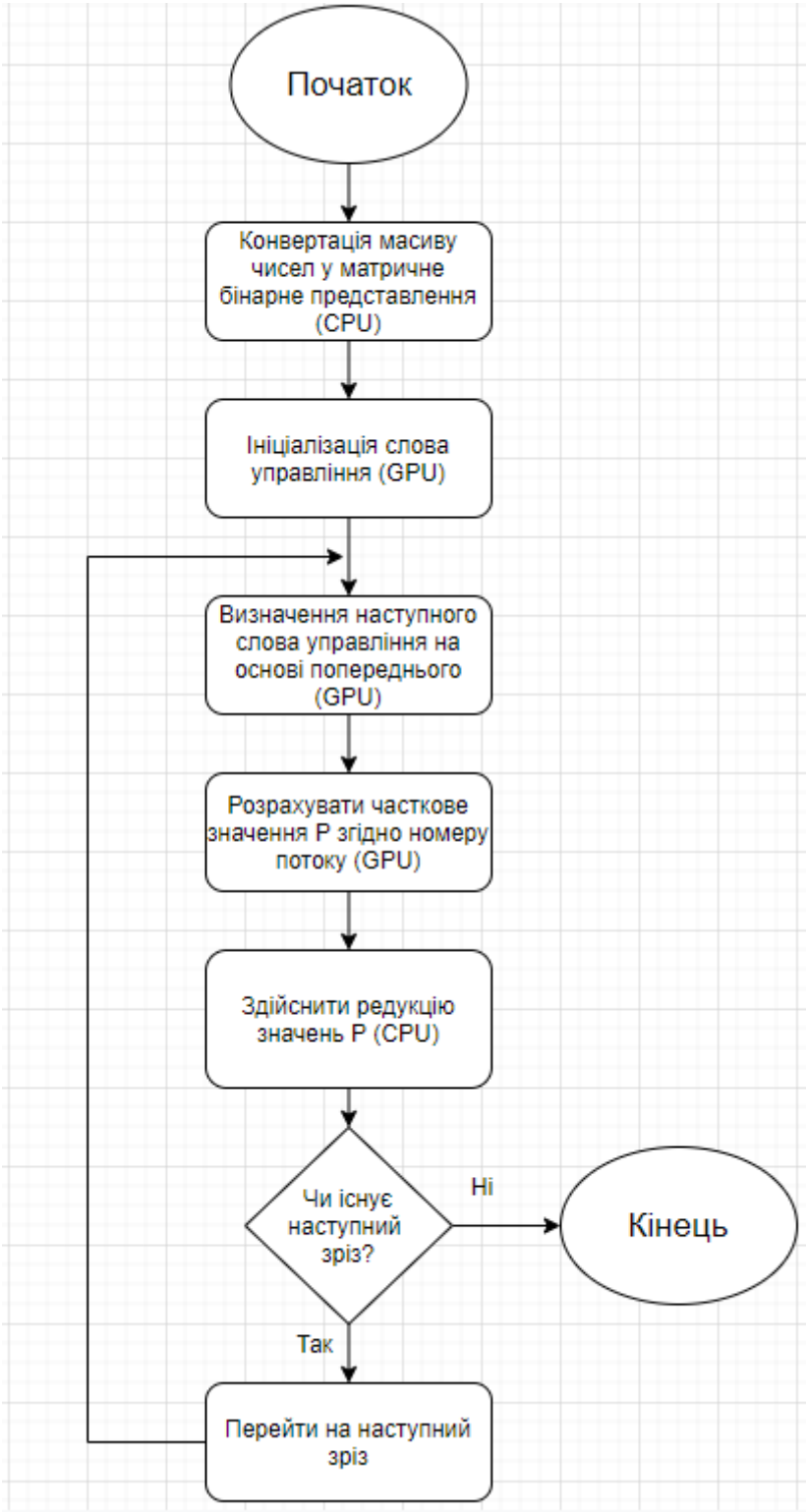


Рис.4.12. Блок-схема алгоритму пошуку максимального чи мінімального числа в масиві з допомогою CPU+GPU.

Оцінювання результатів виконання паралельно-вертикального пошуку максимальних і мінімальних чисел. У таблиці 4.3 представлено результати моделювання звичайного послідовного пошуку максимального/мінімального

елементів, паралельно-вертикального пошуку із застосуванням бар'єрів та паралельно-вертикального пошуку з ітеративним процесом CPU+GPU. Результати це середнє арифметичне між часом пошуку мінімального та часом пошуку максимального елементів у масиві чисел.

Таб. 4.3.

Результати моделювання вертикально паралельного пошуку максимального/мінімального елементів

Кількість чисел/розрядів	Час послідовного пошук (мс)	Час паралельно-вертикального пошуку із локальними бар'єрами (мс)	Час паралельно-вертикального пошуку CPU+GPU (мс)
1000 / 10	3.0010	2.7766	3.2991
1000 / 20	3.1610	2.8796	3.3099
5000 / 10	3.9491	2.6500	3.3192
5000 / 20	3.9560	2.9124	3.7412
10000 / 10	5.2341	2.6180	3.8919
10000 / 20	5.2615	3.0126	4.0131
100000 / 10	10.0234	-	4.2731
100000 / 20	10.9312	-	6.3572

Порівняльний графік зображено на рисунку 4.13.

Рис 4.13. Графік порівняння результатів

4.5. Програмна реалізація вдосконаленого паралельного методу сортування даних злиттям

Програмна реалізація методу паралельно-вертикального сортування даних виконувалась в межах розробки модуля *MergeCore.cpp* зображеного на рис. 4.5. Для ефективної реалізації вдосконаленого паралельного методу сортування злиттям було розроблено блок-схему алгоритму для центрального процесору, яка наведена на рис.4.14.

Реалізація даного методу базується на базових операціях описаних в розділі 2.

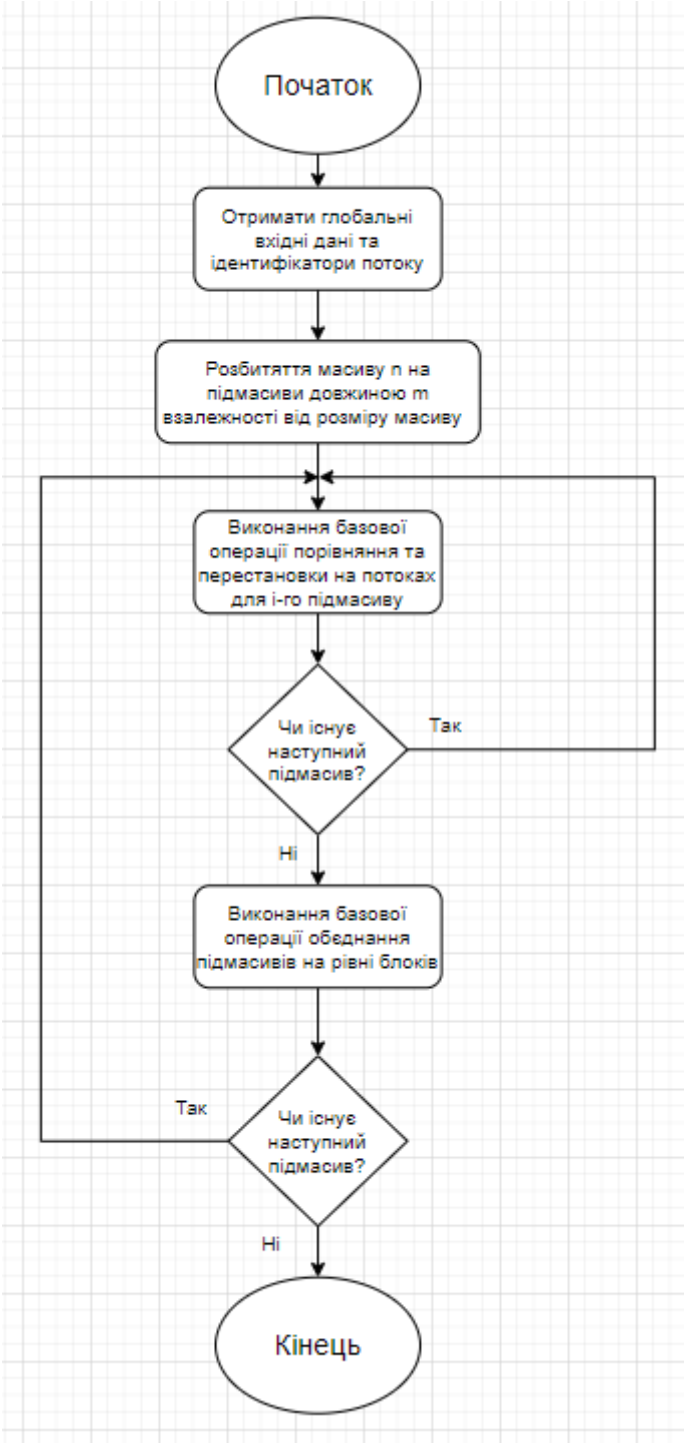


Рис.4.14. Блок-схема алгоритму вдосконаленого паралельного сортування злиттям

Оцінювання результатів виконання вдосконаленого паралельного сортування злиттям. У таблиці 4.4 представлено результати моделювання звичайного послідовного злиття, паралельного злиття з використанням базової операції послідовного об’єднання двох масивів і паралельного злиття з використанням

вдосконаленої базової операції об’єднання двох масивів. Результати це середнє арифметичне між часом сортування та розміром вхідного масиву.

Таб. 4.4.

Результати моделювання вдосконаленого паралельного сортування злиттям

Кількість чисел	Час паралельного сортування злиття з використанням базової операції послідовного об’єднання двох масивів (мс)	Час паралельного сортування злиття з використанням вдосконаленої базової операції об’єднання двох масивів (мс)	Час послідовного злиття (мс)
250000	1.21	0.83248	0.9
500000	1.4	0.9632	1.1
1000000	3.1	2.1328	2.3
1500000	5.4	3.7152	6.1
2000000	6.3	4.3344	8.23536
2500000	9.4	6.4672	12.28768
3000000	12.1	8.3248	15.81712
3500000	20.3	13.9664	26.53616
4000000	34.3	23.5984	44.83696
4500000	43.6	29.9968	56.99392
5000000	56.4	38.8032	77.6064
5500000	65.9	45.3392	90.6784
6000000	73.1	50.2928	105.61488

Порівняльний графік зображено на рисунку 4.15.

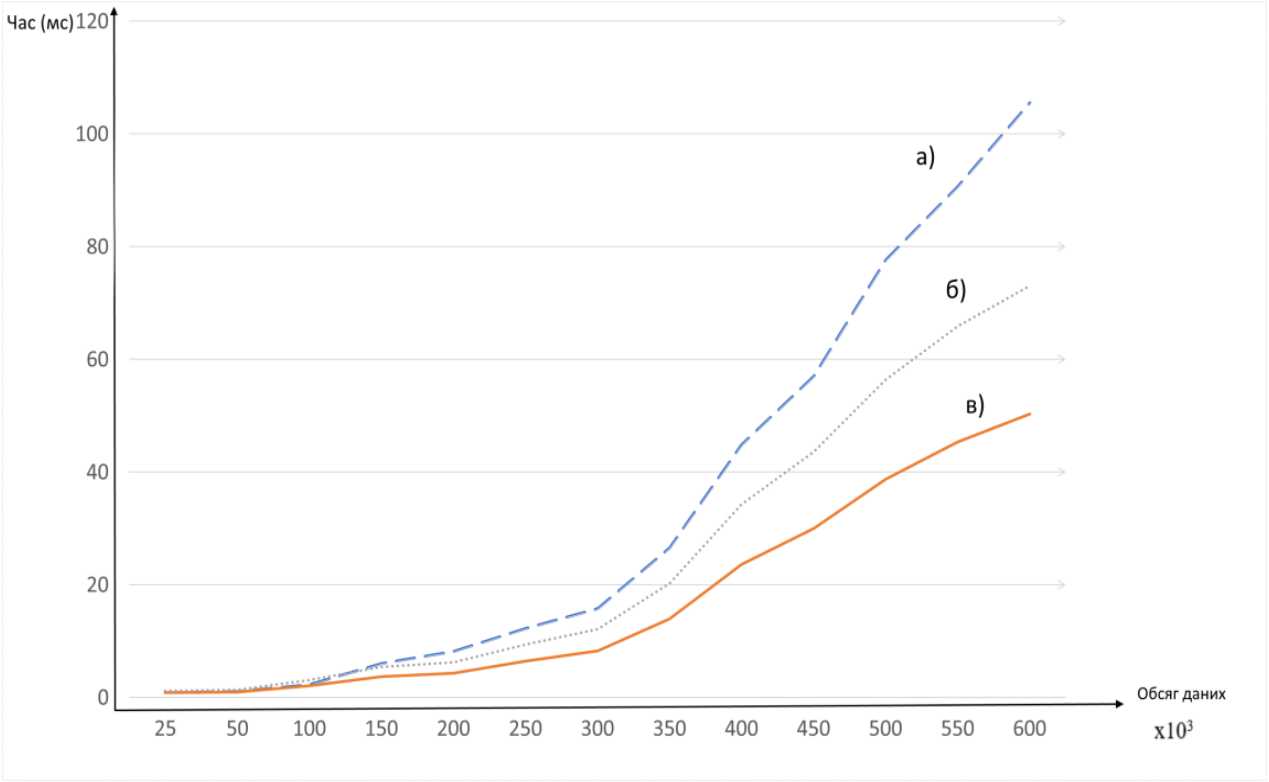


Рис 4.15. Графік часу сортування даних методом злиття:
а) послідовне злиття; б) паралельне злиття з використанням базової операції послідовного об’єднання двох масивів; в) паралельне злиття з використанням вдосконаленої базової операції об’єднання двох масивів

Як показано на графіках з використанням даних отриманих в ході експериментальних досліджень розроблені нові методи паралельно-вертикального сортування даних, пошуку мінімальних і максимальних значень та вдосконалений метод паралельного сортування злиттям показують кращий результат виконання в порівняні з існуючими. Сортування одновимірних масивів даних на графічному процесорі GPU , який завдяки використанню CUDA та вдосконаленого методу злиття зменшує час сортування на 31% порівняно з існуючими програмними засобами сортування злиттям.

4.6. Розробка на ПЛІС FPGA апаратних засобів інформаційних технологій паралельного сортування та пошуку даних

4.6.1. Особливості використання середовища розробки Quartus II для реалізації і моделювання на FPGA апаратних засобів інформаційних технологій паралельного сортування та пошуку даних

Мікросхеми ПЛІС сімейства FPGA використовуються для створення спеціалізованої структури цифрових інтегральних схем. ПЛІС є одними з самих перспективних елементів цифрової схемотехніки. ПЛІС представляють собою кристал, на якому розміщено велику кількість простих логічних елементів. Напочатку ці елементи не з'єднані між собою. З'єднання елементів виконується з допомогою електронних ключів, розміщених на цьому ж кристалі. Електронні ключі керуються спеціальною пам'яттю, в комірки якої заноситься код конфігурації цифрової схеми. З цього випливає, що записавши в пам'ять ПЛІС певні задані коди, отримується можливість синтизувати цифровий пристрій будь-якої степені складності. На відміну від мікропроцесорів, в ПЛІС можна реалізувати алгоритми паралельного сортування та пошуку даних на апаратному (схемному) рівні. При цьому швидкодія реалізації таких алгоритмів зростає.

Основними перевагами технологій для проектування пристроїв на основі ПЛІС є [84]:

- мінімальний час розробки пристрою (необхідно тільки занести в пам'ять ПЛІС конфігураційний код);
- на відміну від звичайних елементів цифрової схемотехніки не потрібно виконувати розробку і виготовлення складних друкованих плат;
- швидке перетворення однієї конфігурації цифрової схеми в іншу (заміна коду конфігурації схеми в пам'яті);
- створення апаратних засобів на основі ПЛІС здійснюється записом конфігурації у пам'ять.

При розроблені пристроїв інформаційних технологій паралельного сортування та пошуку даних з використанням ПЛІС основним інструментом розробки є системи автоматизованого проектування (САПР). Одним з світових лідерів по виробництву ПЛІС є фірма Altera. Для створення цифрових пристроїв на основі своїх виробів Altera розробила дві САПР Max+Plus II і Quartus II. Кожна САПР підтримує всі етапи проектування: ввід проекту, компіляція, верифікація і програмування.

Для ПЛІС фірми Altera використовується середовище розробки Quartus II, яке забезпечує:

- введення за допомогою графічного редактора у пам’ять комп’ютера електричної схеми пристрою;
- перевірку та виправлення помилок;
- визначення параметрів і характеристик апаратних засобів;
- формування файлу конфігурації ПЛІС для конкретної реалізації;
- завантаження в пам’ять ПЛІС файлу конфігурацій.

САПР Quartus II має такі основні можливості:

- різні способи введення структурних описів проекту;
- використання інтегрованих засобів допомоги для створення складних проектів: Mega Wizard і SOPC;
- засоби синтезу;
- засоби розміщення внутрішніх ресурсів та трасування ПЛІС;
- засоби моделювання;
- засоби аналізу в часовій області і аналізу споживаної енергії;
- засоби програмування ПЛІС;
- засоби оптимізації швидкодії LosicLock;
- засоби інтеграції з іншими САПР.

САПР Quartus II надає такі способи введення опису проекту:

1) текстове введення (мови опису апаратури VHDL, AHDL, Verilog);

2) редактор пам’яті (файли Hex, Mif);

3) схемне введення;

4) можливості введення проекту в інших САПР (через EDIF, HDL, VQM);

5) можливості використання мегафункцій та IP- модулів.

Середовище розробки Quartus II підтримує весь процес проектування пристроїв інформаційних технологій паралельного сортування та пошуку даних починаючи з введення проекту користувачем і завершуючи програмуванням ПЛІС, налагодженням як самої мікросхеми, так і пристрою в цілому.

4.6.2 Етапи проектування на ПЛІС апаратних засобів інформаційних технологій паралельного сортування та пошуку даних

Для синтезу засобів інформаційних технологій паралельного сортування та пошуку даних у реальному часі з високою ефективністю використання обладнання доцільно розробити компоненти, які апаратно реалізують базові операції алгоритмів сортування та пошуку даних. Розробку на ПЛІС компонентів інформаційних технологій паралельного сортування та пошуку даних у реальному часі з високою ефективністю використання обладнання та структур паралельного сортування та пошуку даних у реальному часі з високою ефективністю використання обладнання будемо здійснювати на основі інтегрованого підходу. Даний підхід охоплює методи паралельного сортування та пошуку даних, функціональні моделі, ПЛІС та засоби автоматизованого проектування, враховують вимоги конкретних застосувань і інтенсивності надходження даних. Послідовність етапів розробки ПЛІС наведена на рис. 4.16.

Рисунок 4.16 – Послідовність етапів розробки на ПЛІС засобів інформаційних технологій паралельного сортування та пошуку даних

Для забезпечення режиму реального часу необхідно, щоб інтенсивність надходження даних $P_d=knF_d$, де k – кількість каналів надходження даних; n – розрядність каналів надходження даних; F_d - частота надходження даних, була рівна або менша інтенсивності обчислень, які виконує пристрій сортування чи пошуку даних $D_{сор/пош}=m_t n_t F_к$, де m_t – кількість трактів сортування (пошуку) даних; n_t – розрядність траків сортування (пошуку) даних.; $F_к$- тактова частота роботи конвеєра. Узгодженість інтенсивності надходження даних P_d із інтенсивністю сортування чи пошуку даних $D_{сор/пош}$ забезпечує високу ефективність використання обладнання.

4.6.3. Розроблення НВІС-структури для паралельно-вертикального сортування одновимірного масиву чисел

Апаратну реалізацію паралельно-вертикального сортування одновимірного масиву чисел доцільно виконати у вигляді НВІС. Вартість НВІС, яка реалізує паралельно-вертикальне сортування одновимірного масиву чисел, в основному залежить від площі кристала. Площа кристала в основному визначається як кількістю транзисторів, які необхідні для реалізації, так і кількістю зовнішніх виводів, число яких обмежене розміром кристалу та рівнем технології.

Розробку високоефективних НВІС-пристроїв для реалізації паралельно-вертикального сортування одновимірного масиву чисел можна забезпечити при інтегрованому підході, який охоплює:

- розробку нових паралельних методів і алгоритмів сортування масивів даних великої розрядності;
- розроблення нових структурних орієнтованих на НВІС-реалізацівю;
- використання нової елементної бази та засобів автоматизованого проектування НВІС.

При розробці НВІС-структур для паралельно-вертикального сортування одновимірного масиву чисел пропонується використовувати однотипні ПЕ, які з’єднані регулярними зв’язками.

Паралельно-вертикальне сортування одновимірного масиву чисел $\{D_k\}_{k=1}^{N}$ передбачає надходження в кожному такті розрядного зрізу N чисел і його сортування. Надходження таких розрядних зрізів починається з старших розрядів.. Структура НВІС-пристрою паралельно-вертикального сортування одновимірного масиву чисел $\{D_k\}_{k=1}^{N}$ синтезованого на базі розробленого ПЕ наведена рис.4.17, де ТІ – тактовий вхід; ПУ – вхід початкової установки; D_{ki} – вхід i-го розряду k-го числа; ЕУ – елемент управління; ТгД – тригер даних; ТгУ – тригер управління; D_{ki}^{*} - вихід i-го розряду k-го відсортованого числа.

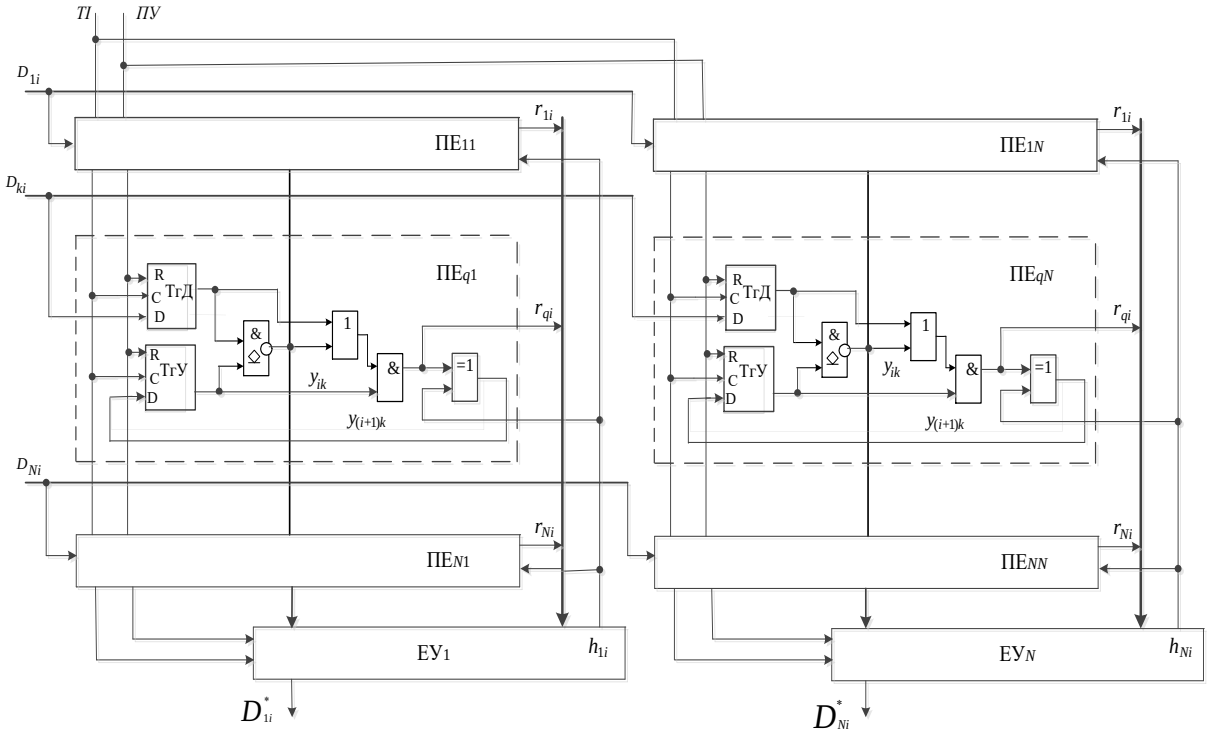


Рис.4.17 Структура НВІС-пристрою паралельно-вертикального сортування одновимірного масиву чисел

Структура НВІС-пристрою паралельно-вертикального сортування одновимірного масиву чисел є матричною і складається з $N\times N$ ПЕ та N ЕУ. Формування i-го розряду k-го відсортованого числа здійснюється за допомогою N ПЕ, які вертикально об’єднанні між собою спільною шиною та утворюють p-й стовпець, де p=1,…,N. Управління кожним p-*м* стовпцем ПЕ здійснюється $ЕУ_p$, структура якого наведена на рис.4.18, де БСм – багатовходовий суматор; Рг – регістр; Від – віднімач; СП – схема порівняння; Тг- тригер.

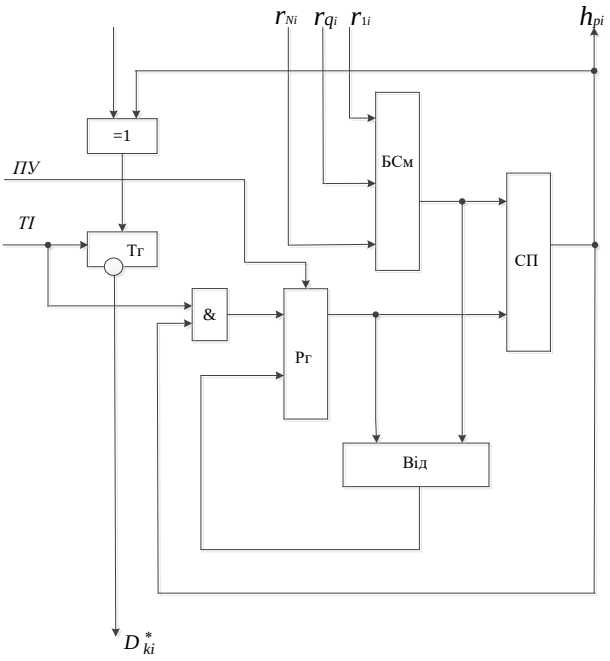


Рис.4.18 Структура *k*-го елемента управління

Перед початком роботи, додатнім імпульсом з входу ПУ всі тригери ТгД та ТгУ у всіх ПЕ встановлюються в лог.0., а в регістр Рг кожного ЕУ$_p$ записується значення p_1, яке дорівнює p. У кожному i-му такті роботи в тригери ТгД процесорних елементів ПЕ$_{q1}$,..., ПЕ$_{qN}$ q-ї стрічки (q=1,…,N) записуються i-й розряд k-го числа, а в тригер ТгУ кожного ПЕ$_{qp}$ - інформація з виходів логічного елемента виключне АБО, яка дозволяє (лог.1), або забороняє (лог.0) участь даних з ТгД у формуванні i-го розряду k-го відсортованого числа. Дані r_{qi} з ПЕ$_q$ кожного p-го стовпчика надходять у ЕУ$_p$, де з допомогою БСм підсумовуються $Z_{pi} = \sum_{q=1}^{N} r_{qi}$. Отримана сума Z_{pi} порівнюється з інформацією p_i, яка записана у регістрі Рг. За результатами даного порівняння формується сигнал h_{pi}, який надходить на другі входи логічних елементів виключне АБО ПЕ p-го стовпчика та бере учать у формуванні $y_{q(i+1)}$ розрядів слова управління. Сигнал h_{pi} формується за таким виразом:

$$h_{pi} = \begin{cases} 0, & коли \quad Z_{pi} \geq p_i \\ 1, & коли \quad Z_{pi} < p_i \end{cases}. \qquad (4.5)$$

На виході віднімача Від отримуємо різницю L=p_i-Z_{pi}, яка у випадку коли h_{pi}=1 записується у регістр Рг елемент управління ЕУp. За n тактів роботи, де n –

розрядність чисел, на виходах D_{1i}^{*} - D_{Ni}^{*} отримаємо порозрядно масив відсортованих чисел.

Об'єднання ПЕ у стовпці забезпечує розпаралелення процесу сортування чисел, час виконання якого визначає тактом роботи пристрою. Такий такт обчислюється за наступним виразом:

$$T_{c1}=t_{Tг}+4t_I+ t_{БСм}+t_{СП} \tag{4.6}$$

де $t_{Tг}$ - час спрацювання тригера; t_I – час затримки логічних елементів типу *АБО, І, І-НЕ,* виключне *АБО;* $t_{БСм}$ – час спрацювання *N*- вхідного однорозрядного суматора; $t_{СП}$ – час порівняння двох чисел. Час паралельно-вертикального сортування одновимірного масиву чисел $\{D_k\}_{k=1}^{N}$ рівний:

$$t_{c1} = T_{c1}n. \tag{4.7}$$

Затрати обладнання на реалізацію НВІС-пристрою паралельно-вертикального сортування одновимірного масиву чисел рівні:

$$W_{c1} = N^2(2W_{Tг} + 4W_I) + N(2W_{Tг} + W_{Pг} + W_{Bід} + W_{Бсм} + W_{СП} + 2W_I) \tag{4.8}$$

де $W_{Tг}$, W_I, $W_{Pг}$, $W_{БСм}$, $W_{СП}$ та $W_{Bід}$ – затрати обладнання на реалізацію відповідно тригера, логічних елементів типу *І,* регістра, *N*- вхідного однорозрядного суматора, схеми порівняння та віднімача.

4.6.4. Реалізація НВІС-пристрою паралельно-вертикального сортування одновимірного масиву чисел

НВІС-пристрій паралельно-вертикального сортування одновимірного масиву чисел реалізуємо в інтегрованому середовищі Quartus II для ПЛІС FPGA EP3C16F484 сімейства Cyclone III фірми Altera на мові програмування апаратури VHDL з використанням бібліотек середовища розробки. Алгоритм паралельно-вертикального сортування одновимірного масиву чисел передбачає надходження в кожному такті роботи розрядного зрізу D_{ki} масиву із *N* чисел, починаючи зі старших розрядів. При реалізації НВІС-пристрою паралельно-вертикального сортування одновимірного масиву чисел необхідно забезпечити можливість працювати як з різною кількістю елементів масиву *N*, так їх розрядністю *n*.

Основними компонентами НВІС-пристрою паралельно-вертикального сортування одновимірного масиву чисел є: регістри, формувач вертикальних розрядних зрізів, одноразрядні багатовходові суматори, компаратори, перетворювачі вертикальних розрядних зрізів в послідовність даних розрядності *n*. Для синтезу НВІС-пристрою сортування було розроблено та відмодельовано роботу його основних компонентів.

Вигляд символу формувача вертикальних розрядних зрізів з розмірністю масиву *N*=16 і їх розрядністю *n*=8 (Vert_Form_16_8)

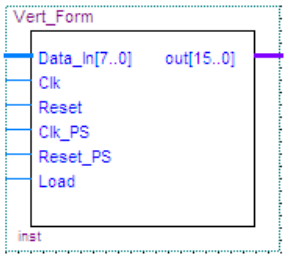
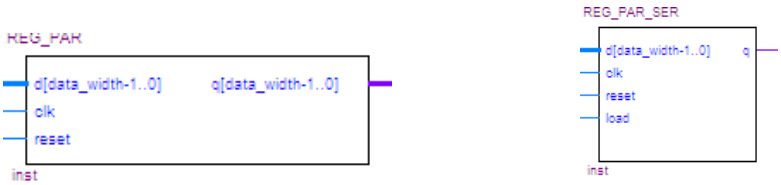


Рис.4.19 Зовнішній вигляд символу формувача вертикальних розрядних зрізів

Формувач вертикальних розрядних зрізів складається з *N* регістрів паралельно – паралельного типу та *N* регістрів паралельно – послідовного типу, символи яких зображені відповідно на рис. 4.20*a* і 4.20*b*.



a) *b*)

Рис.4.20 Зовнішній вигляд символів регістрів:

a) паралельно – паралельного типу; *b)* паралельно – послідовного типу

Входи формувача вертикальних паралельних зрізів: Data_In[7..0] – Dk, (*k*=1, …, *N*) розмірність одновимірного масиву; Clk – вхід синхронізації завантаження чисел масиву; Reset (активний рівень сигналу "0") – вхід початкового скидання в "0" виходу регістрів REG_PAR; Clk_PS – вхід синхронізації завантаження даних в регістр REG_PAR_SER; Reset_PS (активний рівень сигналу "1") – вхід початкового скидання

в "0" виходу регістрів REG_PAR_SER; Load (активний рівень сигналу "1") – сигнал дозволу завантаження даних в регістр REG_PAR_SER. Виходом формувача Vert_Form_16_8 є вертикальний паралельний зріз Out[15..0]. На протязі перших *N* імпульсів синхронізації Clk завантажуються дані в регістри REG_PAR. Під час кожного імпульсу синхронізації Clk_PS формується вертикальний паралельний зріз D_{ki} починаючи зі старшого розряду. Синхронізація реалізована по передньому фронту імпульсів Clk і Clk_PS.

На рис.4.21 наведені часові діаграми роботи формувача вертикальних розрядних зрізів Vert_Form_16_8, який формує вертикальні паралельні зрізи для 16-ти восьмирозрядних чисел (N=16; n=8): 0x2C, 0x35, 0xA0, 0x0F, 0xB7, 0x49, 0x64, 0x72, 0xF3, 0xE8, 0xA8, 0x83, 0x63, 0x2F, 0x7A, 0x2C. Розрядні зрізи для масиву цих чисел (зі старшого розряду): 0x0F14, 0x53E0, 0xF7D7, 0x4192, 0xE629, 0xA059, 0x7998, 0x393A.

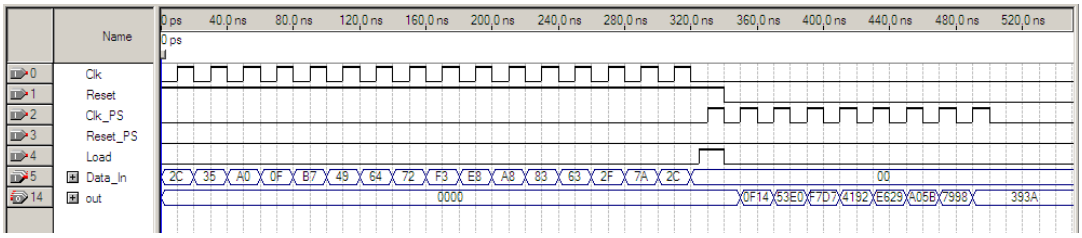


Рис.4.21 Часові діаграми роботи формувача вертикальних розрядних зрізів

Вигляд символу формувача послідовності вихідних даних розмірністю *N*=16 і розрядністю *n*=8 (Out_Data_16_8) наведено на рис. 4.22.

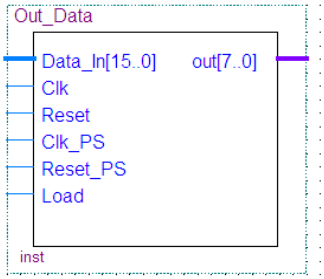


Рис.4.22. Зовнішній вигляд символу формувача послідовності вихідних даних

Структура формувача послідовності вихідних даних є подібною до структури формувача вертикальних паралельних зрізів. Формувач послідовності вихідних даних складається з *n* регістрів паралельно – паралельного типу та *n* регістрів паралельно – послідовного типу

Вхід Data_In[15..0] формувача послідовності вихідних даних – це D_{ki}, вертикальні паралельні зрізи, які поступають з модуля сортування вхідних даних, їх кількість (*n*) визначається розрядністю чисел одновимірного масиву. Входи Clk, Reset, Clk_PS, Reset_PS, Load мають аналогічне призначення і активні рівні сигналу як і в формувачі вертикальних паралельних зрізів. Виходом формувача Out_Data_16_8 (Out[7..0]) є послідовність *N* чисел розрядністю *n*. Для формувача Out_Data_16_8 на протязі перших n імпульсів синхронізації Clk завантажуються дані в регістри REG_PAR. Після цього під час кожного імпульсу синхронізації Clk_PS на виходах регістрів REG_PAR_SER формуються вихідні дані D*k. Кількість таких імпульсів синхронізації - N.

На рис.4.23 наведені часові діаграми роботи формувача послідовності вихідних даних Out_Data_16_8, який формує послідовність 16-ти 8-ми розрядних чисел для 8-ми вхідних вертикальних паралельних зрізів (N=16; n=8): 0x003F, 0x07C3, 0x7BDF, 0x08C5, 0xF44A, 0xF904, 0x92E5, 0x9E25. Послідовність даних на виході формувача: 0xF3, 0xE8, 0xB7, 0xA8, 0xA0, 0x83, 0x7A, 0x72, 0x64, 0x63, 0x49, 0x35, 0x2F, 0x2C, 0x2C, 0x0F.

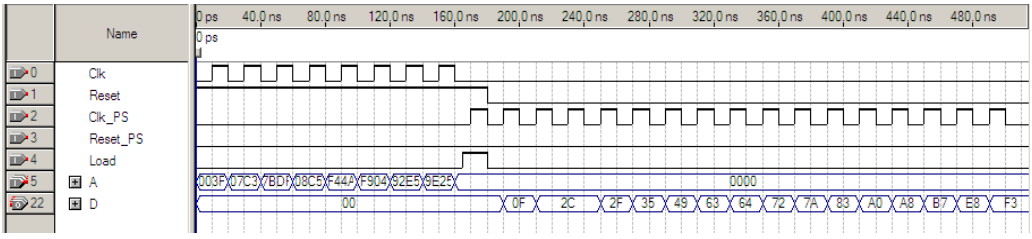


Рис.4.23 Часові діаграми роботи формувача послідовності вихідних даних

4.6.5. Сортування великорозмірних одновимірних масивів чисел з використанням розробленого НВІС-пристрою

Розглянемо сортування одновимірного масиву із M чисел, де $M=N\times b$, з використанням розробленого НВІС-пристрою паралельно-вертикального сортування одновимірного масиву із N чисел. Для виконання такого сортування пропонується використати метод сортування злиттям. Я вже було сказано, в основі алгоритмів сортування методом злиття лежить макрооперація об'єднання двох упорядкованих підмасивів в один впорядкований масив.

На початку сортування вхідний масив із M чисел розбивається на $b=M/N$ масивів довжиною N чисел, які сортуються за допомогою розробленого пристрою. Над відсортованими масивами із N чисел виконується макрооперація першого типу (об'єднання двох упорядкованих масивів із N чисел у один упорядкований масив із $2N$ чисел). У результаті виконання макрооперацій першого типу формуються $b/2$ впорядкованих масивів довжиною $2N$. Кількість типів макрооперації для сортування масиву з масиву з M чисел з використанням базової операції сортування N чисел визначається за формулою:

$$K = \left\lfloor \log_2 b \right\rfloor . \qquad (4.9)$$

Макрооперації першого типу реалізуються на трьох НВІС-пристроях паралельно-вертикального сортування N чисел, які об'єднуються у блок сортування першого типу за схемою наведеною на рис.4.24, де $БС_1$ – блок сортування першого типу; $D_{1i},\ldots, D_{Ni}$ і $D_{(N+1)i},\ldots, D_{2N}$ – входи, на які поступають два впорядковані масиви розміром N чисел кожний; $D^*_{1i},\ldots, D^*_{2Ni}$ – виходи відсортованого масиву із $2N$ чисел.

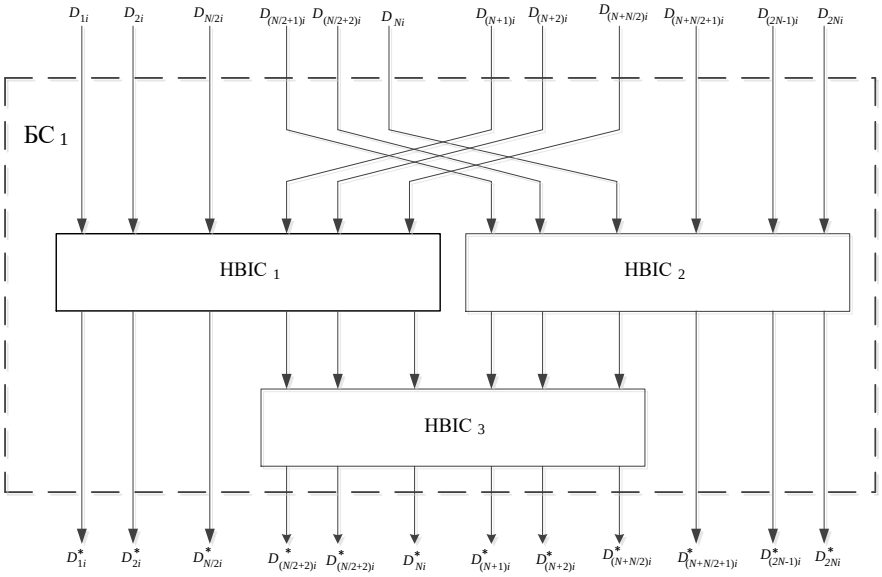


Рис.4.24. Схема блоку сортування першого типу

З входів $D_{1i},\ldots, D_{N/2i}$ на НВІС$_1$ надходять $N/2$ більших чисел першого масиву, а з входів $D_{(N+1)i},\ldots, D_{(N+N/2)i}$ та $N/2$ більших чисел другого масиву. На входи НВІС$_2$ з входів $D_{(N/2+1)i},\ldots, D_{Ni}$ надходять $N/2$ менших чисел першого масиву, а з входів $D_{(N+N/2+1)i},\ldots, D_{2Ni}$ та $N/2$ менших чисел другого масиву. На $1,\ldots,N/2$ виходах ППВС$_1$ отримуємо $N/2$ більших чисел відсортованого масиву із $2N$ чисел, а менші $N/2$ чисел з виходів $N/2+1,\ldots,N$ даного пристрою надходять на ППВС$_3$. На виходах $(N/2+1),\ldots,N$ другого ППВС$_2$ отримуємо $N/2$ менших чисел відсортованого масиву із $2N$ чисел. Відсортовані з допомогою НВІС$_3$ числа надходять на виходи $D^*_{(N/2+1)i},\ldots,D^*_{(N+N/2+1)i}$ блока БС$_1$.

Макрооперація s-го типу (об'єднання двох упорядкованих масивів із $N2^{s-1}$ чисел у один упорядкований масив із $N2^s$ чисел), де $s=1,\ldots,K$, реалізуються блоком сортування БС$_s$, який отримується шляхом об'єднання трьох блоків БС$_{s-1}$. Схема для сортування масиву із $8N$ чисел методом злиття з використанням НВІС-пристроїв паралельно-вертикального сортування N чисел наведена на рис.4.25 де БС$_2$ і БС$_3$ – блоки сортування відповідно другого та третього типів.

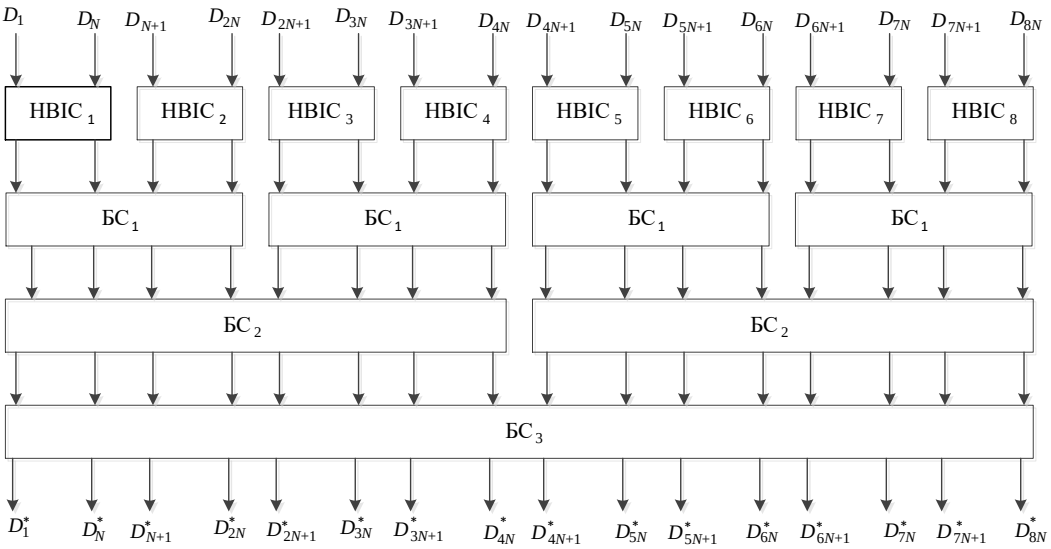


Рис.4.25. Схема сортування масиву із 8*N* чисел методом злиття

Для сортування масиву із 8*N* чисел використовуються три типи блоків сортування, які реалізуються на базі НВІС-пристроїв паралельно-вертикального сортування масивів із *N* чисел. Кількість НВІС, яка необхідна для сортування масиву із 8*N* чисел рівна 65. Сортування масиву із 8*N* чисел виконується за конвеєрним принципом з тактом $T_{c1}=t_{Tг}+4t_I+ t_{БСм}+t_{СП}$. Час сортування масиву із 8*N* чисел визначається за формулою:

$$t_c = T_{c1}(n+6)\,. \tag{4.10}$$

Паралельно-вертикальне сортування масивів чисел великої розмірності з використанням розробленої НВІС забезпечує зменшення часу сортування за рахунок конвеєрної організації сортування та формування в кожному такті відсортованих розрядних зрів.

4.6.6. Сортування двовимірних масивів чисел з використанням розробленого НВІС-пристрою паралельно-вертикального сортування одновимірного масиву чисел

Сортування двовимірного масиву чисел $\{D_{hj}\}_{h=1;\,j=1}^{N/2;M}$ з використанням НВІС-пристрою паралельно-вертикального сортування одновимірних масивів чисел передбачає, що вхідні дані надходять паралельно *N/2* каналами. Таке сортування доцільно виконувати методом витісненням, базовою макрооперацією якого є

паралельно-вертикальне сортування одновимірного масиву із *N* чисел. В кожному макротакті реалізації даного сортування виконується сортування *N* чисел, з яких *N/2* є вхідними числами, а *N/2*- більшими числами з попереднього макротакту роботи. Відсортовані *N/2* більших чисел залишаються для виконання наступної макрооперації з новими вхідними даними, а *N/2* менших записуються у пам'ять. Кількість макротактів, необхідних для сортування двовимірного масиву чисел $\{D_{hj}\}_{h=1;\,j=1}^{N/2;M}$ на базі одного НВІС-пристрою паралельно-вертикального сортування одновимірних масивів із *N* чисел, визначається за формулою:

$$g = \sum_{j=1}^{M}(M - j)\,. \tag{4.11}$$

Одним із шляхів підвищення швидкодії сортування двовимірного масиву чисел є збільшення кількості базових макрооперацій, які виконуються паралельно шляхом розроблення паралельно-потокових структур. Потоковий граф алгоритму паралельно-потокового сортування двовимірного масиву чисел $\{D_{hj}\}_{h=1;\,j=1}^{N/2;M}$ методом витіснення, наведений на рис.4.26, де У –вхід управління; 1-*N/2* – входи даних; $ПЕ_k$ - *j*-й процесорний елемент; $Ф_{Уj1}$, $Ф_{Уj2}$ – перший та другий оператори управління $ПЕ_j$; $Ф_{Кj1}$, $Ф_{Кj2}$ – перший та другий оператори комутації $ПЕ_j$; $Ф_{Пj1}$, $Ф_{Пj2}$ – перший та другий оператори запам’ятовування $ПЕ_j$; $Ф_С$ – оператора сортування *N* чисел.

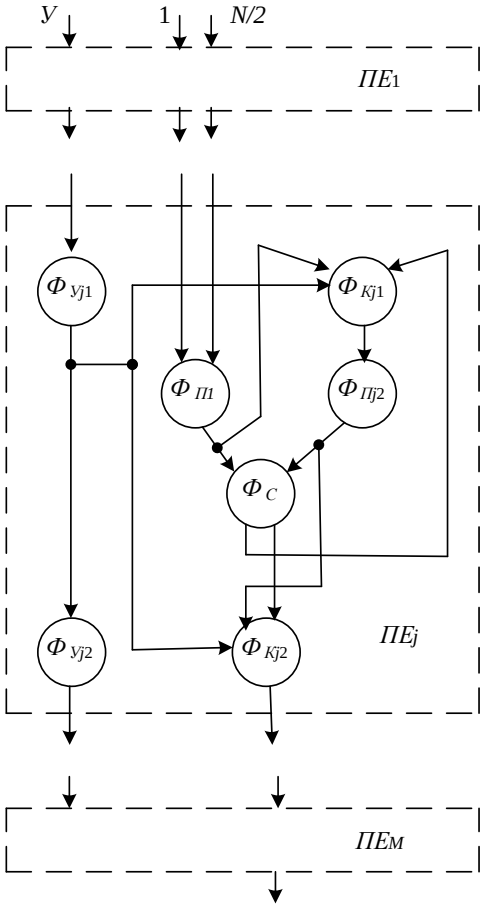


Рис.4.26. Потоковий граф алгоритму паралельно-потокового сортування двовимірного масиву чисел методом витіснення

При паралельно-потоковому сортуванні двовимірного масиву чисел $\{D_{hj}\}_{h=1;\,j=1}^{N/2;M}$ методом витіснення використовуються M ПЕ. Кожний $ПЕ_j$ реалізується на базі першого $Ф_{Уj1}$та другого $Ф_{Уj2}$ операторів управління, першого $Ф_{Кj1}$ та другого $Ф_{Кj2}$ операторів комутації, першого $Ф_{Пj1}$ та другого $Ф_{Пj2}$ операторів запам’ятовування та оператора сортування N чисел $Ф_С$. Особливістю паралельно-потокового сортування є використання НВІС-пристрою паралельно-вертикального сортування одновимірних масивів чисел для реалізації оператора сортування $Ф_С$. Даний оператор виконується за n тактів, тобто один макротакт. Розроблений потоковий граф алгоритму паралельно-потокового сортування двовимірного масиву чисел методом витіснення орієнтований на сортування неперервних потоків даних у реальному часі.

Структура паралельно-потокового пристрою сортування двовимірного масиву чисел методом витіснення наведена на рис.4.27, де *ТІ* – вхід тактових імпульсів; *У* –

вхід управління; $Вх_1 – Вх_{N/2}$ входи даних; *ПЕ* – процесорний елемент; *Тг* – тригер; *П* - пам’ять; *Км* – комутатор; НВІС-Сорт – НВІС сортування *N* чисел; $Вих_1 – Вих_{N/2}$ – виходи відсортованих даних.

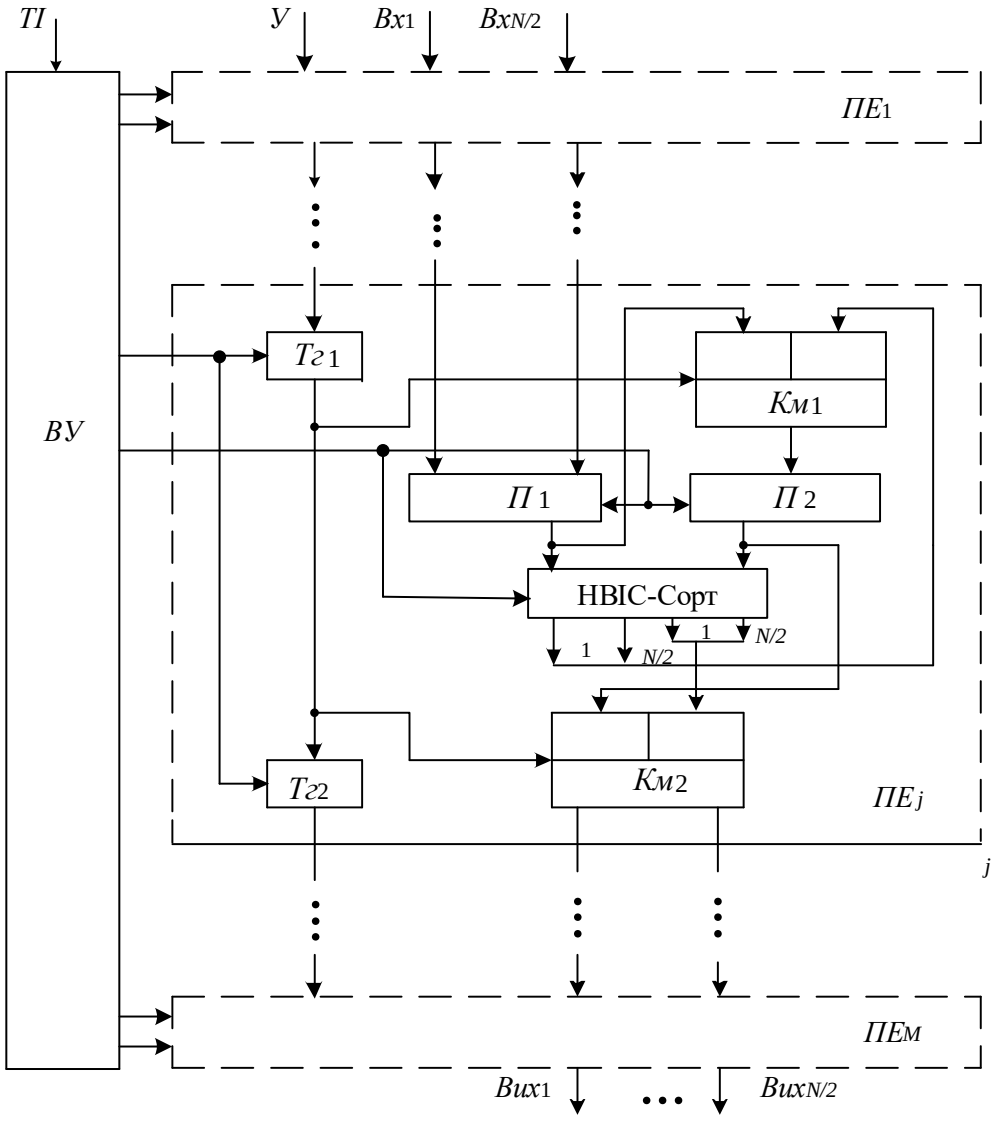


Рис.4.27. Структура паралельно-потокового пристрою сортування двовимірного масиву чисел методом витіснення

Робота паралельно-потокового пристрою сортування двовимірного масиву чисел методом витіснення починається з того, що на вході *У* встановлюється сигнал лог.1, який сигналізує про початок надходження двовимірного масиву. У кожному такті роботи розрядні зрізи вхідних даних з входів $Вх_1 – Вх_{N/2}$ записуються в пам'ять $П_1$. Після *n* тактів у пам'яті $П_1$ отримуємо перший одновимірний масив $\{D_{h1}\}_{h=1}^{N/2}$. Після

цього першим макротактовим імпульсом у тригер Tr1 першого $ПЕ_1$ записується лог.1, яка встановлює $Км_1$ і $Км_2$ на передачу інформації з перших входів. У наступному макротакті у $П_1$ записується другий одновимірний масив $\{D_{h2}\}_{h=1}^{N/2}$, а у $П_2$ переписується перший $\{D_{h1}\}_{h=1}^{N/2}$. Наступним макротактовим імпульсом у $Тг_1$ записується лог.0, а в $Тг_2$ переписується лог.1. У кожному наступному такті дані з виходів $П_1$ і $П_2$ надходять розрядними зрізами на НВІС-Сорт. На виходах НВІС-Сорт отримуємо розрядні зрізи відсортованого масиву із N чисел. Більші $N/2$ чисел через $Км_1$ надходять на входи пам'яті $П_2$, а $N/2$ менших через $Км_2$ на виходи наступного $ПЕ_2$. У наступних макротактах паралельно-потоковий пристрій сортування працює аналогічно.

Відразу після завантаженням M-го одновимірного масиву чисел $\{D_{hM}\}_{h=1}^{N/2}$ в $П_1$ першого $ПЕ_1$ може виконуватися завантаження та сортування наступного двовимірного масиву. Одночасне сортування двох двовимірних масивів забезпечує зменшення часу сортування та високу ефективність використання обладнання. Для виключення можливості змішування чисел з різних двовимірних масивів використовується лог.1 в тригерах $Тг_1$, $Тг_2$, яка просуваються через всі $ПЕ$. Завершення сортування чергового двовимірного масиву даних фіксується наявністю одиночного сигналу на виході Гот.

Сортування двовимірного масиву $\{D_{hj}\}_{h=1;\,j=1}^{N/2;M}$ у паралельно-потоковому пристрої сортування виконується за час:

$$t_c = 2Mn(t_П + t_C + t_{Км}), \tag{4.12}$$

де M – кількість одновимірних масивів; n – розрядність даних; $t_П$ - час звертання до пам’яті; t_C - такт роботи вертикально паралельного пристрою сортування; $t_{Км}$ – час затримки даних на комутаторі.

4.6.7. Реалізація на FPGA пристрою паралельно-вертикального пошуку максимального і мінімального чисел у масивах

Розроблено структуру апаратного засобу для одночасного паралельно-вертикального пошуку максимального та мінімального чисел як у одновимірному

$\{D_k\}_{k=1}^{N}$, так і двовимірному $\{D_k\}_{k=1;j=1}^{N;M}$ масивах даних, яку наведено на рис. 4.28, де *ТІ* – тактові імпульси; *ПУ* – початкова установка; Тг – тригер; D_{kji} – *k*-й вхід *j*-го одновимірного масиву *i*-го розряду; Рг – зсувний регістр; D_{max} – вихід максимального числа; D_{min} – вихід мінімального числа.

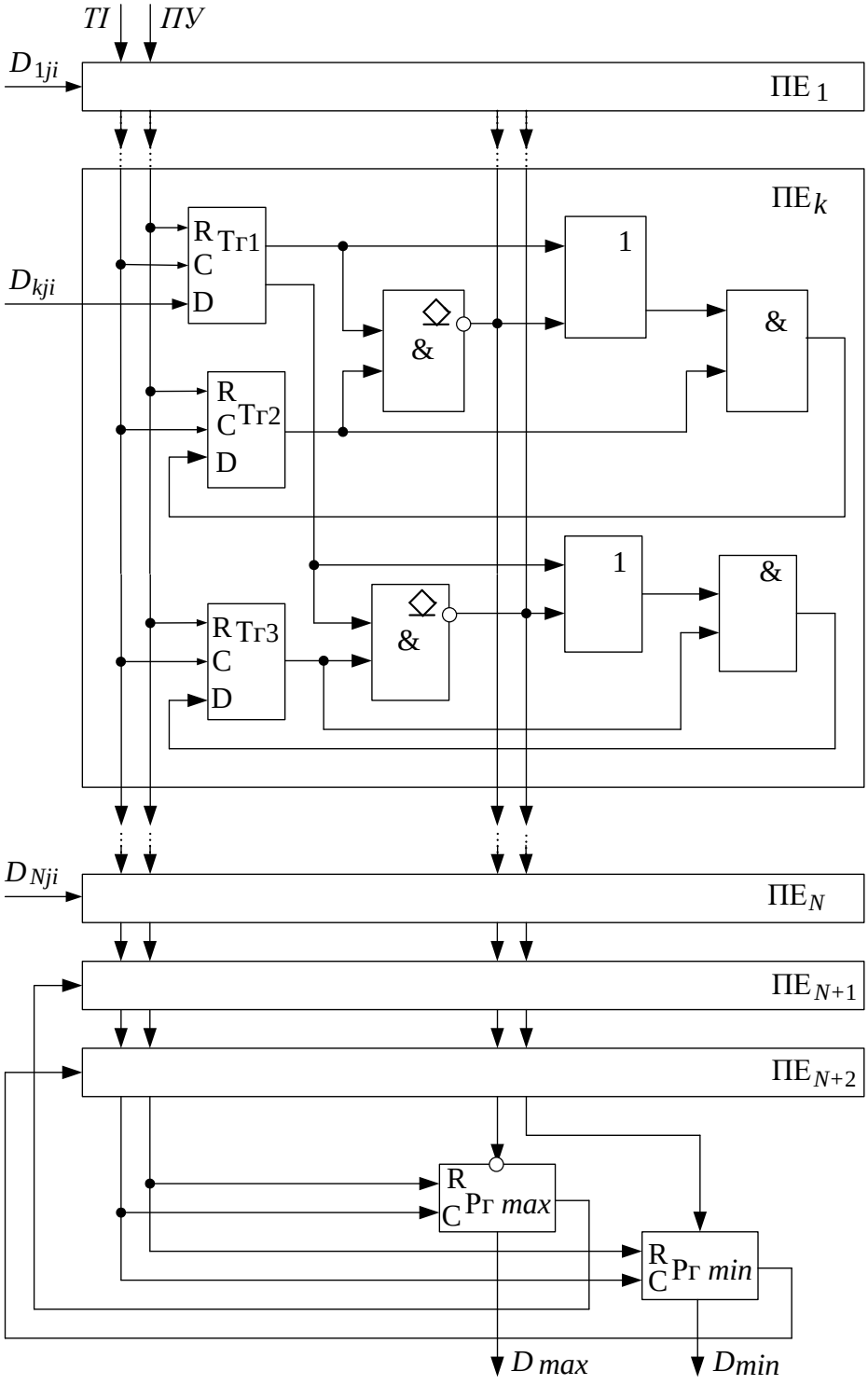


Рис.4.28 – Структура апаратного засобу для одночасного паралельно-вертикального пошуку максимального та мінімального чисел

Кількість ПЕ, що підключені до спільної шини результатів, при одночасному обчисленні максимального і мінімального чисел для одновимірного масиву $\{D_k\}_{k=1}^{N}$ визначається розміром масиву. Використання спільної шини результатів забезпечує розпаралелення процесу опрацювання розрядного зрізу, час опрацювання якого визначає такт роботи пристрою.

Реалізовано на FPGA компоненту пошуку максимального числа, яка забезпечує роботу з різною кількістю елементів масиву та розрядністю чисел 8 і 16 біт. Вигляд символу компоненти пошуку максимального числа з розмірністю масиву N=64 і їх розрядністю n=16 наведений на рис. 4.29.

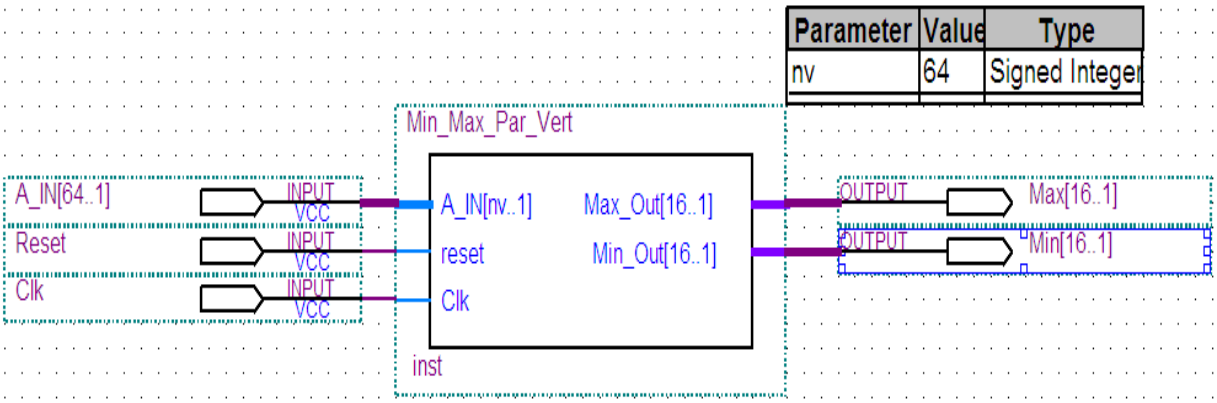


Рис. 4.29 – Вигляд символу компоненти пошуку максимального(мінімального) числа

Його відмінність від розглянутих вище ПЕ полягає в тому, що символ об’єднує в собі N таких ПЕ, тобто пошук максимального чисела виконується паралельно для i-го розрядного зрізу A_i розмірністю N. Він складається з входу синхронізації Clk, входу Reset для початкового встановлення паралельно-вертикального пристрою, входів A_IN[64..1] розрядних зрізів A_i, що формуються з 64-х 16-ти розрядних чисел одномірного масиву та виходів максимального Max_Out[16..1] числа. Перед початком роботи імпульсом Reset=1 встановлюються в "1" початкові значення слів управління Y_k і Z_k, розмірність яких співпадає з розмірністю A_i. Пошук максимального числа виконується за n тактів, під час кожного з яких відбувається паралельно зчитування в пристрій розрядних зрізів A_i та їх опрацювання. Синхронізація реалізована по передньому фронту імпульсів Clk.

На рис. 4.30 наведені часові діаграми роботи компоненти пошуку максимального числа серед чотирьох восьмирозрядних чисел (N=4; n=8): 0xE5, 0xDC, 0x1B, 0x9D. Результатом такого пошуку є число 0xE5. Розрядні зрізи для цих чисел: 0xD, 0xC, 0x8, 0x7, 0x7, 0xD, 0x2, 0xB. Часова затримка визначення i-го розряду максимального числа для кожного розрядного зрізу вхідного масиву чисел складає приблизно 7 нс. Результат отримано за 8-м тактових імпульсів і мінімальний час його знаходження складає приблизно 65 нс.

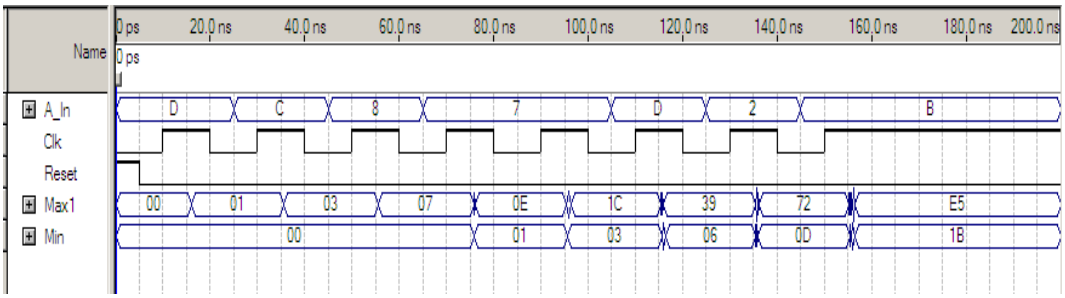


Рис. 4.30 – Часові діаграми роботи компоненти
пошуку максимального числа

Проведено порівняння апаратних ресурсів FPGA, затрачених на реалізацію пристрою паралельно-вертикального обчислення максимального і мінімального чисел одновимірного масиву для розрядних зрізів різної розмірності і розрядності чисел масиву n=8, 16. Результати дослідження відображені на рис. 4.31. Апаратними ресурсами є кількість логічних елементів (LE) та виводів FPGA.

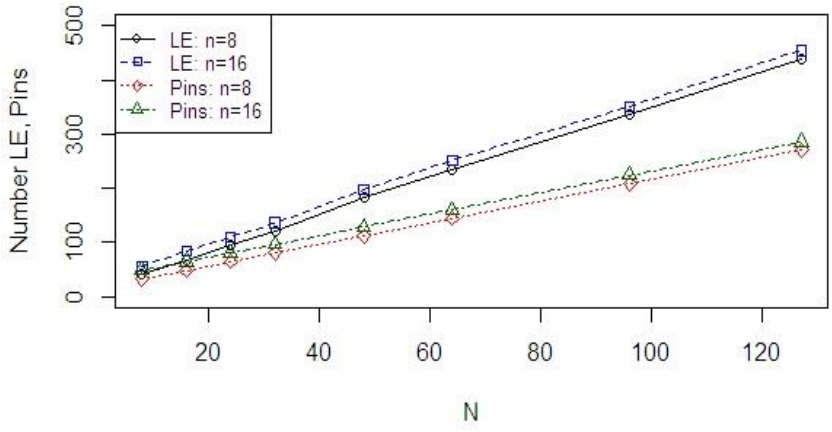


Рисунок 4.31 – Затрати апаратних ресурсів FPGA для реалізації компоненти
паралельно-вертикального пошуку максимального числа

З рисунку бачимо, що кількість необхідних апаратних ресурсів FPGA зростає лінійно зі збільшенням числа елементів в масиві. Так для *N*=127 чисел в масиві кількість LE і виводів FPGA, необхідних для реалізації цього пристрою складає відповідно 438 і 145 для *n*=8 та 454 і 161 – для *n*=16. Кількість необхідних LE мало залежить від розрядності чисел масиву. Реалізовані паралельно-вертикальні пристрої Max_Min_Y_X для розрядності чисел *n*=8 і *n*=16 відрізняються тільки на 16 LE для різних розмірів масиву.

Для обчислення максимального чисела у двовимірному масиві методом паралельно-вертикального пошуку розроблена компонента Max_NxM_n, де *N* – кількість чисел в рядках масиву; *M* – кількість рядків двовимірного масиву; *n* – розрядність чисел. Зовнішній вигляд символу компоненти пошуку максимального числа Max_4x2_8 наведена на рис. 4.32.

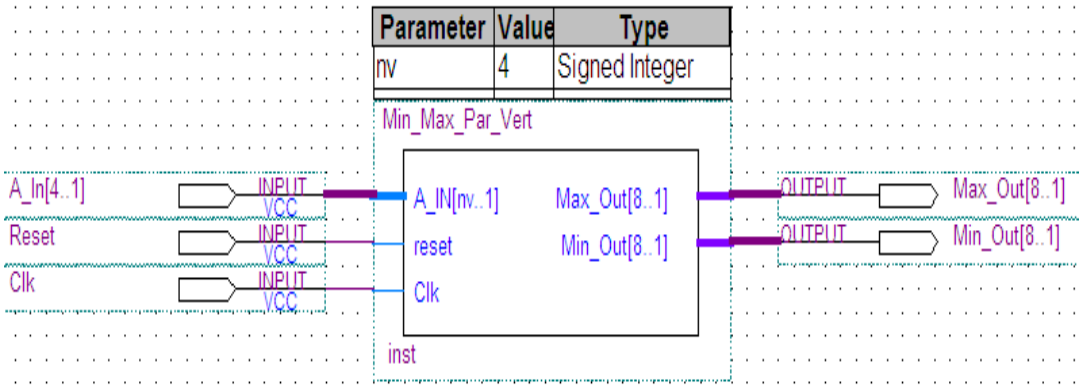


Рисунок 4.32 – Зовнішній вигляд символу Max_4x2_8

Розроблена компонента для пошуку максимального числа у двовимірному масиві працює аналогічно компоненті пошуку максимального числа серед елементів одновимірного масиву. Тільки в компоненту додатково вводиться зсувний регістр розрядністю *n*. При обчисленні максимального числа серед елементів рядка до розрядного зрізу розмірності *N* додається ще один розряд, що отримується з максимального числа попередніх рядків. Робота синхронізується вхідними імпульсами Clk. Імпульсом Reset=1 запускається обчислення максимального числа для кожного рядка двовимірного масиву.

Як приклад розглянуто роботу компоненти паралельно-вертикального пошуку максимального числа серед елементів двовимірного масиву D(4x2). Перший рядок матриці – числа 0xE5, 0xDC, 0x1B, 0x9D. Другий рядок – числа 0x94, 0x3C, 0x7D, 0xA6. На рис. 4.33 наведені часові діаграми роботи компоненти паралельно-вертикального пошуку максимального числа.

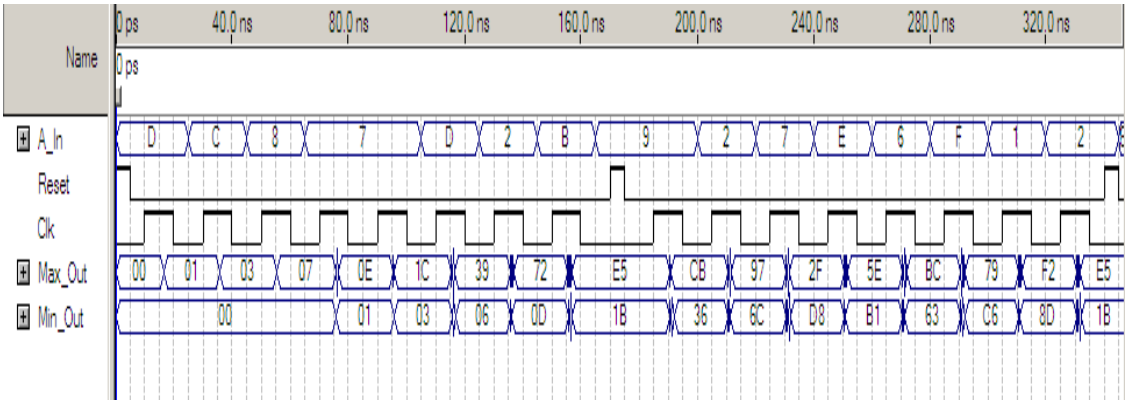


Рисунок 4.33 – Часові діаграми роботи компоненти паралельно-вертикального пошуку максимального числа

Серед них максимальним числом є, а мінімальним - 0x1B. Розрядні зрізи формуються так: для першого рядка двовимірного масиву 0xD, 0xC, 0x8, 0x7, 0x7, 0xD, 0x2, 0xB; для другого рядка: - 0x9, 0x2, 0x7, 0xE, 0x6, 0xF, 0x1, 0x2. Після $M \times n$ тактів (2×8) на виході Max_Out отримаємо 0xE5, яке є максимальним числом.

4.7. Висновки до розділу 4

1. Розроблено інформаційну технологію паралельного сортування даних, яка ґрунтується на інтегрованому підході, який охопив: дослідження, розроблення методів і алгоритмів паралельного сортування масивів даних; розроблення функціональних моделей алгоритмів паралельного сортування потоків даних у реальному часі; сучасну елементну базу (GPU, ПЛІС) та засоби автоматизованого проектування; нові архітектурні рішення, орієнтовані на технології надвеликих інтегральних схем. Розроблена інформаційна технологія паралельного сортування даних ґрунтується на розроблених і вдосконалених методах сортування даних, функціональних моделях і враховує інтенсивність надходження даних, розміри

масивів даних, особливості засобів реалізації та забезпечує сортування даних у реальному часі з високою ефективністю використання обладнання.

2. Розроблено інформаційну технологію паралельного пошуку даних, основними компонентами якої є: засоби збирання та передавання даних розрядними зрізами; розроблені методи паралельно-вертикального пошуку максимальних і мінімальних чисел; функціональні моделі пошуку у реальному часі максимальних і мінімальних чисел у масивах; засоби автоматизованого проєктування апаратного і програмного забезпечення; програмні та апаратні засоби пошуку даних у реальному часі з високою ефективністю використання обладнання.

3. Розроблено програмний засіб паралельного сортування масивів даних, який завдяки використанню CUDA та вдосконаленого методу злиття зменшує час сортування на 31% порівняно з існуючими програмними засобами сортування злиттям.

4. На основі CUDA розроблено програмний засіб для паралельно-вертикального сортування даних методом підрахунку одиниць у розрядному зрізі. Порівняно час сортування масивів даних цим програмним засобом з часом сортування програмою, яку виконують тільки на CPU. Показано, що паралельно-вертикальне сортування масивів даних на основі підрахунку одиниць у розрядному зрізі, яке реалізується на GPU з використанням програмної моделі CUDA, виконується за час у 4 рази менший, ніж час його реалізації на CPU.

5. Розроблено програмні засоби для реалізації паралельно-вертикального пошуку максимальних і мінімальних значень із застосуванням локальних бар’єрів та із застосуванням ітеративного підходу з комбінуванням CPU та GPU. Показано підхід із застосуванням локальних бар’єрів, ефективний для невеликих масивів даних, які ми можемо повністю завантажити у пам’ять GPU і тим самим забезпечити ефективну обробку без постійного обміну з пам’яттю CPU. Підхід з комбінуванням CPU та GPU є ефективним для пошуку максимальних і мінімальних значень у великих масивах даних та вимагає постійного обміну даних між CPU та GPU.

6. Вибрано для реалізації апаратних засобів інформаційних технологій паралельного сортування та пошуку ПЛІС фірми Altera та автоматизовану систему проектування Quartus II. Показано, що перевагами технології проєктування пристроїв на основі ПЛІС є такі: мінімальний час розроблення схеми реалізації операцій сортування та пошуку даних (необхідно тільки занести в пам’ять ПЛІС конфігураційний код); на відміну від звичайних елементів цифрової схемотехніки не потрібно розробляти і виготовляти складні друковані плати; швидке перетворення однієї конфігурації схеми на іншу (заміна коду конфігурації схеми в пам’яті); апаратні засоби на основі ПЛІС створюють записом конфігурації у пам’ять.

7. Розроблено структуру пристрою для паралельно-вертикального сортування одновимірного масиву із N чисел методом підрахунку кількості одиниць у розрядному зрізі та реалізовано його на ПЛІС FPGA EP3C16F484 сімейства Cyclone III фірми Altera.

8. Розроблено структури для паралельно-вертикального сортування одновимірного масиву довжиною $M=N\times b$ на базі реалізованого на ПЛІС пристрою паралельно-вертикального сортування одновимірного масиву із N чисел.

9. Розроблено структуру апаратного засобу для одночасного паралельно-вертикального пошуку максимального та мінімального чисел як у одновимірному $\{D_k\}_{k=1}^{N}$, так і двовимірному $\{D_k\}_{k=1;\,j=1}^{N;M}$ масивах даних.

ВИСНОВКИ

У дисертаційній роботі на основі проведених теоретичних та експериментальних досліджень вирішено актуальне наукове завдання: розроблено нові та вдосконалено існуючі методи, моделі та програмно-апаратні засоби інформаційних технологій паралельного сортування та пошуку даних у реальному часі з високою ефективністю використання обладнання. При цьому отримано такі основні результати:

1. Проаналізовано інформаційні технології сортування та пошуку даних, визначено основні напрями розроблення методів, моделей та засобів ефективного сортування та пошуку даних, що дало змогу сформулювати задачі дисертаційного дослідження.

2. Розроблено метод паралельно-вертикального пошуку максимальних і мінімальних чисел у масивах, який завдяки паралельному опрацюванню *i*-го розрядного зрізу масиву чисел і паралельному формуванню слів управління зменшує час пошуку, що визначається в основному розрядністю чисел.

3. Удосконалено метод паралельного сортування даних злиттям, який внаслідок використання базової операції об'єднання двох масивів із одночасним формуванням елементів зростаючого і спадаючого масивів забезпечує зменшення часу сортування даних приблизно удвічі.

4. Удосконалено метод паралельно-вертикального сортування даних, який завдяки підрахунку одиниць у *i*-му вхідному розрядному зрізі та паралельному формуванню *i*-го розрядного зрізу відсортованого масиву чисел зменшує час сортування на 17%.

5. Розроблено інформаційну технологію паралельного сортування даних, яка завдяки використанню розроблених і вдосконалених методів, функціональних моделей паралельно-потокового сортування даних та врахуванню інтенсивності надходження даних, розмірів масивів даних і вибору засобів реалізації забезпечує сортування даних у реальному часі з високою ефективністю використання обладнання.

6. Розроблено інформаційну технологію паралельного пошуку даних, яка завдяки використанню розробленого методу паралельно-вертикального пошуку максимальних і мінімальних чисел, функціональних моделей та врахуванню інтенсивності надходження даних і вибору засобів реалізації забезпечує пошук даних у реальному часі з високою ефективністю використання обладнання.

7. Розроблено засоби сортування масивів даних на базі графічного процесора з використанням вдосконаленого методу паралельного сортування даних злиттям, які зменшують час сортування на 31%.

8. Розроблено на ПЛІС FPGA EP3C16F484 сімейства Cyclone III фірми Altera апаратний засіб для паралельно-вертикального пошуку максимальних і мінімальних значень, який працює з тактом, що дорівнює часу спрацювання тригера та трьох логічних елементів типу “І”.

9. Результати дисертаційної роботи використано для розроблення засобів збирання та попереднього опрацювання телеметричних даних на ДП “Львівський державний завод “ЛОРТА” та впроваджено у навчальний процес кафедри автоматизованих систем управління Національного університету “Львівська політехніка”.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Цмоць І. Г., Антонів В. Я. Вертикально-паралельний метод сортування масивів чисел // Збірник наукових праць “Моделювання та інформаційні технології”. Інститут проблем моделювання в енергетиці ім. Г. Є. Пухова. 2016, Випуск 77. С. 186 – 192.

2. Цмоць І. Г., Антонів В. Я., Рабик В. Г. Метод вертикально-паралельного обчислення максимальних і мінімальних чисел у масивах // Збірник наукових праць “Моделювання та інформаційні технології”. Інститут проблем моделювання в енергетиці ім. Г. Є. Пухова. 2016, Випуск 76. С. 190 – 196.

3. Цмоць І. Г., Антонів В. Я. Удосконалення паралельного сортування масивів чисел методом злиття // Науковий вісник НЛТУ України : збірник наукових праць. Львів, 2020, Т. 30, № 4. С. 134 – 142.

4. Ivan Tsmots, Oleksa Skorokhoda, Volodymyr Antoniv. Parallel algorithms and structures for implementation of merge sort // International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 5 Issue 3, March 2016. Pp. 798 – 807.

5. Цмоць І. Г., Скорохода О. В., Красовський В. Б., Антонів В. Я. Методи та НВІС-структури узгоджено-паралельного обчислення максимальних і мінімальних значень // Збірник наукових праць інституту проблем моделювання в енергетиці ім. Г. Є. Пухова. 2014, Випуск 70. С.38 – 48.

6. Цмоць І. Г., Парубчак В. О., Антонів В. Я. Паралельно-вертикальне сортуванням одновимірних масивів даних методом злиття з використанням підрахунку // Збірник наукових праць інституту проблем моделювання в енергетиці ім. Г. Є. Пухова. 2013, Випуск 68. С.92 – 100.

7. Цмоць І. Г., Антонів В. Я. Апаратні засоби сортування даних методом злиття в реальному часі // Вісник національного університету “Львівська політехніка” Збірник наукових праць, 2015, № 814. - С. 171 – 186.

8. Цмоць І. Г., Кісь Я. П., Антонів В. Я. Застосування графічного процесора для підвищення швидкодії процесу сортування великих масивів даних // Науковий вісник

НЛТУ України: Збірник науково-технічних праць. – Львів : РВВ НЛТУ України. – 2015. – Вип. 25.6. – С. 328 – 334.

9. Ivan Tsmots, Oleksa Skorokhoda, Vasyl Rabyk, Volodymyr Antoniv Vertically-Parallel Method and VLSI-Structure sorting of Arrays of Numbers // Advances in Intelligent Systems and Computing III. Springer International Publishing AG 2019. Pp.267 – 284.

10. Цмоць І. Г., Антонів В. Я. Алгоритми та паралельні структури сортування даних методом вставки // Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів : РВВ НЛТУ України. – 2016. – Вип. 26.1. – С. 340 – 350.

11. Ivan Tsmots, Oleksandr Kuzmin, Vasyl Dubuk, Volodymyr Antoniv Improvement and orientation of method of data arrays sorting by confluence on architecture of graphic processor unit // Advances in Intelligent Systems and Computing V. International Conference on Computer Science and Information Technologies, CSIT 2020, September 23-26, 2020, Zbarazh, Ukraine. Pp. 276 – 286.

12. Цмоць І. Г., Антонів В. Я. Метод розробки апаратних засобів паралельно-узгодженого сортування масивів даних у реальному часі // Матеріали міжнародної конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту» ISDMCI 2015, 25–28 травня. — Залізний Порт, 2015. – С.221 – 222.

13. Парубчак В. О., Антонів В. Я. Методи паралельного сортування одновимірних масивів даних // Матеріали міжнародної конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту» ISDMCI 2014, 28-31 травня. — Залізний Порт, 2014. – С.23 – 24.

14. Цмоць І. Г., Антонів В. Я. Застосування графічного процесора для підвищення швидкодії сортування великих масивів даних // Матеріали XIV Міжнародного наукового семінару «Сучасні проблеми інформатики в управлінні, економіці, освіті» , Світязь. – 2015. – С.90 – 92.

15. Цмоць І. Г., Антонів В. Я. Методи та апаратно-програмні засоби паралельно-вертикального сортування масивів чисел // Збірник наукових праць міжнародної наукової конференції “Інтелектуальні системи прийняття рішень та

проблеми обчислювального інтелекту ISDMCI’2016”. – Залізний Порт, 2016. – С. 226 – 227.

16. Tsmots I., Rabyk V., Skorokhoda O., Antoniv V. FPGA implementation of vertically parallel minimum and maximum values determination in array of numbers // 2017 14th International Conference The Experience of Desingning and Application of CAD Systems in Microelectronics (CADSM). PROCEEDINGS. Polyana. February 21-25, 2017. Pp. 234 – 236.

17. Tsmots I., Skorokhoda O., Antoniv V., Rabyk V. Vertically-Parallel Method and VLSI-Structure for Sorting of One-Dimensional Arrays // Computer Sciences and Information Technologies. In Proceedings of 13th International Scientific and Technical Conference CSIT 2018. Pp. 112 – 116.

18. Цмоць І. Г., Скорохода О. В., Медиковський М. О., Антонів В. Я. Пристрій для визначення максимального числа з групи чисел. Патент України на винахід №110187, 25.11.2015, Бюл. №22.

19. Knuth E. D., The art of computer programmin. Sorting and searching. (Second edition) // Addison-Wesley Professional, Volume 3, 1998. – 792p.

20. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C., Introduction to Algorithms (3rd ed.) // MIT Press and McGraw-Hill, 2009. – 1032p.

21. Korf R. E., Artificial Intelligence Search Algorithms // In Algorithms Theory Computation Handbook, CRC Press, 1998. – Pp. 1–20.

22. Левитин А. В., Алгоритмы: введение в разработку и анализ // Москва: Вильямс, 2006. – 576с.

23. Edelkamp S., Schrödl S., Heuristic search: theory and applications // Morgan Kaufmann Publishers, 2012. – Pp. 465–466

24. Almasi G. S., Gottlieb A., Highly Parallel Computing // Redwood City: Benjamin-Cummings Publishing Co, 1989. – Pp. 237.

25. Cormen T. H., Algorithms Unlocked // MIT Press, 2009. – Pp. 25–59.

26. Skiena S. S, The Algorithm Design Manual // Springer, 2008. – Pp. 103–139.

27. Narasimha Karumanchi, Data Structures and Algorithms Made Easy: Data Structures and Algorithmic Puzzles Paperback // CareerMonk Publications, 2017. – Pp.505–507.

28. Brass P., Advanced Data Structures // Cambridge: Cambridge university press, 2008. – Pp. 50-51,148–150, 210, 374.

29. H. W. Lang, Sequential and Parallel Sorting Algorithms // FH Flensbur, 2000. [Електронний ресурс]. URL: https://www.inf.hs-flensburg.de/lang/algorithmen/sortieren/algoen.htm

30. Adve S. V., and all. Parallel Computing Research at Illinois: The UPCRC // University of Illinois at Urbana-Champaign, 2007. – Pp.16–19

31. Tsmots I., Medykovskyy M., Skorokhoda A., Teslyuk T. Design of Intelligent Component of Hierarchical Control System. // An International Quarterly Journal. Vol. 05, No. 2. 2016. – Pp. 3–10.

32. Якименко Ю. І., Терещенко Т. О., Сокол Є. І., Жуйков В. Я., Петергеря Ю. С., Мікропроцесорна техніка: Підручник. За ред. Т.О. Терещенко. К.: Видавництво "Політехнік", 2003. – С. 340.

33. Patterson D. A., Hennessy J. L., Computer Organization and Design. The Hardware/Software Interface // Morgan Kaufmann Publishers, 2020. – Pp. 549, 668–669, 713, 714.

34. Дейт К. Дж, Введение в системы баз данных Седьмое издание // Москва: Вильямс, 2006. – С. 31, 159, 398.

35. Райордан Р., Основы реляционных баз данных // Издательско-торговый дом «Русская Редакция», 2001. – С. 235.

36. Inmon W. H., Building the Data Warehouse, Third Edition // New York: John Wiley & Sons, Inc., 2002. – 428 p.

37. Шпак Н. О., Венгер О. І. Переваги використання інформаційно-комунікаційних технологій в Україні // Вісник Національного університету “Львів. Політехніка”. № 727, 2012. – С. 461–467.

38. Маслянко П. П., Системне проектування процесів інформатизації // Наукові вісті НТУУ “КПІ, 2008. – С.28–36.

39. Грушвицький Р., Мурсаев А., Цгрюмов Е, Проетирование систем на микросхемах програмируемой логики// Санкт – Петербург 2002. – С. 219, 148, 287–288.

40. Боресков А. В., Харламов А. А., Марковский Н. Д., Микушин Д. Н., Мортиков Е. В., Мыльцев А. А., Сахарных Н. А., Фролов В.А. Паралельные вычисления на GPU, Архитектрура и программная модель CUDA // Издательство Московского университета, 2012. –336 с.

41. Гергель В. П., Высокопроизводительные вычисления для мнгопроцесорных многоядерных систем // Издательство Московского университета, 2010. – С. 500-544.

42. Плахтеев А.П., Плахтеев П.А., Анализ 32-разрядных архитектур микроконтроллеров встраиваемых и мобильных систем. Радіоелектронні і комп'ютерні системи. 2009. 6. С.187–192.

43. Грушицкий Р. И., Мурсаев А. Х., Угрюмов Е. П. Проектирование систем на микросхемах программируемой логики // СПб.: БХВ-Петербург, 2002. – 683с.

44. Кун С., Матричные процессоры на СБИС // Москва: Мир, 1991. – 602 с.

45. Кормен Т., Лейзерсон Ч., Ривест Р., Алгоритмы: построение и анализ: Пер. с англ. / Под ред. А. Шеня. // Москва: МЦНМО: БИНОМ. Лаборатория знаний, 2004. – 900 с.

46. Петров И. В. Программируемые контроллеры. Стандартные языки и приёмы прикладного проектирования. / Под ред. проф. В. П. Дьяконова. // Москва: СОЛОН-Пресс, 2004. – 256 с.

47. Лорин Г. Сортировка и системы сортировки // Москва: Мир, 1983. – 348 с.

48. Мельничук А. С., Луценко С. П., Громовий Д. С., Трофимова К. В., Аналіз методів сортування масиву чисел // Технологический аудит и резервы производства - №4/1(12), 2013. – С. 26–31.

49. Воеводин В. В., Воеводин Вл. В. Параллельные вычисления // СПб: БХВ-Петербург, 2002. – 681 с.

50. Ахо А. В., Хопкрофт Д., Ульман Д. Д., Структуры данных и алгоритмы / Пер. с англ // Москва: Издательский дом “Вильямс”, 2000. – 387с.

51. Скорохода, Олекса Володимирович, Синтез нейроелементів і нейромереж реального часу паралельно-вертикального типу: автореф. дис. канд. техн. Наук. Львів, 2013. – 20 с.

52. Кухарев Г. А. и др., Техника параллельной обработки бинарных данных на СБИС // Москва: Выш. Шк., 1991. – 226 с.

53. Цмоць І. Г., Інформаційні технології та спеціалізовані засоби обробки сигналів і зображень у реальному часі. [текст] // Львів: УАД, 2005. – 227с.

54. Байков В. Д., Смолов В. Б., Специализированные процессоры: Интегральные алгоритмы и структуры //Москва: Радио и связи, 1985. – 288 с.

55. Лотов А. В., Бушенков В. А., Каменев Г. К., Черных О. Л., Компьютер и поиск компромисса. Метод достижимых целей // М.: Наука, 1997. – 239 с

56. Грицык В. В., Параллельная обработка информации: Т.4. Высокопроизводительные системы параллельной обработки информации // Киев: Наук. думка, 1988. – 272 с.

57. Мельник А. А. Процессоры обработки сигналов // Львов, 1989. – 63 с. – (Препринт / АН УССР, ИППММ; 29-89).

58. Цмоць І. Г., Особливості проектування спеціалізованих комп’ютерних систем для обробки інтенсивних потоків даних // Збірник наукових праць. Моделювання та інформаційні технології. – 1999. – Випуск 4. – С. 113–118.

59. Корнеев В. В., Киселев А. В., Современные микропроцессоры. 3-е изд. // СПб.: БХВ-Петербург, 2003. – 448 с.

60. Hammerstrom D. A, VLSI architecture for high-performance, low-cost, on-chip learning // Proceedings of the International Joint Conference on Neural Networks, SanDiego, CA, 1990. – Pp. 537–544.

61. Khalifa B. K., Girau B., Alexandre F., Bedoui M. H., Parallel FPGA implementation of self-organizing maps // ICM Proceedings. The 16th International Conference on Microelectronics/ Lviv, 2004. – Pp. 709–712.

62. Зотов. В., Особенности архитектуры нового поколения ПЛИС с архитектурой FPGA фирмы Xilinx // Компоненты и технологии. – 2010. – № 12. – С. 17–24.

63. Норенков И. П., САПР: Системы автоматизированного проектирования. Принципы построения и структур // Москва: Издательство МГТУ имени Н. Э. Баумана, 1987. – 123 с.

64. Боюн В. П., Управляющая вычислительная техника и системы реального времени в Украине. Состояние, проблемы, перспективы // УСиМ. – 1998. – №4. – С. 64–67.

65. Мельник А. О., Тарасенко В. П., Сучасні ситуативно-методологічні аспекти створення спеціалізованих комп'ютерних систем // Наукові вісті. Національний технічний університет КПІ. – 1997. – №1. – С. 18–21.

66. Палагин А., Опанасенко В., Реконфигурируемые вычислительные системы // К.: Просвіта, 2006. – 280 с.

67. Воеводин В. В., Параллельные вычисления // СПб.: БХВ-Петербург, 2002. – 608 с.

68. Поспелов Д. А., Введение в теорию вычислительных систем // М.: Советское радио, 1972. – 280 с.

69. Уилсон Р., Введение в теорию графов. Москва: Мир, 1977. 208с.

70. Татт У., Теория графов. Москва: Мир, 1988. – 424 с

71. Цмоць І., Іванців Г., Паралельно-вертикальний метод і базова структура пристрою обчислення сум парних добутків // Вісник Нац. ун-ту «Львівська політехніка»: Комп'ютерні науки та інформаційні технології. – Львів, 2008. – № 616. – С. 27–32.

72. Шалыто А. А., Методы аппаратной и программной реализации алгоритмов // Шалыто. – СПб.: Наука, 2000 – 780 с.

73. Боюн В. П., Динамическая теория информации. Основы и приложения. / В.П. Боюн. – К.: Ин-т кибернетики им. В.М. Глушкова НАН Украины, 2001. – 326 с.

74. Гамаюн В. П., О развитии многооперандных вычислительных структур // Управляющие системы и машины. – 1990. – №4. – С. 31–33.

75. Паулин О. Н., Ляховецкий А. М., Модель и метод проектирования многооперандных сумматоров на базе симметрических функций // Тези доповідей на міжнар. конф. з індуктив. моделювання МКІМ-2002. – Львів: ДНДІ. – 2002. – С. 208–213.

76. Мартко Е. О., Сингулярный спектральный анализ как метод моделирования электрической загрузки / Е.О. Мартко, И.В. Белицын // Ползуновский вестник. – Барнаул, 2009. – №4. – С. 76–85.

77. Sapienware – Software solutions with focus on engineering field and scientific investigations [Електронний ресурс]. URL: http://sapienware.net/

78. Гусениця SSA [Електронний ресурс]. URL: http://wiki.tntu.edu.ua/Гусениця_SSA .

79. ISE WebPACK Design Software [Електронний ресурс]. URL: http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm .

80. Ashenden P. J., The designer's guide to VHDL. Third Edition // Morgan Kaufmann, 2010. – 936 p.

81. Ott D. E., Wilderotter T. J., A designer's guide to VHDL synthesis // Springer Publisher, 2010. – 336 p.

82. Bloom G. S ., Golomb S . W., Applications of Numbered Undirected Graphs // Proceedings of the IEEE. – 1977. – Vol.65, No. 4. – Pp. 562–570.

83. Bloom G. S ., Golomb S . W. Numbered Complete Graphs, Unusual Rulers, and Assorted Applications // Theory and Applications of Graphs. – Springer, 1978. – Pp. 53–65.

84. Leech J. Another Tree Labelling Problem // The American Mathematical Monthly. – 1975. – Vol. 82, No. 9. – Pp. 923–925.

85. OpenCL 3.0 Materials [Електронний ресурс]. URL: https://www.khronos.org/opencl/

86. Kernighan B. W., Ritchie D. M., The C Programming Language, 2nd ed // Prentice Hall, 1988. – 304 p.

87. Stewart, Bill. History of the C Programming Language, 2000. [Електронний ресурс]. URL: https://livinginternet.com/i/iw_unix_c.htm

88. Dennis Ritchie. The Development of the C Language. [Електронний ресурс]. URL: https://www.bell-labs.com/usr/dmr/www/chist.html

89. Шпак З.Я. Програмування мовою С. // Львів: Видавництво Львівської політехніки, 2011. – 436 с.

90. C++ AMP Documentation. [Електронний ресурс]. URL: https://docs.microsoft.com/en-us/cpp/parallel/amp

91. CUDA Documentation. [Електронний ресурс]. https://www.nvidia.com/object/cuda_directcompute.html

92. Cron A., West M., Efficient Classification-Based Relabeling in Mixture Models // The American Statistician, №65, 2011. – Pp. 16 – 20

93. M. Suchard, Q. Wang, C. Chan, J. Frelinger, A. Cron and M. West.Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures // Journal of Computational and Graphical Statistics, №19, 2010. – Pp. 419-438

94. GPU. [Електронний ресурс]. https://www.techpowerup.com/gpu-specs/geforce-rtx-2060.c3310

95. Kimball R., The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3rd Edition // Wiley, 2013. – 917p.

96. Jukic N., Vrbsky S., Nestorov S., Database Systems: Introduction to Databases and Data Warehouses 1st Edition // Prospect Press, 2017. – 372p.

97. Korotkevitch D., Pro SQL Server Internals // Apress, 2016. – 776 p.

98. Цзясін Лю, Світ радіоелектроніки // Техносфера. – 2016. – 632 с.

99. Żurek D., Pietroń, M., Wielgosz, M., Wiatr, K., The comparison of parallel sorting algorithms implemented on different hardware platforms // Computer Science, Vol. 14(4), 2013. – p.679-691.

100. Chacon S., Straub B., Pro Git // Apress, 2014. – 419p.

101. Loeliger J., Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development // O'Reilly Media, Inc, USA, 2012. – 400p.

102. Sintorn E., Assarson U., Fast parallel GPU-sorting using a hybrid algorithm // Journal of Parallel and Distributed Computing, vol. 68, 2008. – Pp. 1381–1388.

103. Cederman D., Tsigas P., A practical quick-sort algorithm for graphics processors // Proc. of the 16th annual European Symposium on Algorithms, Springer-Verlag, Heidelberg, 2008. – Pp. 246–258.

104. Greß N. K., Zachmann G., Gpuabisort: Optimal parallel sorting on stream architectures // Proc. of the 20th IEEE International Parallel and Distributed Processing Symposium IPDPS , Heidelberg, 2006. – Pp. 1 – 24.

105. Sabahat S., M. IkramUllah Lali, Saqib Nawaz, Abou Bakar Nauman, Multi-Core Program Optimization: Parallel Sorting Algorithms in Intel Cilk Plus // International Journal of Hybrid Information Technology, Vol.7, No.2, 2014. – Pp.151 – 164.

106. Nadathur S., Harris M., Garland M., Designing Efficient Sorting Algorithms for many Core GPU's // IEEE International Symposium on Parallel & Distributed Processing, 2009. – Pp. 23 – 29.

107. Inoue H., Moriyama T., Komatsu H., Nakatani T., AA-Sort: A New Parallel Sorting Algorithm for Multi-Core SIMD Processors // IEEE 16th International Conference on Parallel Architectures and Compilation Techniques, 2007. pp. – 189– 198.

108. Fasiku I., Performance Evaluation of Multi-Core Processors // M. Tech Thesis, Federal University of Technology, Akure, Nigeria, 2012.

109. Sintroon E., Assarsson U., Fast Parallel GPU Sorting using a Hybrid Algorithm // Journal of Parallel and Distributed Computing, vol. 66, 2008. – Pp.1381 – 1388.

110. Man D., Ito Y., Nakano K., An Efficient Parallel Sorting Compatible with Standard qsort // International Conference on Parallel and Distributed Computing, Applications and Technologies, 2009. – Pp. 512– 517.

111. Joen M., Kim D., Parallel Merge Sort with Load Balancing // International Journal of Parallel Programming, vol. 31, 2003. – Pp. 21–33.

112. Bader D. A., Kanade V., Madduri K., SWARM: A Parallel Programming Framework for Multi-Core Processors // First Workshop on Multithreaded Architectures and Applications (MTAAP), 2007. – Pp. 21–33.

113. Helman D. R., Bader D. D., Jaja J.,A Randomized Parallel Sort Algorithm with an Experimental Study // Journal of Parallel and Distributed Computing, vol. 53, 1998. Pp. 1–23.

114. Dean J., Ghemawat S., MapReduce: Simplified Data Processing on Large Clusters // Communications of the ACM. Vol. 51, No. 1, 2008. – Pp. 107– 113.

115. X. -j. Qian, J. -b. Xu, Optimization and Implementation of Sorting Algorithm based on Multi-Core and Multi-Thread // IEEE 3rd International Conference on Communication Software and Networks, 2011. – Pp. 29– 32.

116. Masato E., Parallelizing Fundamental Algorithms such as Sorting on Multi-core Processors for EDA Acceleration // Proceedings of the 2009 Asia and South Pacific Design Automation Conference, 2009. – Pp. 230 – 233.

117. Shirako J., Kasahara H., Sarkar V., Language Extensions in Support of Compiler Parallelization // Languages and Compilers for Parallel Computing, Springer, vol. 5234, 2008. – Pp. 78– 94.

118. Graefe G., Implementing Sorting in Database Systems // ACM Computer Surveys, Vol. 38, No. 3, 2006. – Pp. 1–37.

119. Bilardi G., Nicolau A., Adaptive bitonic sorting: an optimal parallel algorithm for shared-memory machines // SIAM J. Comput, Vol. 18, No. 2, 1989. – Pp.216—228.

120. Toan Nguyen Mau, Yasushi Inoguchi, Audio fingerprint hierarchy searching strategies on GPGPU massively parallel computer // Journal of Information and Telecommunication, 2018. – Pp. 265–290.

121. Tianyi D. H., Tarek S. A., hiCUDA: High-Level GPGPU Programming // IEEE Transactions on Parallel and Distributed Systems, Vol. 22, Issue 1., 2011. – Pp. 78 – 90.

122. Ueng S.-Z., CUDA-lite: Reducing GPU Programming Complexity // Proc. Int'l Workshop Languages and Compilers for Parallel Computing, 2008. – Pp. 1–15.

123. Luk C.-K., Hong S., Kim H., Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping // Proc. Int'l Symp. Microarchitecture, 2009. – Pp. 45–55.

124. Ryoo S., Program Optimization Space Pruning for a Multithreaded GPU // Proc. Int'l Symp. Code Generation and Optimization, 2008. – P.p. 195–204.

125. Merill D. G., Grimshaw A. S., Revisiting sorting for GPGPU stream architectures // PACT '10: Proceedings of the 19th international conference on Parallel architectures and compilation techniques, September 2010. – Pp. 545 – 546.

126. Owens D. et al, GPU Computing // Proceedings of the IEEE, Vol. 96, No. 5, May 2008. – Pp. 879 – 899.

127. Satish N., Harris M., Garland M., Designing efficient sorting algorithms for manycore GPUs // in IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, 2009. – Pp. 1–10.

128. Chhugani J. et al., Efficient implementation of sorting on multi-core SIMD CPU architecture // Proc. VLDB Endow, 2008. – Pp. 1313 – 1324.

129. Yang Y., Xiang P., Kong J., A GPGPU compiler for memory optimization and parallelism management // ACM SIGPLAN Notices, Volume 45, Issue 6, June 2010. – Pp. 86– 97.

130. Kothapalli K. et al., A performance prediction model for the CUDA GPGPU platform // 2009 International Conference on High Performance Computing (HiPC), 16-19 Dec. 2009. – Pp. 463 – 472.

131. Guo P., Wang L., Accurate CUDA performance modeling for sparse matrix-vector multiplication // in The 2012 International Conference on High Performance Computing and Simulation (HPCS). IEEE, July 2012. – Pp. 496 –502.

132. Bolz J., Farmer I., Grinspun E., Schroder P., Sparse matrix solvers on the GPU: conjugate gradients and multigrid // ACM Trans. Graph., Vol. 22, No. 3, 2003. – Pp. 917–924.

133. Kurzak J., Alvaro W., Dongarra J., Optimizing matrix multiplication for a short-vector simd architecture-cell processor // Parallel Comput., Vol. 35, No. 3, 2009. – Pp. 138–150.

134. Xu S., Xue W., Lin H., Performance modeling and optimization of sparse matrix-vector multiplication on NVIDIA CUDA platform // The Journal of Supercomputing, 2011. – Pp. 1–12.

135. Cong J., Ding Y., FlowMap: an optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs // IEEE Transactions on Computer-

Aided Design of Integrated Circuits and Systems, Volume: 13, Issue: 1, Jan 1994. – Pp. 1 – 12.

136. Martinez J., Cumplido R., Feregrino C., An FPGA-based parallel sorting architecture for the Burrows Wheeler transform // 2005 International Conference on Reconfigurable Computing and FPGAs (ReConFig'05), 28-30 Sept. 2005. – Pp. 7-14.

137. Chen R., Siriyal S., Prasanna V., Energy and Memory Efficient Mapping of Bitonic Sorting on FPGA // FPGA '15: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, February 2015. – Pp. 240–249. DOI: https://doi.org/10.1145/2684746.2689068

138. Mihhailov D., Sklyarov V., Sudnistson A., Parallel FPGA-Based Implementation of Recursive Sorting Algorithms // 2010 International Conference on Reconfigurable Computing and FPGAs, 13-15 Dec. 2010. – Pp. 121 – 126.

139. Grozea C., Bankovic Z., LasKov P., FPGA vs. Multi-core CPUs vs. GPUs: Hands-On Experience with a Sorting Application // Lecture Notes in Computer Science book series (LNCS, volume 6310), 2010. . – Pp. 105– 117.

140. Koch D., Torresen J., FPGASort: a high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting // In Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA), 2011. – Pp. 45– 54.

141. Harkins, J., El-Ghazawi, T., El-Araby, E., Huang, M.: Performance of sorting algorithms on the SRC 6 reconfigurable computer. In: Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology, 2005. – Pp. 295–296.

142. Hou, Q., Zhou, K., Guo, B., BSGP: bulk-synchronous GPU programming // In: ACM SIGGRAPH 2008 papers, ACM, New York, 2008. – Pp. 19.

143. Marcelino R., Neto H., Cardoso J.M.P, Sorting Units for FPGA-Based Embedded Systems // IFIP – The International Federation for Information Processing, IFIPAICT, volume 271, 2008. – Pp. 11—22.

144. Chen H., Madaminov S., Ferdman M., Milder P., Sorting Large Data Sets with FPGA-Accelerated Samplesort // 2019 IEEE 27th Annual International Symposium on

Field-Programmable Custom Computing Machines (FCCM), 28 April-1 May 2019. – Pp. 326 – 326.

145. Srivastava A., Chen R., Prasanna V.K., Chelmis C., A hybrid design for high performance large-scale sorting on FPGA // 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 7-9 Dec. 2015. – Pp. 1 – 6.

146. Sklyarov V., Slkiarova I., High-performance implementation of regular and easily scalable sorting networks on an FPGA // Microprocessors and Microsystems, Volume 38, Issue 5, July 2014. – Pp. 470 – 484.

147. Dong S., Wang X., Wang X., A Novel High-Speed Parallel Scheme for Data Sorting Algorithm Based on FPGA // 2009 2nd International Congress on Image and Signal Processing, 17-19 Oct. 2009. – Pp. 1–4.

148. Abirami R., VHDL Implementation of Merge Sort Algorithm // International Journal of Computer Scienceand Communication Engineering, Vol. 3, No. 2, 2014. – Pp. 15–18.

149. Skliarova I., Sklyarov V., Mihhaailov D., Subnitson A., Implementation of sorting algorithms in reconfigurable hardware // 2012 16th IEEE Mediterranean Electrotechnical Conference, 25-28 March 2012. – Pp. 107 – 110.

150. Lobo J., Kuwelkar S., Performance Analysis of Merge Sort Algorithms // 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2-4 July 2020. – Pp. 110 – 115.

ДОДАТКИ

Додаток А. Список публікацій здобувача за темою дисертації

1. Цмоць І. Г., Антонів В. Я. Вертикально-паралельний метод сортування масивів чисел // Збірник наукових праць “Моделювання та інформаційні технології”. Інститут проблем моделювання в енергетиці ім. Г. Є. Пухова. 2016, Випуск 77. С. 186 – 192.

2. Цмоць І. Г., Антонів В. Я., Рабик В. Г. Метод вертикально-паралельного обчислення максимальних і мінімальних чисел у масивах // Збірник наукових праць “Моделювання та інформаційні технології”. Інститут проблем моделювання в енергетиці ім. Г. Є. Пухова. 2016, Випуск 76. С. 190 – 196.

3. Цмоць І. Г., Антонів В. Я. Удосконалення паралельного сортування масивів чисел методом злиття // Науковий вісник НЛТУ України : збірник наукових праць. Львів, 2020, Т. 30, № 4. С. 134 – 142.

4. Ivan Tsmots, Oleksa Skorokhoda, Volodymyr Antoniv. Parallel algorithms and structures for implementation of merge sort // International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 5 Issue 3, March 2016. Pp. 798 – 807.

5. Цмоць І. Г., Скорохода О. В., Красовський В. Б., Антонів В. Я. Методи та НВІС-структури узгоджено-паралельного обчислення максимальних і мінімальних значень // Збірник наукових праць інституту проблем моделювання в енергетиці ім. Г. Є. Пухова. 2014, Випуск 70. С.38 – 48.

6. Цмоць І. Г., Парубчак В. О., Антонів В. Я. Паралельно-вертикальне сортуванням одновимірних масивів даних методом злиття з використанням підрахунку // Збірник наукових праць інституту проблем моделювання в енергетиці ім. Г. Є. Пухова. 2013, Випуск 68. С.92 – 100.

7. Цмоць І. Г., Антонів В. Я. Апаратні засоби сортування даних методом злиття в реальному часі // Вісник національного університету “Львівська політехніка” Збірник наукових праць, 2015, № 814. - С. 171 – 186.

8. Цмоць І. Г., Кісь Я. П., Антонів В. Я. Застосування графічного процесора для підвищення швидкодії процесу сортування великих масивів даних // Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів : РВВ НЛТУ України. – 2015. – Вип. 25.6. – С. 328 – 334.

9. Ivan Tsmots, Oleksa Skorokhoda, Vasyl Rabyk, Volodymyr Antoniv Vertically-Parallel Method and VLSI-Structure sorting of Arrays of Numbers // Advances in Intelligent Systems and Computing III. Springer International Publishing AG 2019. Pp.267 – 284.

10.Цмоць І. Г., Антонів В. Я. Алгоритми та паралельні структури сортування даних методом вставки // Науковий вісник НЛТУ України: Збірник науково-технічних праць. – Львів : РВВ НЛТУ України. – 2016. – Вип. 26.1. – С. 340 – 350.

11.Ivan Tsmots, Oleksandr Kuzmin, Vasyl Dubuk, Volodymyr Antoniv Improvement and orientation of method of data arrays sorting by confluence on architecture of graphic processor unit // Advances in Intelligent Systems and Computing V. International Conference on Computer Science and Information Technologies, CSIT 2020, September 23-26, 2020, Zbarazh, Ukraine. Pp. 276 – 286.

12.Цмоць І. Г., Антонів В. Я. Метод розробки апаратних засобів паралельно-узгодженого сортування масивів даних у реальному часі // Матеріали міжнародної конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту» ISDMCI 2015, 25–28 травня. — Залізний Порт, 2015. – С.221 – 222.

13.Парубчак В. О., Антонів В. Я. Методи паралельного сортування одновимірних масивів даних // Матеріали міжнародної конференції «Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту» ISDMCI 2014, 28-31 травня. — Залізний Порт, 2014. – С.23 – 24.

14.Цмоць І. Г., Антонів В. Я. Застосування графічного процесора для підвищення швидкодії сортування великих масивів даних // Матеріали XIV Міжнародного наукового семінару «Сучасні проблеми інформатики в управлінні, економіці, освіті» , Світязь. – 2015. – С.90 – 92.

15.Цмоць І. Г., Антонів В. Я. Методи та апаратно-програмні засоби паралельно-вертикального сортування масивів чисел // Збірник наукових праць міжнародної

наукової конференції “Інтелектуальні системи прийняття рішень та проблеми обчислювального інтелекту ISDMCI’2016”. – Залізний Порт, 2016. – С. 226 – 227.

16.Tsmots I., Rabyk V., Skorokhoda O., Antoniv V. FPGA implementation of vertically parallel minimum and maximum values determination in array of numbers // 2017 14th International Conference The Experience of Desingning and Application of CAD Systems in Microelectronics (CADSM). PROCEEDINGS. Polyana. February 21-25, 2017. Pp. 234 – 236.

17.Tsmots I., Skorokhoda O., Antoniv V., Rabyk V. Vertically-Parallel Method and VLSI-Structure for Sorting of One-Dimensional Arrays // Computer Sciences and Information Technologies. In Proceedings of 13th International Scientific and Technical Conference CSIT 2018. Pp. 112 – 116.

18.Цмоць І. Г., Скорохода О. В., Медиковський М. О., Антонів В. Я. Пристрій для визначення максимального числа з групи чисел. Патент України на винахід №110187, 25.11.2015, Бюл. №22.

Додаток Б. Фрагменти коду розробленого програмного засобу

Робота кожного потоку (треду) CUDA:

```
@cuda.jit(«void(byte[:,:], byte[:,:])»)
    def sort(sharedInput, out):
        id = cuda.grid(1)
        threadNumber = id
        # seems like only hardcoded/static values
        workingRange = cuda.local.array(digits, numba.uint16)
        for i in range(digits):
            workingRange[i] = i
        for position in range(positions):
            numberOfOnes = 0
            integers = sharedInput[position]
            for i in workingRange:
                if (i > -1): numberOfOnes += integers[i]
            if (numberOfOnes == 0):
                out[id, position] = 0
                continue
            if (threadNumber < numberOfOnes):
                output = 1
            else:
                output = 0
            out[id, position] = output
            if (position != positions – 1):
                # changeRange function
                i = 0
                integers = sharedInput[position]
                for integer in workingRange:
                    if (integers[integer] == output):
                        workingRange[i] = integer
                        i += 1
                for element in range(i, len(workingRange)):
                    workingRange[element] = -1
                # changeThreadNumber function
                if (i == 1):
                    threadNumber = 0
                    else if (output == 0):
                    threadNumber -= numberOfOnes
```

Сортування злиттям:

```
#include <stdio.h>
#include <time.h>
extern "C" int mergesort(int *list, int *sorted, int n);
extern "C" int mergesort_cpu(int *list, int *sorted, int n);
//  GPU
__device__ void merge_gpu(int *list, int *sorted, int start, int mid, int end)
{
    int k=start, i=start, j=mid;
    while (i<mid || j<end)
    {
        if (j==end) sorted[k] = list[i++];
        else if (i==mid) sorted[k] = list[j++];
        else if (list[i]<list[j]) sorted[k] = list[i++];
        else sorted[k] = list[j++];
        k++;
    }
}
__global__ void mergesort_gpu(int *list, int *sorted, int n, int batch){
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    int start = tid * batch;
    if(start >= n) return;
    int mid, end;

    mid = min(start + batch/2, n);
    end = min(start + batch, n);
    merge_gpu(list, sorted, start, mid, end);
}
void mergesort_gpu_s(int *list, int *sorted, int n, int batch){
    int batch_id;
    for(batch_id=0; treat_id*batch<=n; treat_id++){
        int start = treat_id * batch, end, mid;
        if(start >= n) return;
```

```
        mid = min(start + batch/2, n);
        end = min(start + batch, n);
        merge(list, sorted, start, mid, end);
    }
}
int mergesort(int *list, int *sorted, int n){
    int *list_d;
    int *sorted_d;
    int dummy;
    bool flag = false;
    bool sequential = false;
    int size = n * sizeof(int);
    cudaMalloc((void **)&list_d, size);
    cudaMalloc((void **)&sorted_d, size);
    cudaMemcpy(list_d, list, size, cudaMemcpyHostToDevice);
    cudaError_t err = cudaGetLastError();
    if(err!=cudaSuccess){
        printf("Error_2: %s\n", cudaGetErrorString(err));
        return -1;
    }
    cudaDeviceProp prop;
    cudaGetDeviceProperties(&prop, 0);
    int major = prop.major;
    int minor = prop.minor;
    const int max_active_blocks_per_sm = 16;
    const int max_active_warps_per_sm = 64;
    int warp_size = prop.warpSize;
    int max_grid_size = prop.maxGridSize[0];
    int max_threads_per_block = prop.maxThreadsPerBlock;
    int max_procs_count = prop.multiProcessorCount;
    int max_active_blocks = max_active_blocks_per_sm * max_procs_count;
    int max_active_warps = max_active_warps_per_sm * max_procs_count;
    int batch_size;
    for(batch_size=2; batch_size<2*n; batch_size*=2){
        int blocks_required=0, threads_per_block=0;
        int threads_required = (n%batch_size==0) ? n/batch_size : n/batch_size+1;
        if (threads_required<=warp_size*3 && !sequential){
            sequential = true;
            if(flag) cudaMemcpy(list, sorted_d, size, cudaMemcpyDeviceToHost);
            else cudaMemcpy(list, list_d, size, cudaMemcpyDeviceToHost);
            err = cudaGetLastError();
            if(err!=cudaSuccess){
                printf("ERROR_4: %s\n", cudaGetErrorString(err));
                return -1;
            }
            cudaFree(list_d);
            cudaFree(sorted_d);
        }
        else if (threads_required<max_threads_per_block){
            threads_per_block = warp_size*4;
            dummy = threads_required/threads_per_block;
            blocks_required = (threads_required%threads_per_block==0) ? dummy : dummy+1;
        }
        else if(threads_required<max_active_blocks*warp_size*4){
            threads_per_block = max_threads_per_block/2;
            dummy = threads_required/threads_per_block;
            blocks_required = (threads_required%threads_per_block==0) ? dummy : dummy+1;
        }else{
            dummy = threads_required/max_active_blocks;
            // int estimated_threads_per_block = (dummy%warp_size==0) ? dummy : (dummy/warp_size + 1)*warp_size;
            int estimated_threads_per_block = (threads_required%max_active_blocks==0) ? dummy : dummy+1;
            if(estimated_threads_per_block > max_threads_per_block){
                threads_per_block = max_threads_per_block;
                dummy = threads_required/max_threads_per_block;
                blocks_required = (threads_required%max_threads_per_block==0) ? dummy : dummy+1;
            } else{
                threads_per_block = estimated_threads_per_block;
                blocks_required = max_active_blocks;
            }
        }
        if(blocks_required>=max_grid_size){
            printf("ERROR_2: Too many Blocks Required\n");
            return -1;
        }
        if(sequential){
            mergesort_gpu_seq(list, sorted, n, batch_size);
        }else{
            if(flag) mergesort_gpu<<<blocks_required, threads_per_block>>>(sorted_d, list_d, n, batch_size);
```

```
        else mergesort_gpu<<<blocks_required, threads_per_block>>>(list_d, sorted_d, n, batch_size);
        cudaDeviceSynchronize();
        err = cudaGetLastError();
        if(err!=cudaSuccess){
           printf("ERROR_3: %s\n", cudaGetErrorString(err));
           return -1;
        }
        flag = !flag;
      }
   }
   return 0;
}
```

Код пакету для інтеграціїї даних між технологічною базою даних і сховищем даних:

```
<Biml xmlns="http://schemas.varigence.com/biml.xsd">
<Packages>
        <Package Name="IT_Devices"
                ConstraintMode="Parallel"
                ProtectionLevel="DontSaveSensitive"
                DelayValidation="true"
                DisablePackageConfigurations="false"
        >
        <Variables>
                <Variable Name="IsCalledFromMasterVariable" DataType="Boolean" EvaluateAsExpression="false">False</Variable>
                </Variables>
        <Tasks>
                <ExecuteSQL ConnectionName="DW_System" Name="SQL_CleanUpImportTable">
                        <DirectInput>
                                truncate table etl.DevicesImport;
                        </DirectInput>
                </ExecuteSQL>
                        <Dataflow Name="DFT_GetDevicesToImportTable">
                                <Transformations>
                                        <OleDbSource ConnectionName="OLTP_System" Name="OLE_SRC_OLTPDevicesSource"
ValidateExternalMetadata="false" DefaultCodePage="1252" AlwaysUseDefaultCodePage="true">
                                                <DirectInput>
                                                        <![CDATA[
                                                                select convert(nvarchar(20), i.DeviceId) as DeviceId
                                                                        , convert(nvarchar(200), case i.DeviceName when '' then
i.[Description] else i.DeviceName] end) as [DeviceName]
                                                                        , DevicePrice
                                                                        , convert(nvarchar(70), i.[Description]) as Description
                                                                        , convert(nvarchar(10), i.[Vendor]) as Vendor
                                                                        , convert(decimal(18, 2), i.[width]) as width
                                                                        , convert(decimal(12, 4), i.[lengh]) as lengh
                                                                        , convert(decimal(12, 4), i.[mass]) as mass
                                                                        , i.[Reliability] as Reliability
                                                                        , i.[type] as type
                                                                from dbo.[OLTP_Devvices] as i
                                                        ]]>
                                                </DirectInput>
                                        </OleDbSource>
                                        <OleDbDestination ConnectionName="DW_System"
Name="OLE_DST_OLTP_DevicesImportTable" ValidateExternalMetadata="false" DefaultCodePage="1252" AlwaysUseDefaultCodePage="true"
UseFastLoadIfAvailable="true">
                                                <ExternalTableOutput Table="etl.DevicesImport"/>
                                                <InputPath OutputPathName="OLE_SRC_OLTPDevicesSource.Output"/>
                                        </OleDbDestination>
                                </Transformations>
                                <PrecedenceConstraints>
                                        <Inputs>
                                                <Input OutputPathName="SQL_CleanUpImportTable.Output" />
                                        </Inputs>
                                </PrecedenceConstraints>
                        </Dataflow>
                        <ExecuteSQL ConnectionName="DW_System" Name="SQL_MergeDataOfDevices">
                                <PrecedenceConstraints>
                                        <Inputs>
                                                <Input OutputPathName="DFT_GetDevicesToImportTable.Output" />
                                        </Inputs>
                                </PrecedenceConstraints>
                                <DirectInput>
                                        exec etl.spDevicesImportMerge ?;
                                </DirectInput>
                                <Parameters>
```

```
                                        <Parameter Name="@IsCalledFromMaster" VariableName="User.IsCalledFromMasterVariable"
DataType="Boolean" Direction="Input"></Parameter>
                                    </Parameters>
                                </ExecuteSQL>
                        </Tasks>
                        <Events>
                                <Event Name="EH_OnError" EventType="OnError" ConstraintMode="Linear">
                                        <Tasks>
                                                <ExecuteSQL Name="SQL_LogErrorInformation" ConnectionName="Logs"
IsStoredProcedure="true">
                                                        <DirectInput>
                                                                exec ssis.spLogToPackagesErrorLogs @ErrorCode = ?
                                                                        , @ErrorDescription = ?
                                                                        , @SourceName = ?
                                                                        , @PackageName = ?
                                                                        , @MachineName = ?
                                                        </DirectInput>
                                                        <Parameters>
                                                                <Parameter Name="0" VariableName="System.ErrorCode"
DataType="Int32" Direction="Input"></Parameter>
                                                                <Parameter Name="1" VariableName="System.ErrorDescription"
DataType="String" Direction="Input"></Parameter>
                                                                <Parameter Name="2" VariableName="System.SourceName"
DataType="String" Direction="Input"></Parameter>
                                                                <Parameter Name="3" VariableName="System.PackageName"
DataType="String" Direction="Input"></Parameter>
                                                                <Parameter Name="4" VariableName="System.MachineName"
DataType="String" Direction="Input"></Parameter>
                                                        </Parameters>
                                                </ExecuteSQL>
                                        </Tasks>
                                </Event>
                        </Events>
                                <Connections>
                                        <Connection ConnectionName="DW_System" RetainSameConnection="false"/>
                                        <Connection ConnectionName="Logs" RetainSameConnection="false"/>
                                        <Connection ConnectionName="OLTP_System" RetainSameConnection="false"/>
                                </Connections>
                                <PackageConfigurations>
                                        <!-- XML Configuration File -->
                                        <!-- The name is for the name in the Package Configurations Organizer window-->
                                        <PackageConfiguration Name="SiteCore_ProductCatalogReplication">
                                                <!-- ExternalFilePath is the path of the config file  -->
                                                <ExternalFileInput
ExternalFilePath="c:\NemligSSIS_config_files\SiteCore_ProductCatalogReplication.dtsConfig" />
                                        </PackageConfiguration>
                                </PackageConfigurations>
                        </Package>
        </Packages>
        <Connections>
                        <OleDbConnection Name="DW_System"
                                ConnectionString="Data Source=.;Initial Catalog=DW_System;Provider=SQLNCLI11.1;Integrated Security=SSPI;Auto
Translate=False;"
                                DelayValidation="true"
                        />
                        <OleDbConnection Name="Logs"
                                ConnectionString="Data Source=.;Initial Catalog=Log;Provider=SQLNCLI11.1;Integrated Security=SSPI;Auto
Translate=False;"
                                DelayValidation="true"
                        />
                        <OleDbConnection Name="OLTP_System"
                                ConnectionString="Data Source=.;Initial Catalog=OLTP;Provider=SQLNCLI11.1;Integrated Security=SSPI;Auto
Translate=False;"
                                DelayValidation="true"
                        />
        </Connections>
</Biml>
```

Код для розгортання бази даних:

```
param(
   [string] $PublishEnvironment = $null,
   [string] $SolutionFolderPath = $null,
   [string] $DeployLogsFolderPath = $null,
   [string[]] $DbProjects = $null,

   [string] $DW_username = "Sql_DW_deolyment",
```

```
    [string] $DW_userpassword = "",

    [string] $OLTP_username = "Sql_OLTP_deployment",
    [string] $OLTP_userpassword = "",

    [string] $Configuration = "Debug"
)
#initialization
    [string[]]$iDbProjects = $null
    [bool]$PublishToDatabase = $true
    [bool]$AutoLaunchBuildLogOnFailure = $false
            [bool]$ShowErrorsInTheOutput = $true
#custom settings
    #$iDbProjects = ""
    #$autoLaunchBuildLogOnFailure = $true
#setting up list of databases from parameter
if ($DbProjects)
{
    $iDbProjects = $DbProjects
}
[string] $iSolutionFolderPath = $null
if (-Not ([string]::IsNullOrEmpty($SolutionFolderPath)))
{
    $iSolutionFolderPath = $SolutionFolderPath
}
else
{
    $key = "Registry::HKEY_CURRENT_USER\Software\ITParallelSort"
    $value = "SolutionPath"
    $regSolutionPath = (Get-ItemProperty -Path $key -Name $value).$value
    $iSolutionFolderPath = $regSolutionPath
}
if (-Not ([string]::IsNullOrEmpty($DW_userpassword)))
{
    $serverCredentials = [ordered]@{
        "DW" = @("$DW_username", "$DW_userpassword");
        "util2" = @("$OLTP_username", "$OLTP_userpassword");
    }
}
else
{
    $serverCredentials = $null
}
$invokeDbDeploymentPath = $iSolutionFolderPath + "\DatabaseProjects\_scripts\Invoke-DbDeployment.psm1"
Import-Module -Name $invokeDbDeploymentPath -Force
$isFailed = 1
if ($serverCredentials -ne $null)
{
    Invoke-DbDeployment -PublishEnvironment $PublishEnvironment `
        -DbProjects $iDbProjects `
        -SolutionFolderPath $iSolutionFolderPath `
        -BuildLogDirectoryPath $deployLogsFolderPath `
        -ServerCredentials $serverCredentials `
        -AutoLaunchBuildLogOnFailure $autoLaunchBuildLogOnFailure `
                        -ShowErrorsInTheOutput $ShowErrorsInTheOutput `
        -PublishToDatabase $PublishToDatabase `
        -Configuration $Configuration `
        -IsDebug $false `
        -isFailed ([ref]$isFailed) `
        -ErrorAction Stop
}
else {
    Invoke-DbDeployment -PublishEnvironment $PublishEnvironment `
        -DbProjects $iDbProjects `
        -SolutionFolderPath $iSolutionFolderPath `
        -BuildLogDirectoryPath $deployLogsFolderPath `
        -AutoLaunchBuildLogOnFailure $autoLaunchBuildLogOnFailure `
                        -ShowErrorsInTheOutput $ShowErrorsInTheOutput `
        -PublishToDatabase $PublishToDatabase `
        -Configuration $Configuration `
        -IsDebug $false `
        -isFailed ([ref][int]$isFailed) `
        -ErrorAction Stop
}

exit $isFailed
```

Додаток В. Акти впровадження результатів дисертаційного дослідження

"ЗАТВЕРДЖУЮ"
Проректор з науково-педагогічної роботи
Національного університету
"Львівська політехніка"
_____ О.Р. Давидчак
"01" 01 2021 р.

АКТ
про впровадження в навчальний процес результатів
кандидатської дисертаційної роботи
Антоніва Володимира Ярославовича

Цей акт складено про те, що результати кандидатської роботи Антоніва Володимира Ярославовича на тему "Інформаційні технології паралельного сортування та пошуку даних" представленої на здобуття наукового ступеня кандидата технічних наук, використовуються у навчальному процесі кафедри "Автоматизованих систем управління" Національного університету "Львівська політехніка". Матеріали дисертаційного дослідження використовуються під час написання студентами курсових робіт, кваліфікаційних бакалаврських та магістерських робіт, а також під час викладання дисципліни: "Організація баз даних та знань".

Зокрема в навчальному процесі використовується запропоновані В.Я. Антоніва:

- вертикально паралельний метод сортування даних, який за рахунок паралельного опрацювання розрядного зрізу масиву чисел і паралельного формування розрядного зрізу відсортованого масиву чисел забезпечує зменшення часу обробки великих масивів даних у сховищах даних;
- вдосконалений метод паралельного сортування даних злиття, який за рахунок використання базової операції об'єднання двох масивів з одночасним формуванням елементів зростаючого і спадаючого масивів забезпечує зменшення часу сортування даних у сховищах даних.

Ефект від використання результатів кандидатської дисертаційної роботи Антоніва В.Я. полягає у вивчені майбутніми фахівцями сучасних методів, алгоритмів та засобів паралельного сортування та пошуку даних для збільшення швидкодії операцій обробки великих масивів даних.

Доцент каф. автоматизованих систем управління
к.т.н., доцент А.В. Дорошенко

в.о. зав. каф. автоматизованих систем управління,
д.т.н., професор В.М. Теслюк

"ЗАТВЕРДЖУЮ"
Проректор з наукової роботи
Національного університету
"Львівська політехніка"
проф. _____ Демидов І.В.
"__" 01 2021 р.

АКТ
про використання результатів дисертації
"Інформаційні технології паралельного сортування та пошуку даних"
асистента кафедри автоматизованих систем управління
Антоніва Володимира Ярославовича
представленого на здобуття наукового ступеня кандидата технічних наук
за спеціальністю 05.13.06 – Інформаційні технології
при виконанні науково-дослідних роботі Національного університету "Львівська політехніка"

Ми, що нижче підписалися, начальник НДЧ д.т.н., ст.досл. Небесний Р.В. та члени комісії: завідувачка відділу науково-організаційного супроводу наукових досліджень к.т.н. Лазько Г.В. завідувачка планово-фінансового відділу Чулой Т.М. та завідувач кафедри автоматизованих систем управління Теслюк Василь Миколайович цим актом підтверджуємо, що результати дисертаційної роботи асистента кафедри автоматизованих систем управління Антоніва Володимира Ярославовича використано під час виконання науково-дослідної роботи Національного університету «Львівська політехніка»: "Інтелектуальні інформаційні технології багаторівневого управління енергоефективністю регіону" (державний реєстраційний №0117U004450).

В рамках науково-дослідної роботи Антонів В.Я. розробив метод вертикально-паралельного сортування даних, який за рахунок паралельного опрацювання розрядного зрізу масиву чисел і паралельного формування розрядного зрізу відсортованого масиву чисел забезпечує зменшення часу сортування, на основі цього методу розробив програмні засоби паралельного сортування даних, який за рахунок використання багатопроцесорних системи та багатоядерні процесорів забезпечило зменшення часу обробки даних та вартість системи.

Голова комісії:

Начальник науково-дослідної частини, д.т.н., ст.досл — Р.В. Небесний

Члени комісії:

зав. відділу науково-організаційного супроводу наукових досліджень, к.т.н. — Г.В. Лазько

заст. начальника планово-фінансового відділу — Т.М. Чулой

в.о. зав. каф. автоматизованих систем управління, д.т.н., професор — В.М. Теслюк

ОКРЕМЕ КОНСТРУКТОРСЬКЕ БЮРО "ТЕКОН-ЕЛЕКТРОН"

ДОЧІРНЄ ПІДПРИЄМСТВО ПрАТ "КОНЦЕРН-ЕЛЕКТРОН"

р/р UA863253650000000731 6
у ПАТ «Кредобанк» м. Львов
МФО 325365; ЄДРПОУ 20848471
Інд. № 208484713037

вул. Стороженка,12, м. Львів, Україна, 79018
тел.: (032) 233-72-02, (032) 239-52-02;
факс: (032) 239-55-84
E-mail:okb1@i.ua

"26"липня 2021 р. № 067-69

АКТ

впровадження результатів кандидатської дисертаційної роботи

Антоніва Володимира Ярославовича.

Комісія у складі начальника сектору Лаб'яка Романа Степановича, провідного інженера програміста Равського Віталія Михайловича склали дійсний акт про те, що результати кандидатської дисертаційної роботи Антоніва Володимира Ярославовича на тему "ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПАРАЛЕЛЬНОГО СОРТУВАННЯ ТА ПОШУКУ ДАНИХ" представлені на здобуття наукового ступеня кандидата технічних наук, використовуються при розробленні засобів збору та попереднього опрацювання телеметричних даних, а саме.

- інформаційну технологію паралельного сортування даних, яка за рахунок використання розроблених і вдосконалених методів, функціональних моделей паралельно-потокового сортування даних та врахуванню інтенсивності надходження даних, розмірів масивів даних і засобів реалізації забезпечує виконання сортування даних у реальному часі з високою ефективність використання обладнання;
- метод паралельно-вертикального пошуку максимальних і мінімальних чисел у масивах, який за рахунок паралельного опрацювання і-о розрядного зрізу масиву чисел і паралельного формування слів управління забезпечує зменшення часу пошуку, який визначається в основному розрядністю чисел;
- метод паралельно-вертикального сортування даних, який завдяки підрахунку одиниць у і-у вхідному розрядному зрізі та паралельному формуванню і-о розрядного зрізу відсортованого масиву чисел забезпечує зменшення часу сортування.

Отримані в дисертаційній роботі результати представляють практичну цінність при розробленні програмно-апаратних засобів збору та попереднього опрацювання телеметричних даних. Розроблені і удосконалені методи порівняно з відомими відзначається високою ефективністю використання обладнання і забезпечують опрацювання даних в реальному часі. Це дозволяє, застосовуючи розроблені методи, підвищити ефективність засобів збору та попереднього опрацювання телеметричних даних і уникнути при цьому, дорогих та довготривалих фізичних експериментальних досліджень та випробувань.

Даний акт не є основою для проведення фінансових взаєморозрахунків.

Генеральний директор М.І. Гладкий

"ЗАТВЕРДЖУЮ"
Директор ДП "Львівський
державний завод "ЛОРТА"
Державного концерну
"Укроборонпром", Заслужений
машинобудівник України
Чаус В.М.
"__" _____2021 р.

АКТ
впровадження результатів кандидатської дисертаційної роботи
Антоніва Володимира Ярославовича

Комісія у складі: Головного конструктора д.т.н., професора, Оліярника Б.О., начальника відділу систем управління к.т.н., Євтушенко К.С., заступника. начальника відділа систем керування Власюка П.С. склали дійсний акт про те, що результати кандидатської дисертаційної роботи Антоніва Володимира Ярославовича на тему "ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ ПАРАЛЕЛЬНОГО СОРТУВАННЯ ТА ПОШУКУ ДАНИХ" представлені на здобуття наукового ступеня кандидата технічних наук, враховані при розробленні засобів попередньої обробки (сортуванні) телекодових даних в підсистемі зв'язку машин управління комплексу «Оболонь-А».

Отримані в дисертаційній роботі результати представляють практичну цінність при розроблені програмно-апаратних засобів збору та попереднього опрацювання масивів даних. Розроблені і удосконалені методи порівняно з відомими відзначається високою ефективністю використання обладнання і забезпечують опрацювання даних в реальному часі.

Даний акт не є основою для проведення фінансових взаєморозрахунків.

Головний конструктор
д.т.н., професор — Б.О. Оліярник

Нач. відділу
к.т.н. — К.С. Євтушенко

Заст. нач.відділу — П.С. Власюк