УДК 681.3.06

АРХІТЕКТУРА ПРОГРАМНОЇ СИСТЕМИ ІЄРАРХІЧНОГО НЕЧІТКОГО ЛОГІЧНОГО ВИВЕДЕННЯ

С.В. Єршов, Р.М. Пономаренко

Розроблено програмну архітектуру для системи ієрархічного нечіткого логічного виведення на основі алгоритмічних моделей нечіткого логічного виведення та окремих технологій паралельних обчислень, таких як MPI та CUDA. Представлено ключові рішення при проектування архітектури для забезпечення гнучкості та продуктивності паралельної системи нечіткого виведення. Проаналізовано шаблони проектування та програмування різного рівня абстракції, а також основні програмні модулі та інтерфейси програмної системи нечіткого виведення. Побудовано метод контейнеризації застосунків для GPU з метою їх повторного використання та розгортання на обраній програмній платформі високопродуктивних паралельних обчислень.
Ключові слова: архітектура програмної системи, шаблони паралельного проектування, технологія паралельних обчислень, нечітке логічне виведення.

Разработана программная архитектура для систем иерархического нечеткого логического вывода на основе алгоритмических моделей нечеткого логического вывода и отдельных технологий параллельных вычислений, таких, как MPI и CUDA. Представлены ключевые решения при проектировании архитектуры для обеспечения гибкости и продуктивности параллельной системы нечеткого вывода. Проанализировано шаблоны проектирования разного уровня абстракции, а также основные программные модули и интерфейсы программной системы нечеткого вывода. Создан метод контейнеризации приложений для GPU с целью их повторного использования и развертывания на выбранной программной платформе высокопродуктивных параллельных вычислений.
Ключевые слова: архитектура программной системы, шаблоны параллельного программирования, технология параллельных вычислений, нечеткий логический вывод.

A software architecture for hierarchical fuzzy logic hierarchy based on fuzzy logic algorithmic models and separate parallel computing technologies such as MPI and CUDA has been developed, as well as key architectural design solutions to provide the flexibility and performance of a parallel fuzzy system. The patterns of designing and programming of different levels of abstraction, as well as the main software modules and interfaces of the fuzzy software system are analyzed. The method of containerization of applications for the GPU for their reuse and deployment on the selected software platform of high-performance parallel computing is presented.
Key words: program architecture, patterns of parallel programming, parallel computing technology, fuzzy inference.

При проектуванні складних програмних систем програмна архітектура має містити набір ключових проектних рішень, що дозволяють визначити напрям подальшого проектування. Архітектурні рішення мають на меті забезпечити нефункціональні вимоги до програмного забезпечення, що також встановлюються на початкових стадіях проектування. Проектування програмної архітектури надає можливість застосовування раніше апробованих рішень, які зарекомендували себе в аналогічних задачах. Типові проектні рішення на різних рівнях абстракції представляються у вигляді архітектурних шаблонів (патернів) та шаблонів проектування низького рівня [1].

В [2–5] розроблено методи та моделі прискореного ієрархічного нечіткого логічного виведення на основі технологій паралельного програмування. Системи нечіткого логічного виведення широко використовуються у багатьох сферах промисловості, бізнесу, медицини, науки тощо. Такі системи призначені для вирішення задач прийняття рішень в умовах невизначеності, моделювання інтелектуальних та експертних систем та ін. [6, 7]. Метою даної роботи є представлення принципів та методів побудови архітектурних моделей програмних систем ієрархічного нечіткого логічного виведення.

Програмна система ієрархічного нечіткого логічного виведення (ІНЛВ) реалізує комплекс методів паралельного ієрархічного нечіткого логічного виведення на основі окремих апаратних платформ. Метою програмних систем ІНЛВ є отримання швидких результатів обчислень, а також експериментальне дослідження стосовно прискорення паралельних методів нечіткого логічного виведення. Основну проблему при проектуванні систем ІНЛВ становить об’єднання різних за побудовою паралельних методів ієрархічних нечітких систем в єдиному програмному комплексі, та надання можливості подальшого розширення такої програмної системи шляхом додавання нових алгоритмів та методів. Для забезпечення відповідного прискорення розроблений метод послідовного виконання ієрархічних систем нечіткого логічного виведення порівнювався з паралельними версіями для однакових задач [2–5]. Обчислення на CPU реалізовані на основі технології багатопроцесорних розподілених обчислень з використанням суперкомп’ютера СКІТ-4 Інституту кібернетики імені В.М. Глушкова НАН України. Обчислення на GPU відбуваються з використанням графічного прискорювача та реалізовані на основі персонального комп’ютера. За основу була взята відеокарта NVIDIA GTX 1050Ti апаратної архітектури Pascal.

Зазначимо, які саме алгоритми та технології були запроектовані в програмній системі обчислення ієрархічних систем нечіткого логічного виведення:

1) на основі послідовної форми обчислень для CPU;
2) на основі ярусно-паралельної форми обчислень для CPU (технологія MPI);

3) на основі динамічної паралельної моделі для CPU (технологія MPI);

4) на основі технології CUDA для графічних прискорювачів NVIDIA.

Використання різних програмно-апаратних підходів вимагає створення такої архітектури програмної системи, яка б дозволяла, по-перше, мінімізувати кількість інтерфейсних модифікацій при додаванні нового підходу. По-друге, зазначена архітектура має забезпечити одночасне використання всієї множини зазначених алгоритмів в межах однієї програмної системи, надаючи користувачу можливість вибору програмно-апаратних засобів.

На рис. 1 представлено узагальнену архітектуру програмної системи, що представлена архітектурними шарами на основі класичного багатошарового шаблону проектування. Архітектурний шаблон передбачає багатошарову організацію архітектури, шари якої є закритими. На відміну від багатошарової архітектури Open–layered, в класичній багатошаровій архітектурі є можливість взаємодії шарів тільки в порядку їх слідування.

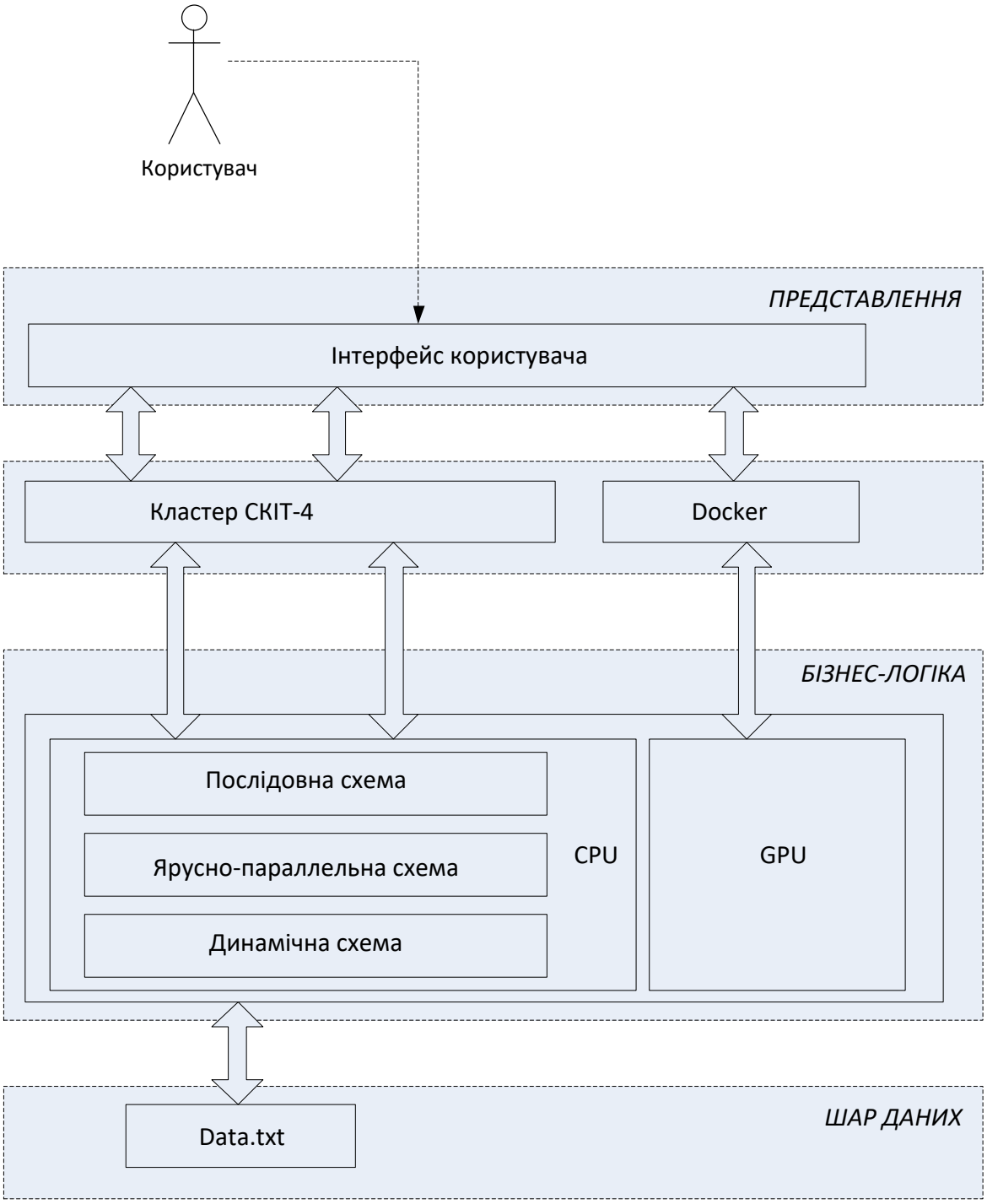

Рис. 1. Узагальнена архітектура програмної системи ієрархічного нечіткого логічного виведення

Представлена на рис.1 архітектура містить наступні чотири шари:

1) представлення – містить графічний інтерфейс користувача, який взаємодіє як з суперкомп'ютером СКІТ-4 [8], так і з програмно-апаратною частиною персонального комп'ютера;

2) шар сервісу – представлений системою команд для запуску задач на кластері СКІТ-4, а також на платформі Docker;

3) бізнес-логіка – архітектурний шар, який реалізує структурну та алгоритмічну складову обчислювального процесу програмної системи ІНЛВ, та підтримує дві апаратні платформи, що забезпечують обчислення на центральному процесорі, та обчислення на основі графічних прискорювачів. Також на центральному процесорі реалізована послідовна схема обчислень ієрархічних нечітких систем;

4) шар даних – представлений у вигляді текстового файлу, що містить вхідні дані системи.

На рис. 2. показана модель ярусно-паралельної схеми обчислень, що ґрунтується на декомпозиції по задачам (Functional decomposition) зі статичним плануванням [9]. Кожен процес відповідає за окрему елементарну нечітку систему, тобто композиція відбувається по задачам. Планування декомпозиції виконується перед початком роботи для всіх ярусів обчислень. Як планувальник зазвичай виступає процес з молодшим індексом (у даному випадку процес 1). Ярус – це кількість систем, у яких може бути паралельно здійснено нечітке логічне виведення, враховуючи кількість наявних процесів. Показана на рис. 2. модель застосовується для розподіленого виведення у паралельних ієрархічних нечітких системах на основі CPU. Стрілками показано напрям передачі даних. Процеси можуть обмінюватися даними, в залежності від інформаційних залежностей між нечіткими системами.

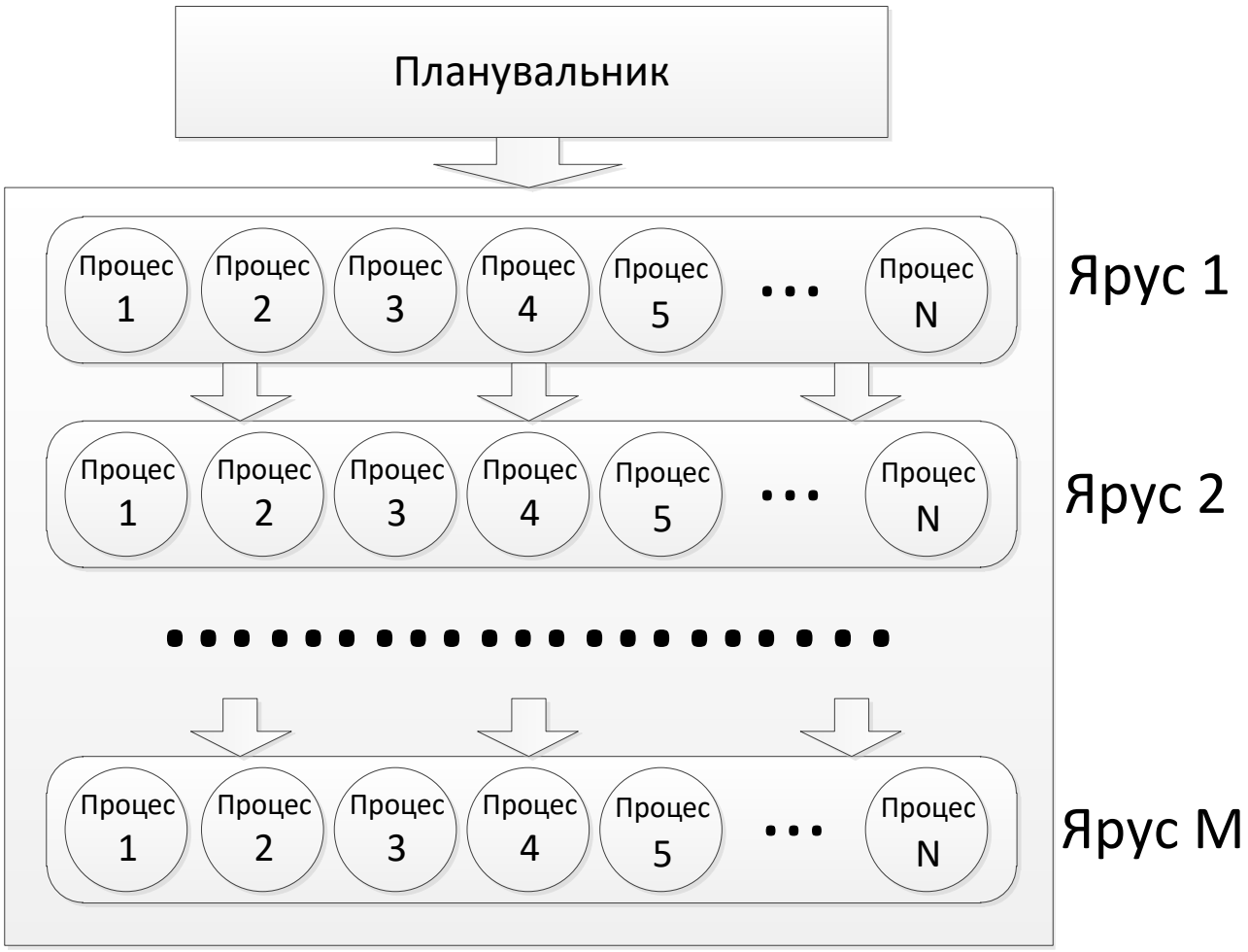

Рис. 2. Ярусно-паралельна модель обчислень на основі декомпозиції по задачам (Functional decomposition) зі статичним плануванням

Розроблено модель паралельних обчислень ієрархічних систем нечіткого логічного виведення, яка базується на основі шаблону (патерну) паралельного проектування Master-Slave (Майстер-Робітники) [9] та показана на рис. 3. На відміну від ярусно-паралельної моделі, динамічна модель містить динамічний планувальник робіт, що виступає як «Майстер». Слід зазначити, що «Майстер» не бере участі в безпосередніх обчисленнях. Його головні обов'язки полягають у швидкому розподілі задач по процесам, а також у виконанні комунікативних функцій, передачі вхідних даних та прийманні отриманих результуючих та проміжних даних, які акумулюються зазначеним процесом «Майстер».

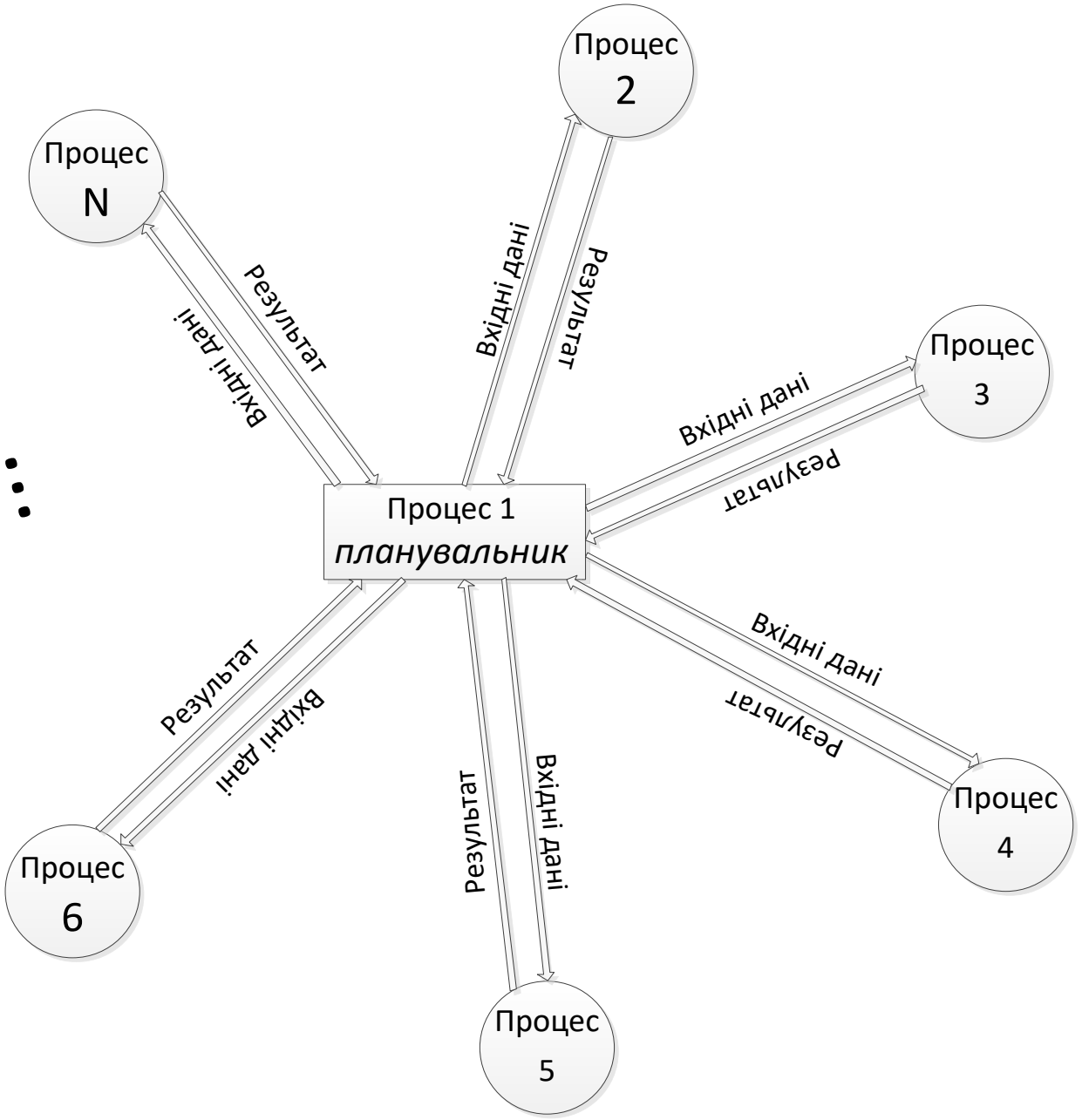

Рис. 3. Динамічна модель паралельних обчислень на основі шаблону паралельного проектування Master-Slave

Модель обчислень для графічних прискорювачів базується на моделі ярусно-паралельної форми [10], але має два рівні паралелізації (рис. 4). Планування для всіх рівнів здійснюється на початку обчислень загальним планувальником. Таким чином, обчислення кожної елементарної нечіткої системи здійснюється паралельно, що позначено на рис. 4 великими кружками.

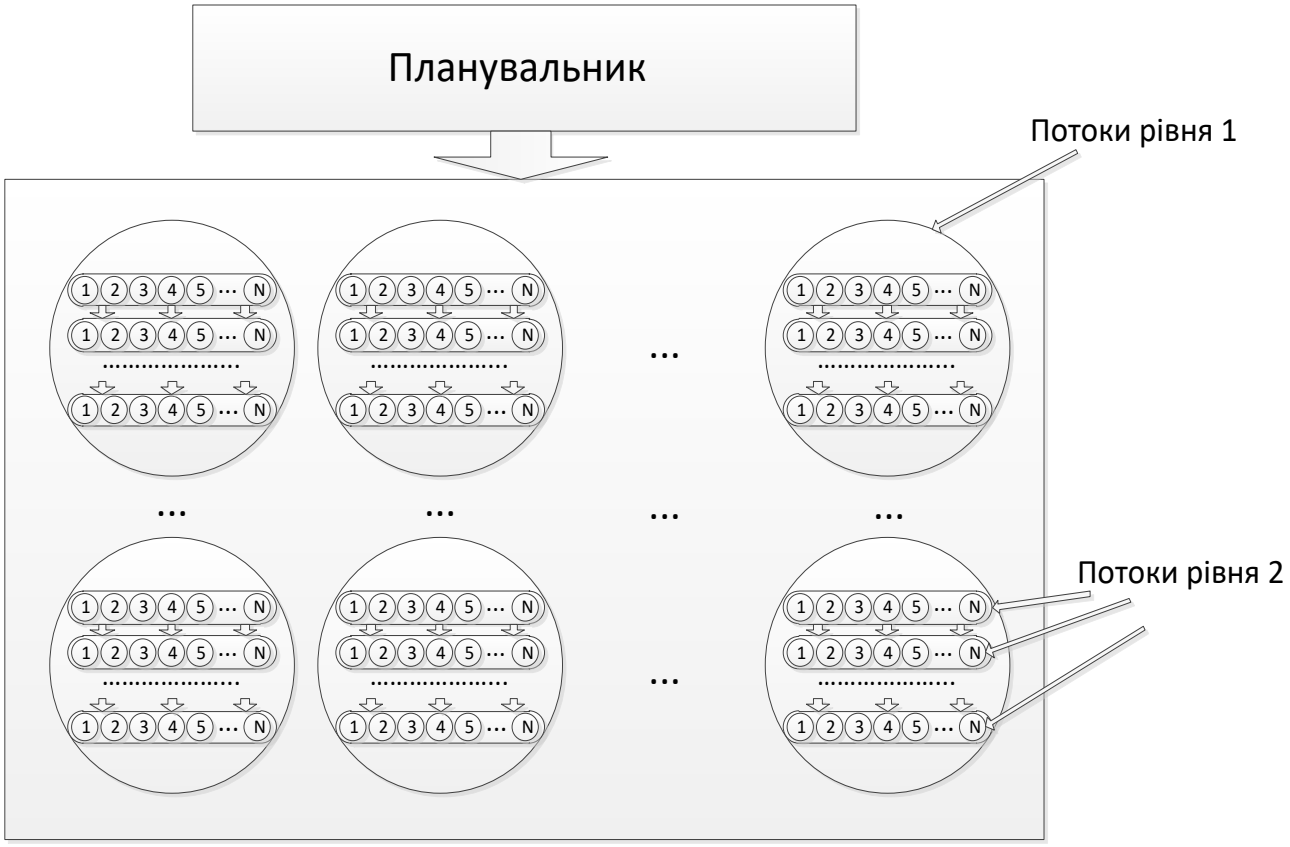

Рис. 4. Модель паралельних обчислень для GPU на основі дворівневої декомпозиції по задачам

Моделі розпаралелювання ієрархічних систем нечіткого логічного виведення побудовані на основі шаблону паралельної обробки Map-Reduce [11], який призначений для обробки наборів даних великих розмірностей. Обчислення на основі моделі Map-Reduce розподіляються на 2 етапи:

- розподілена обробка частки обчислень (Map);
- підсумовуюча згортка проміжних результатів для подальшого обчислення, або остаточне завершення обчислень (Reduce);

При проектуванні алгоритмічної схеми для системи ІНЛВ був застосований патерн низькорівневого проектування Factory Method [12], основне призначення якого полягає у наданні можливостей визначення щодо того, який клас буде реально створений, що є доцільним відповідно до поставленої задачі гнучкого та швидкого розширення системи ІНЛВ (рис. 5).

Розглянемо детальніше показану на рис. 3 діаграму класів для архітектурного шару бізнес-логіки. Головним класом виступає клас hierarchicalFuzzySystem, який агрегує класи CPU та GPU, кожен з яких моделює низку методів для апаратної платформи на основі паралельного програмування на центральних процесорах (CPU) та на графічних прискорювачах (GPU) відповідно. Він містить атрибут fuzzyNet, що представляє деревоподібну структуру та реалізує граф залежностей між нечіткими системами. Агрегація є слабким зв'язуванням класів, тому існує можливість модифікувати архітектуру шляхом додавання нових апаратних технологій без істотних змін іншого функціоналу програмної системи.

Клас CPU містить метод setData(), що забезпечує передачу вхідних даних, та забезпечує наступні методи ієрархічного нечіткого логічного виведення на основі обчислень на центральному процесорі:

- nonParallel() – послідовне виконання ієрархічної системи;
- tiredParallel() – паралельне виконання обчислень на основі ярусно-паралельної форми обчислень;
- dynamicParallel() – паралельне виконання на основі динамічної форми обчислень.

Тіло класу GPU містить метод setData() що є аналогом однойменного методу в класі CPU, але також містить метод loadDataToGPU(), призначення якого – копіювання вхідних даних на графічний пристрій. Метод gpuParallel() моделює метод паралельного ієрархічного нечіткого логічного виведення на графічному прискорювачі.

Слід зазначити, що класи CPU та GPU містять всю інформацію стосовно ієрархії нечітких елементарних систем. Робота кожної елементарної нечіткої системи моделюється інтерфейсом FuzzySystem та його похідними класами FuzzySystemCPU (для техногогії на основі центральних процесорів) та FuzzySystemGPU (для технології на основі графічних прискорювачів). Розглянемо основні абстрактні методи інтерфейсу FuzzySystem, всі які перевизначаються в похідних класах:

- метод addInputVariables() – додати вхідну змінну;
- метод addOuputVariables() – додати вихідну змінну;
- метод addRuleBlock() – додати блок нечітких правил;
- метод run() – запустити систему на виконання.

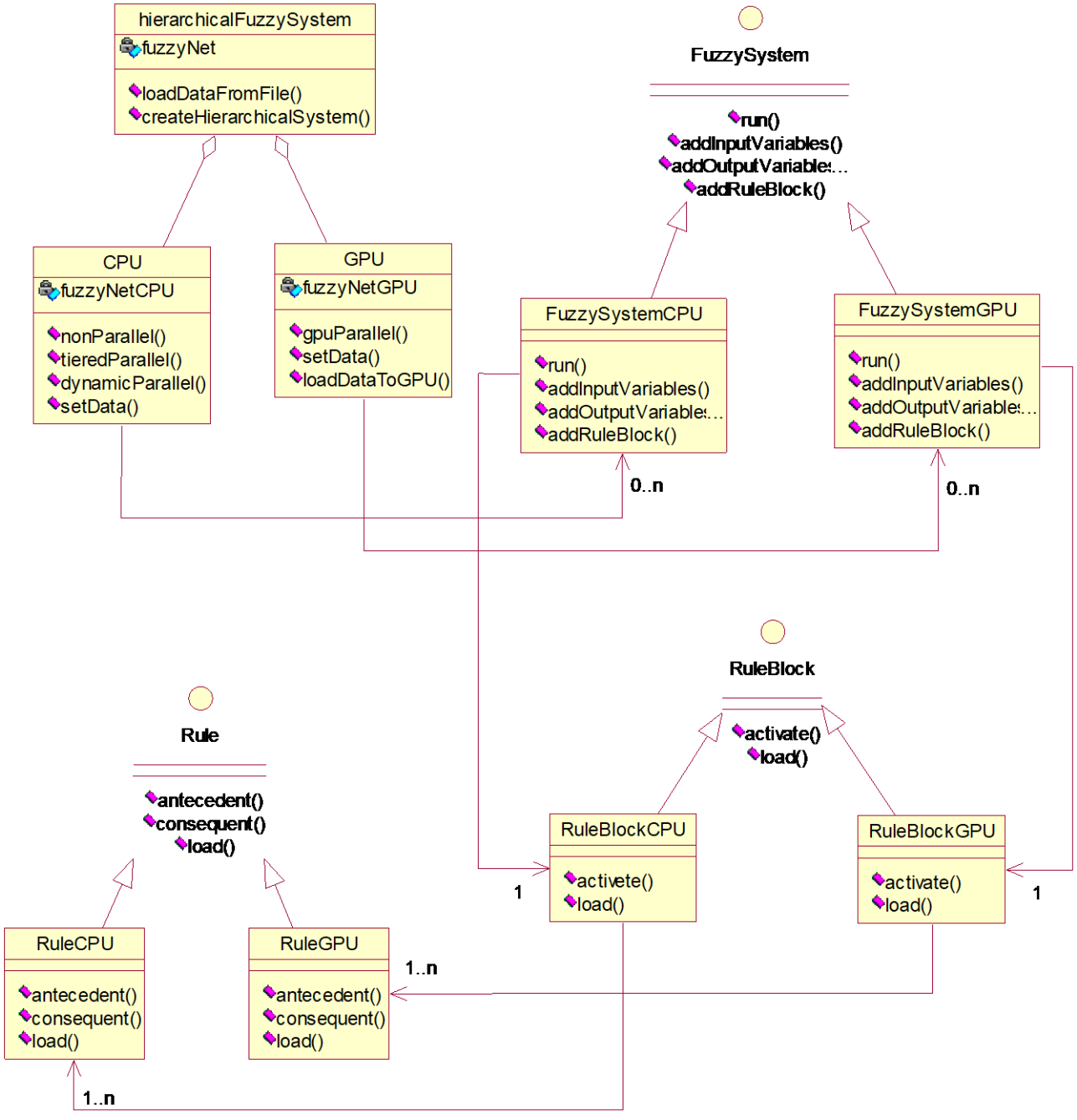

Рис. 5. Діаграма класів для системи ієрархічного нечіткого логічного виведення

Аналогічно розглянемо класи RuleBlock (моделює блок нечітких правил) та Rule (моделює об'єкт нечіткого правила), які також є інтерфейсами, з можливістю перевизначення всіх методів. Методи класу RuleBlock наступні:

- метод load() – задає завантаження блоку нечітких правил;
- метод () – визначає активацію всієї множини правил блоку.

Методи класу Rule:

- метод load() – надає завантаження нечіткого правила;
- метод antecedent() – забезпечує оцінювання лівої частини нечіткого правила (антецедента);
- метод consequent() – застосовує праву частини правила (консеквента).

Зазначимо, що широке використання поліморфізму уніфікує доступ до методів класів та сприяє розширенню програмної системи з мінімальними змінами викликаючих функцій. Існує можливість додавання як нових моделей паралельного ієрархічного нечіткого логічного виведення, так і нових апаратних технологій (шляхом розширення класу hierarchicalFuzzySystem).

Контейнеризація застосунків

При проектуванні GPU-застосунків існує проблема, що пов'язана зі складністю їх подальшого розгортання, тобто налагодження супутніх програм, драйверів, бібліотек, системних залежностей тощо. Особливо це стосується випадків, коли програмне чи апаратне забезпечення відрізняється від забезпечення, що використовувалося при проектуванні, у випадку конфлікту версій. Для подолання цієї проблеми було розроблено метод контейнеризації GPU-застосунків для швидкого їх розгортання на обраній програмній платформі для паралельних обчислень [13]. Іншими словами, «запакувавши» додаток у контейнер, можна у подальшому запускати такий застосунок як на локальному хості, так і на хмарних платформах, які підтримують технологію контейнеризації. Слід зазначити, що на сьогоднішній час майже всі основні хмарні платформи, такі як Microfoft Azure, IBM Cloud, Google Cloud Platform, Oracle Cloud Infrastruction тощо підтримують одночасно контейнеризацію та обчислення на GPU.

Контейнерізація GPU-прискореної інтелектуальної системи оцінювання якості стартапів в середовищі Docker. Для здійснення контейнеризації нечітких систем було встановлено програмне середовище Docker. Docker-застосунок складається з наступних основних компонент: образи (ще називають зображення) та контейнери. Контейнери складаються з одного або декількох образів, та служать для запуску застосунків. Образи можна як створити власноруч (наприклад, за допомогою програми Dockerfile), так і взяти з Docker-реєстру [14], або взяти за основу будь-які образи, створені іншими розробниками. Образи створюються тільки для читання, властивість запису в них відсутня. Як правило, при створенні образів, беруться за основу образи вже створені (наприклад, з репозиторію), та вже на основі них методом накладання шарів створюють інші образи, відмінні від основного (рис. 6). Такий підхід дозволяє по-перше, користуватися вже готовими програмними рішеннями, а по-друге, створювати нове програмне забезпечення з мінімальними витратами часу.

Рис. 6. Графічне зображення шарів при контейнеризації застосунку

При роботі з програмою Docker з підтримкою Nvidia CUDA, всі команди передуються ключовим словом nvidia-docker: nvidia-docker [КОМАНДА].

Наведемо приклад створення образу для інетелектуальної системи оцінки якості стартапів:

```
ARG VERSION_ID
FROM ubuntu:${VERSION_ID}
LABEL "STARTUPS"
RUN apt-get update && apt-get install -y --no-install-recommends \
      dh-make \
```

```
        fakeroot \
        build-essential \
        devscripts \
        lsb-release && \
    rm -rf /var/lib/apt/lists/*
……
COPY fuzzy_system ./ fuzzy_system
WORKDIR ./ fuzzy_system
RUN nvcc sturtups.cu
```

Завершальна стадія створення образу була визначена командою build:

```
nvidia-docker build -t startups .
```

Команда запуску контейнеру Docker є run: nvidia-docker run [ОПЦІЇ] ОБРАЗ [КОМАНДА] [АРГУМЕНТИ].

Для запуску контейнеризованої інтелектуальної системи оцінки якості стартапів застосована наступна команда:

```
nvidia-docker run --rm -ti startups
```

Як бачимо з наведеного вище прикладу, інтерактивно запускається контейнер з ім'ям startups, з видаленням після закінчення роботи. Також до основних команд можна віднести наступні команди: images (перегляд репозиторію), rmi (видалення образу), rm (видалення контейнеру), stop (зупинити роботу контейнера), ps (відобразити список працюючих контейнерів) та ін. Повний список команд та їх опис можна отримати в [15].

Архітектура мікросервісів як основа контейнеризації інтелектуальної системи для нечіткого виведення. Поява архітектури мікросервісів пов'язана з розв'язанням проблем супроводження та масштабування додатків. При проектуванні монолітних програм (декомпозиція системи проводиться на рівні класів, компонентів, просторів імен, тощо) внесення змін у будь-яку частину будь-якого елемента вимагає перезбирання всієї системи заново, навіть у тому випадку, коли зміни не стосуються власне модульних інтерфейсів. Використання бібліотек також не вирішує проблему, оскільки програмний модуль збирається разом з бібліотеками, і внесення змін в бібліотеку також призводить до необхідності повторного збирання застосунку. Масштабування ж монолітних додатків, в тому числі на основі CUDA вимагає створення нової версії продукту.

Успішно вирішує проблеми масштабування та супроводження організація архітектури програмної системи у вигляді сукупності мікросервісів, кожен з яких реалізує роботу окремої бізнес-задачі. Мікросервіси можуть взаємодіяти на різних машинах, різних програмно-апаратних платформах. Модифікація одного модулю (сервісу) не потребує жодних змін інших компонентів системи, модулі можуть бути запроектовані різними інструментальними засобами, різними мовами. Кожний сервіс може працювати окремо від інших та виконує окрему функцію системи. Використання хмарних технологій дозволяє поставити компонент 1 раз, завантаживши його на сервер.

Як мікросервіси платформи Docker виступають контейнери. Контейнер фактично є «чорною скринькою» з власним інтерфейсом. Узагальнюючи, мікросервісну Docker архітектуру можна вважати як таку, в якій реалізовано патерн проектування низького рівня Фасад (Facade) [12], основна задача якого полягає в забезпеченні єдиного інтерфейсу для підсистеми, забезпечуючи єдину вхідну точку (точку доступу до функціоналу) та ізоляцію змін в межах підсистеми.

В цілому можна стверджувати, що контейнеризація програмних систем ІНЛВ забезпечує високу ступінь гнучкості та досить високу простоту розгортання застосунків. Архітектура на основі мікросервісів створює можливість модифікації окремих модулів та механізмів нечіткого виведення окремо від всієї системи ІНЛВ. Необхідно ще зазначити такий момент при використанні контейнеризації, пов'язаний з продуктивністю контейнеризованих програм. Будь-який контейнер можна розгорнути на хмарних сервісах, що підтримують GPU-обчислення, та отримати в розпорядження потужні графічні прискорювачі останнього покоління. Таким чином, проектування програм для GPU на основі контейнеризації є ефективним з точки зору гнучкості розгортання, зручності модифікації та супроводження, а також дозволяє значно підвищити прискорення контейнерів за рахунок хмарних платформ з потужною підтримкою GPU-прискорених контейнеризованих програм.

Висновки

Таким чином, розроблено програмну архітектуру для здійснення ієрархічного нечіткого логічного виведення. Зазначена архітектура надає в межах єдиного програмного комплексу цілу низку алгоритмічних схем паралельних обчислень для нечіткого логічного виведення на основі високопродуктивних програмно-апаратних технологій. Представлено архітектурні рішення, які застосовані в інтелектуальній програмній системі для оцінювання привабливості стартапів, та забезпечують гнучкість та продуктивність нечіткого виведення, що реалізовані на основі архітектурних шаблонів проектування та патернів низькорівневого проектування. Продемонстровано методи та підходи до побудови ієрархічних систем нечіткого логічного виведення у їх архітектурному контексті, а також контексті паралельного програмування. Представлено метод контейнеризації застосунків для GPU для їх повторного використання та розгортання на обраній програмній платформі високопродуктивних паралельних обчислень.

Література

1. Bass L., Clements P., Kazman R. Software architecture in practice. Addison Wesley Professional. 2012. 640 с.
2. Єршов С.В., Пономаренко Р.М. Паралельні моделі багаторівневих нечітких систем Такагі-Сугено. *Проблеми програмування*. 2016. № 1. С. 141–149.
3. Єршов С.В., Пономаренко Р.М. Ярусно-паралельна модель обчислень для логічного виведення у нечітких багаторівневих системах. *Комп'ютерна математика.* 2016. № 1. С. 28–36.
4. Єршов С.В., Пономаренко Р.М. Метод побудови паралельних систем нечіткого логічного виведення на основі графічних прискорювачів. *Проблеми програмування*. 2017. № 4. С. 3–15.
5. Пономаренко Р.М. Моделі паралельних ієрархічних систем для нечіткого логічного виведення. *Комп'ютерна математика*. 2017. № 2. С. 28–36.
6. Ершов С.В. Модель интеллектуальных агентов, основанная на нечеткой логике высшего типа. *Компьютерная математика*. 2012. № 1. С. 10–16.
7. Ершов С.В. Принципы построения нечетких мультиагентных систем в распределенной среде. *Компьютерная математика*. 2009. № 2. С. 54–61.
8. Суперкомпьютеры ИК НАН Украины. http:// /icybcluster.org.ua.
9. Timothy G. Mattson, Beverly Sanders, Berna Massingill. A pattern language for parallel programming. Addison-Wesley Professional, 2004. 534 с.
10. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.
11. Miner D., Shook A. MapReduce Design Patterns. O'Reilly Media, 2012. 251 с.
12. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2015. 368 с.
13. Контейнер Docker с поддержкой GPU. Nvidia. http://www.nvidia.com.ua/object/docker-container-ru.html.
14. Docker Hub. https://hub.docker.com.
15. Wagner B., Rousos M. .NET Microservices: Architecture for Containerized .NET Applications. Microsoft Corporation, 2018. 302 с.
16. Документація платформи Докер. https://docs.docker.com.

References

1. Bass L., Clements P., Kazman R. (2004) Software architecture in practice. Addison Wesley Professional. 452 p.
2. Yershov S.V., Ponomarenko R.M. (2016) Parallel models multilevel fuzzy Takagi-Sugeno systems. Problems of programming. N 1. P. 141–149.
3. Yershov S.V., Ponomarenko R.M. (2016) Tiered parallel calculating model for inference in fuzzy multilevel systems. Computer mathematics, N 1, K.: Institute of Cybernetics Glushkov National Academy of Sciences of Ukraine. P. 28–36.
4. Yershov S.V., Ponomarenko R.M. (2017) Method of constructing parallel fuzzy inference systems based on graphical accelerator. Problems of programming, N 4. P. 3–15.
5. Ponomarenko R.M. (2017) Models of parallel hierarchical systems for fuzzy inference. Computer mathematics, N 2, K.: Institute of Cybernetics Glushkov National Academy of Sciences of Ukraine. P. 28–36.
6. Yershov S.V. (2012) Model of intelligent agents based on highest type fuzzy logic. Computer mathematics, N 1, K.: Institute of Cybernetics Glushkov National Academy of Sciences of Ukraine. P. 10–16.
7. Yershov S.V. (2009) Principles of construction of fuzzy multi-agent systems in a distributed environment. Computer mathematics, N 2, K.: Institute of Cybernetics Glushkov National Academy of Sciences of Ukraine. P. 54–61.
8. Supercomputers of IC NAS of Ukraine. http://icybcluster.org.ua.
9. Timothy G. Mattson, Beverly Sanders, Berna Massingill. (2004) A pattern language for parallel programming. Addison-Wesley Professional. 534 p.
10. Voevodin V.V., Voevodin Vl.V. (2002) Parallel computing. SPb.: BHV-Petersburg. 608 p.
11. Miner D., Shook A. (2012) MapReduce Design Patterns. O'Reilly Media. 251 p.
12. Gamma E., Helm R., Johnson R., Vlissides J. (2015) Methods of object-oriented design. Design patterns. – SPb.: Peter. 368 p.
13. Docker container with GPU support. Nvidia. http://www.nvidia.com.ua/object/docker-container-ru.html.
14. Docker Hub. https://hub.docker.com.

15. Wagner B., Rousos M. (2018) .NET Microservices: Architecture for Containerized .NET Applications. Microsoft Corporation. 302 p.
16. Docker documentation. https://docs.docker.com.

Про авторів:

Єршов Сергій Володимирович,
доктор фізико-математичних наук,
завідувач відділу інтелектуальних програмних систем
Інституту кібернетики імені В.М. Глушкова НАН України.
Кількість наукових публікацій в українських виданнях – 69.
Кількість наукових публікацій в зарубіжних виданнях – 10.
http://orcid.org/0000-0002-9895-777X

Пономаренко Роман Миколайович,
аспірант
Інституту кібернетики імені В.М. Глушкова НАН України,
Кількість наукових публікацій в українських виданнях – 8.
http://orcid.org/0000-0001-9681-2297

Місце роботи авторів:

Інститут кібернетики імені В.М. Глушкова НАН України,
проспект Академіка Глушкова, 40,
Київ, 03187, Україна.
Тел.: (044) 526 64 22,
(044)526 5168.

E-mail: sershv@ukr.net,
ponomarenko_roman@ukr.net