УДК 004.424

*І.В. Оконський, А.Ю. Дорошенко, К.А. Жереб*

# ІНСТРУМЕНТАЛЬНІ ЗАСОБИ МОДЕЛЮВАННЯ ГЕТЕРОГЕННИХ СЕРЕДОВИЩ ЗАСНОВАНИХ НА ВІДЕОГРАФІЧНИХ ПРИСКОРЮВАЧАХ

Запропоновані архітектура гнучкого та розширюваного інструментарію моделювання гетерогенних Грід-систем з відеографічними прискорювачами gpusim на базі Java-фреймворку GridSim та розроблений на її основі прототип. Проведена перевірка адекватності та первинне дослідження моделі на прикладі конкретної задачі для симуляції на розробленому прототипі.

## Вступ

В даний час Грід-системи все більше використовуються як в наукових, так і в промислових застосуваннях. Вони дозволяють ефективно використовувати існуючі різнорідні паралельні ресурси та вирішувати задачі великих обсягів, які не можуть бути вирішені на окремих паралельних комплексах. Проте Грід-системи є складними як для створення та конфігурування, так і для програмування. При цьому підбір найбільш оптимальних конфігурацій як для структури самого Грід-середовища, так і для задач, що виконуються на ньому, недоцільно здійснювати на реальних системах через великі витрати та необхідність координації різних учасників Грід-проектів. Тому актуальною є задача моделювання, яка дозволяє без витрат на створення, підтримку та використання реальної Грід-системи провести на моделі необхідні експерименти, результати яких не будуть суттєво відрізнятися від оригіналу.

Також нині активно зростає інтерес до використання для обчислень графічних прискорювачів (відеоадаптерів, GPU) через їх високу продуктивність у порівнянні зі звичайними процесорами (CPU), доступності, а також розвитку відповідного інструментарію розробника. Зокрема, актуальними є задачі моделювання паралельних систем (кластерів, Грід-систем), окремі вузли яких мають гетерогенний характер і містять як CPU, так і GPU-компоненти.

У даній роботі запропонована архітектура гнучкого і розширюваного середовища моделювання гетерогенних Грід-систем gpusim на базі Java-фреймворку GridSim. Особливістю системи є обраний підхід до опису гетерогенної паралельної системи, при якому CPU і GPU-компоненти розглядаються як окремі вузли віртуального Грід-середовища. Описано прототип інструментальної системи для моделювання, заснований на використанні запропонованої архітектури. Описано механізм налаштування системи на параметри конкретної паралельної системи за рахунок автоматизованого підбору параметрів моделі. Експерименти показують достатню точність побудованої моделі для великих розмірів вхідних даних.

Матеріал даної роботи організований наступним чином. У розділі 1 наведено огляд існуючих рішень і пояснення необхідності розробки gpusim. У розділі 2 поставлені задачі, які має вирішувати gpusim і наведена її архітектура. У розділах 3 та 4 обрана конкретна задача (блочний алгоритм множення матриць), виконання якої буде симулюватись на моделі, та описаний експериментальний модуль, який реалізує необхідну функціональність. Роботу завершують висновки та напрямки подальшої роботи.

## 1. Огляд існуючих рішень

На сьогоднішній день активно ведуться дослідження у напрямі моделювання Грід-систем, розроблено досить багато

інструментальних засобів. Розробляються як аналітичні, так і імітаційні моделі Грід-систем, при цьому розглядаються різні аспекти їх функціонування.

Аналітичні моделі дозволяють отримати загальні теоретичні результати для широкого діапазону можливих параметрів Грід-середовища, хоча вони абстрагуються від багатьох реальних деталей функціонування. Зокрема, в [1] описано модель продуктивності Грід-системи, що складається з окремих компонентів, продуктивність яких вважається відомою. Аналітична модель дозволяє обчислити стаціонарний стан системи для заданого вхідного потоку задач та параметрів окремих вузлів. Аналогічний підхід запропоновано в [2]. Описується мережева взаємодія між обчислювальними вузлами, при цьому враховується різнорідність вузлів та з'єднань. Модель розглядає два рівні взаємодії: між Грід-вузлами, які є кластерами, та всередині одного кластера. В роботі [3] моделюється обчислювальна мережа, що містить як процесори загального призначення (CPU), так і спеціалізовані реконфігуровані процесори (FPGA). Аналітична модель описує структуру окремих вузлів: чистих CPU- або FPGA-вузлів, а також гібридних вузлів, що одночасно містять CPU та FPGA. При цьому моделюються обчислювачі та ієрархія пам'яті в такому вузлі.

Також був запропонований підхід моделювання Грід на основі апарата мереж Петрі [4]. Даний підхід та його інструментарій дозволяє детально розглянути внутрішні процеси і взаємодії ключових вузлів Грід, однак є складним у налаштуванні та завданні вхідних даних, і не підходить для Грід зі складною або великою внутрішньою структурою.

Для більш точних вимірів продуктивності Грід-систем при конкретних значеннях параметрів використовуються імітаційні моделі. Вони дозволяють отримати довільну точність моделювання за рахунок врахування всіх необхідних деталей. Але на практиці слід досягати балансу між точністю моделювання та обчислювальною складністю.

Серед існуючих напрацювань слід виділити імітаційну модель, розроблену для дослідження ефективності методів планування ресурсів для різних інтенсивностей потоків завдань в Грід [5] і її програмну реалізацію GRID_Scheduler_Model. Моделюється гомогенна Грід-система, і для підтримки гетерогенних Грід модель потребує доопрацювання, а розширюваність у реалізації не передбачена.

На даний момент існує кілька інструментаріїв (фреймворків), які дозволяють розробнику на їх основі створити свою модель Грід і проводити на ній експерименти. Огляд і порівняльний аналіз найбільш функціональних та стабільних фреймворків MicroGrid, OptorSim, SimGrid, GridSim, Bricks наведено в [6, 7]. На підставі порівняння можна зробити висновок, що найбільш зручним і функціональним є GridSim [8], фреймворк для мови програмування Java, який в якості симуляційного ядра використовує бібліотеку SimJava [9].

Проблеми моделювання часу виконання задач на GPU також розглянуті в літературі. Зокрема запропоновано детальну аналітичну модель [10], емпіричну модель часу виконання та споживання енергії [11], а також модель порівняння продуктивності CPU та GPU [12]. Роботи [11, 12] використовують емулятор Ocelot [13] для врахування деталей виконання задач на GPU. Такі дослідження дозволяють досягти великої точності моделювання, але потребують значних ресурсів для виконання моделі. Також вони не передбачають вбудовування в моделі Грід-середовища. Дана робота продовжує дослідження, розпочаті в [14], з побудови досить простої моделі виконання задач на GPU, яка дозволила б отримати точність достатню для прийняття важливих рішень з розподілу навантаження в Грід-середовищі.

## 2. Опис розробленої системи – gpusim

**2.1. Опис основних принципів роботи з фреймворком GridSim.** Фреймворк GridSim для завдання моделі Грід використовує ряд сутностей [9], найбільш важливими серед яких є:

- користувач Грід (GridUser), нащадок класу GridSim – який задає сценарій симуляції;
- ресурс Грід (GridResource) – агрегує структуру і властивості певного ресурсу (вузла Грід-мережі). Структуру ресурсу визначає список ЕОМ (MachineList), пропускна здатність з'єднання між ЕОМ, а також абстрактна вартість використання ресурсу в секунду. Кожна ЕОМ складається з однакових обчислювальних елементів (PE), і характеризується кількістю PE і їх рейтингом у MIPS;
- завдання (Gridlet) – описує обчислювальну задачу, виконувану на PE. Параметрами Gridlet є розмір вхідних і вихідних даних в байтах та обсяг робіт, необхідний для виконання (Length) в MI (million instructions).

Паралельна система з CPU + GPU за допомогою сутностей GridSim може бути представлена у вигляді ресурсу з двома ЕОМ: перша ЕОМ визначає кількість ядер CPU та їх рейтинг, друга – кількість ядер GPU та їх рейтинг. Ядра CPU та GPU моделюються у вигляді обчислювальних елементів (PE). Моделювання CPU і GPU-компонентів з використанням базових сутностей GridSim дозволить надалі спростити включення розробленої моделі в більш складні конфігурації гетерогенних систем, зокрема кластери і Грід-системи з GPU-вузлами.

Для проведення симуляції необхідно ініціалізувати GridSim, створити і налаштувати сутності ресурсів і завдань. Після старту симуляції необхідно в методі run() об'єкта-нащадка GridSim виконати необхідний сценарій розподілу завдань між ресурсами. У процесі виконання сценарію GridSim надає користувачу можливість збору та обробки поточних статистичних даних про час виконання кожного із завдань, завантаженості ресурсів у певний момент часу тощо. Візуалізація результатів у системі GridSim не передбачена і має забезпечуватись зовнішніми компонентами.

**2.2. Задачі gpusim.** Враховуючи вищевикладене, прийнято рішення на основі Java-фреймворку GridSim розробити гнучке і розширюване середовище для моделювання Грід. Система gpusim має надавати можливість максимально гнучко задавати структуру як Грід-середовища (кількість вузлів, характер з'єднань, наявність спеціалізованих вузлів, у тому числі таких, що містять GPU), так і навантаження (кількість завдань різних типів, можливість їх виконання на різних вузлах, розміри вхідних даних). Також система моделювання дозволяє проводити дослідження заданої Грід-конфігурації змінюючи її параметри.

Однак чим більш універсальним є інструмент, тим складніше він у використанні для конкретних завдань. Враховуючи це, основною метою, крім гнучкості та розширюваності, для gpusim є простота використання кінцевим користувачем та наочність відображення результатів симуляцій для конкретних завдань.

**2.3. Опис архітектури gpusim.** Для досягнення розширюваності та гнучкості і, разом з тим простоти, прийнято рішення розділити систему на кілька компонентів (рис. 1).

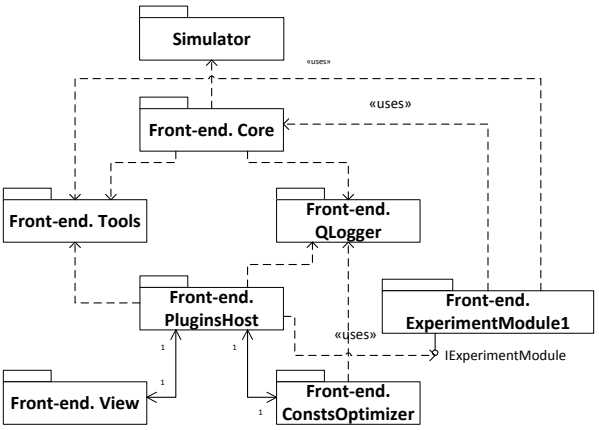


Рис. 1. Діаграма компонентів gpusim

**Симулятор gpusim** – оболонка на мові програмування Java над фреймворком GridSim, основною задачею якої є налаштування GridSim на основі даних конфігураційного файлу, запуск симуляції, збір і вивід статистики у файл.

**Front-end компонент** – написаний на мові програмування C++ із використанням фреймворку Qt версії 4.8.3 [15]. Включає до себе модуль логування QLogger, інструментальне ядро, виконавче ядро, модуль оптимізації констант, сервер плагінів і набір експериментальних модулів.

**Експериментальним модулем** є плагін для front-end, який інкапсулює у собі всі засоби для роботи з конкретною задачею: надає графічний інтерфейс користувача для введення/виведення вхідних/вихідних даних, генератор конфігурації для симулятора, а також обробник статистики. Експериментальний модуль має ряд попередньо встановлених параметрів, які не є специфічними для окремих експериментів, проте мають суттєвий вплив на результати симуляції (ці параметри описані у розділі 3.2). На даний момент розроблений один експериментальний модуль MatrixMultiply, що надає можливість дослідження задачі множення матриць за допомогою блочного алгоритму на моделі паралельної системи, що складається з CPU і GPU.

**Сервер плагінів** – віртуальний сервер, який вирішує завдання завантаження і виконання плагінів у front-end компоненті.

**Інструментальне ядро** – набір допоміжного функціонала для спрощення роботи з даними та іншими модулями.

**Модуль логування QLogger** – інкапсулює функції логування та надає користувачу модуля зручний інтерфейс для запису повідомлень до лог-файлу та до консолі. Використовує як основу механізм налагоджувальних повідомлень Qt-модуля QDebug.

**Виконавче ядро** – модуль, основна задача якого – надання іншим модулям функціонала для роботи з симулятором, його налаштування та обробки статистики.

**Модуль оптимізації констант** призначений для підвищення точності симуляції шляхом автоматизації проведення серій симуляцій з різними встановленими параметрами одного з експериментальних модулів. Даний модуль, на основі конфігураційного файлу, що містить імена параметрів, їх початкові, кінцеві значення, а також величину і спосіб інкрементування кожного з параметрів генерує набір можливих поєднань значень параметрів, потім послідовно запускає симуляцію з кожним із наборів параметрів і порівнює результати симуляції з результатами експерименту, проведеного на реальній паралельній системі. Якщо результат останньої проведеної симуляції має меншу розбіжність з результатами експерименту на реальній системі – то набір попередньо встановлених параметрів, відповідний останній симуляції, вважається найкращим.

**2.4. Взаємодія модулів gpusim.** На рис. 2 показана діаграма послідовностей для найбільш частого варіанта використання gpusim – проведення одного експерименту.

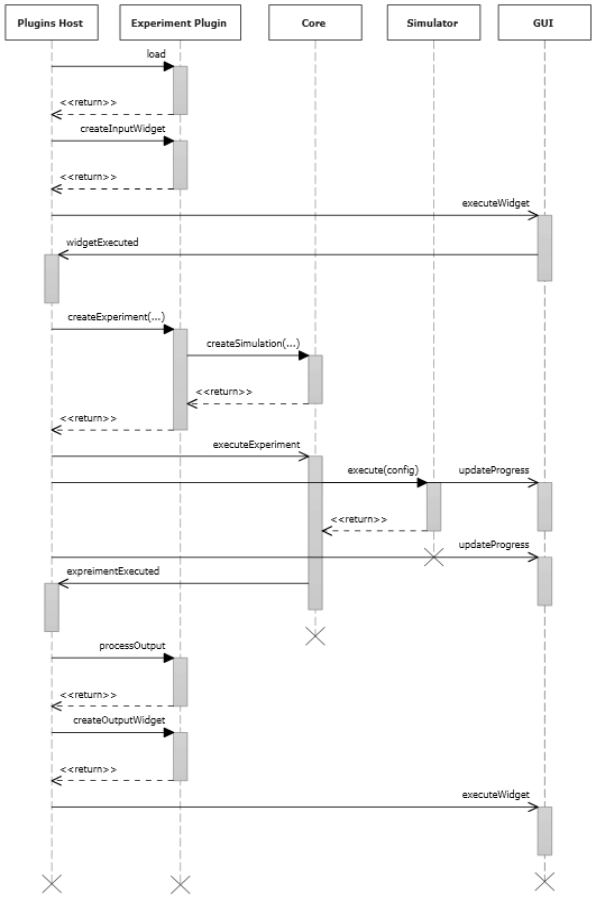


Рис. 2. Діаграма послідовностей проведення експерименту в gpusim

Експериментом gpusim є набір симуляцій, кожна з яких включає конфігурацію для симулятора і статистику, отриману в результаті його роботи.

Спочатку сервер плагінів завантажує необхідний експериментальний модуль, потім запитує в нього віджет, який дозволяє кінцевому користувачу задати вхідні дані для експерименту. Після цього сервер плагінів запитує у експериментального модуля експеримент, створений на основі даних, введених кінцевим користу-

вачем. Експериментальний модуль частину роботи делегує виконавчому ядру. Потім створений експеримент передається через виконавчий модуль на виконання симулятору. Виконавче ядро асинхронно передає дані про прогрес експерименту, які потім відображаються в графічному інтерфейсі.

Після закінчення роботи симулятор формує файл вихідних даних, який читається виконавчим ядром і передається через сервер плагінів на обробку до експериментального модуля.

Після обробки вихідних даних у експериментального модуля запитується віджет, що відображає вихідні дані, і посилається на виконання графічному інтерфейсу.

## 3. Моделювання задачі множення матриць

**3.1. Задача множення матриць блочним алгоритмом.** Для перевірки точності та актуальності розробленого середовища моделювання гетерогенних Грід-систем обрана задача множення матриць за допомогою блочного алгоритму [14].

Результатом множення матриці $A$ розмірністю $m \times n$ $^{m \times n}$ й матриці $B$ розмірності $n \times l$ $^{n \times l}$ є матриця $C$ розмірності $m \times l$, кожен елемент якої обчислюється наступним чином:

$$c_{ij} = \sum_{k=0}^{n-1} a_{ik} b_{kj}\ ,\ i = 0,..,m\ ,\ \ j = 0,..,l\ .$$

Цей алгоритм вимагає $m \cdot n \cdot l$ операцій множення й додавання елементів матриць. При множенні квадратних матриць розмірності $n \times n$ кількість виконаних операцій має порядок $O(n^3)$. Оскільки для обчислення одного елементу результуючої матриці потрібні тільки відповідні рядок та стовпець матриць, що множаться, цей алгоритм природно розділяється на незалежні потоки обчислень.

Якщо ми множимо дві квадратні матриці розмірністю $n \times n$, то результуюча матриця $C$ буде тієї самої розмірності, що й вхідні матриці $A$ і $B$. Спочатку матриця $C$ розбивається на прямокутні блоки розмірністю $k \times p$. Кожен з цих блоків обчислюється незалежно окремою задачею, для чого потрібно передати блок матриці $A$ розмірністю $n \times k$ й відповідний блок матриці $B$ розмірністю $p \times n$.

Перевагою такого алгоритму є те, що задачу можна розбивати на довільну кількість підзадач, а не тільки на квадратні блоки. Також відсутність будь-яких залежностей між підзадачами дозволяє ефективно виконувати обчислення у випадку коли кількість наявних процесорів значно менша за кількість підблоків – у такому випадку підзадачі отримують нові блоки «за готовністю». Недоліком є додаткові витрати пам’яті, які виникають при частковому дублюванні вхідних даних для різних підзадач. Загальна кількість операцій не відрізняється від послідовного алгоритму й має порядок $O(n^3)$.

В роботі [14] також побудовано емпіричну модель часу виконання блочного алгоритму на GPU. Ця модель має наступний характер: $T_{gpu} = N^2 T_{st.global} + N^3 T_{ld.global}$, де $T_{ld.global}$ та $T_{st.global}$ – параметри, що визначають час завантаження та збереження даних при роботі з глобальною пам'яттю GPU. Ці параметри є частиною опису моделі й мають підбиратись для конкретних обчислювальних вузлів експериментальним шляхом.

**3.2. Експериментальний модуль множення матриць блочним алгоритмом.** На основі емпіричної моделі часу виконання задачі множення матриць на GPU [14] були розроблені генератор експериментів симулятора і обробник статистики, що входять до складу експериментального модуля MatrixMultiply. Вхідними параметрами генератора є розмір блоку, мінімальний і максимальний розмір матриці, а також інкремент розміру матриці. Оброблювач статистики отримує вихідні дані симулятора, перетворює їх для наочного відображення кінцевому користувачу, а також порівнює час кожної з симуляцій з результатами експерименту на реальній паралельній системі.

Генератор має ряд попередньо встановлених параметрів, що не відносяться безпосередньо до експерименту, але впливають на час симуляції:

- кількість обчислювальних елементів CPU і GPU, а також їх рейтинги в одиницях MIPS (million instructions per second): cpuMachinePECount, cpuMachinePERating, gpuMachinePECount, gpuMachinePERating;
- пропускна здатність підключення між ресурсами і розподільниками завдань: resourceBaudRate, linkBaudRate;
- вартість використання ресурсів Грід – resourceCostPerSec;
- вартість операцій роботи з пам'яттю: завантаження та збереження даних: loadOperationCost, saveOperationCost.

Дані параметри залежать від внутрішньої структури Грід-системи, відповідної моделі, і для проведення подальших досліджень з моделлю, необхідно знайти їх точні значення за допомогою модуля оптимізації констант.

## 4. Перевірка адекватності моделі та аналіз отриманих результатів

**4.1. Перевірка адекватності моделі.** Для перевірки адекватності створеної моделі проведений експеримент, описаний в [14] на реальній паралельній системі, що складається з CPU Intel Core i7 3770k (4 Core, HT, 3.5 GHz, Smart Cache 8MB)[16] і GPU GeForce 650 GTX (384 CUDA Cores, 1058 MHz, 86,4 GFLOPS FP64, 1024 MB) [17].

Результатом експерименту є залежність часу виконання множення матриць блочним алгоритмом з урахуванням пересилки даних від розміру матриці. Через обмеження, що накладаються операційною системою Windows на час завантаженості GPU, отримані результати для діапазону розміру матриць [16; 3760] із інкрементом 16.

Для підстроювання попередньо встановлених параметрів генератора експерименту множення матриць блочним алгоритмом використовувався модуль оптимізації констант. Для перевірки адекватності моделі і точності підбору встановлених параметрів, модуль оптимізатора налаштований на роботу в діапазоні розмірів матриць [512; 1552] із інкрементом 80.

Спочатку отримані значення попередньо встановлених параметрів у першому наближенні. Значення кожного з параметрів змінювалося на порядок, при цьому значення решти параметрів залишалися без змін:

- cpuMachinePECount: [1; 1000], величина інкремента – один порядок, краще значення – 10;
- cpuMachinePERating: [1; 100000], величина інкремента – один порядок, краще значення – 1000;
- gpuMachinePECount: [64; 1024], величина інкремента – 64, краще значення – 384;
- gpuMachinePERating: [1; 100000], величина інкремента – один порядок, краще значення – 10000;
- resourceBaudRate: [1e+01; 1e+13], величина інкремента – два порядки, краще значення – 1e05;
- resourceCostPerSec: [1; 1000], величина інкремента – один порядок, краще значення – 1;
- linkBaudRate: [1e+01; 1e+13], величина інкремента – два порядки, краще значення – 1e05;
- loadOperationCost: [1e-05; 1e03], величина інкремента – один порядок, краще значення – 1e-04;
- saveOperationCost: [1e-05; 1e03], величина інкремента – один порядок, краще значення – 1e-03.

В ході експерименту виявлено, що параметри cpuMachinePECount, cpuMachinePERating, resourceCostPerSec практично або зовсім не впливають на час симуляції, це пояснюється тим, що всі обчислення проводяться на GPU, а оскільки Грід-ресурс один, то яка б не була його вартість використання в секунду , розподілити завдання на інший не представляється можливим. Також слід зазначити, що параметри resourceBaudRate і linkBaudRate незначно впливають на час симуляції при значеннях до 1e05. Значення даних параметрів більші за 1e05 не мають впливу на час симуляції, це пояснюється малим обсягом даних, що пересилаються між елементами Грід.

Вищеописаний експеримент повторений для параметрів loadOperationCost і

saveOperationCost із уточненим діапазоном: [1e-04; 1e-03] та величиною інкремента 1e-04, оскільки вони мають найбільший вплив на час симуляції.

Далі методом повного перебору всіх можливих комбінацій параметрів з кроком 1e-06 (приблизно 1000) в околі оптимальних значень в першому наближенні, отримані точні значення встановлених параметрів. Ці значення відповідають характеристикам реальної паралельної системи, на якій вироблялися обчислення:

- cpuMachinePECount = 8;
- cpuMachinePERating = 1000;
- gpuMachinePECount = 384;
- gpuMachinePERating = 10000;
- resourceBaudRate = 1e+10;
- resourceCostPerSec = 1;
- linkBaudRate = 1e+10;
- loadOperationCost = 1.8e-4;
- saveOperationCost = 1.936e-3.

**4.2. Аналіз отриманих результатів.** На рис. 3 показано графіки часу виконання реального експерименту (пунктирна лінія) і симуляції (суцільна лінія), а також значення відносної похибки симуляції для діапазону розміру матриць [512; 1552] із інкрементом 80 (тобто для тих даних, для яких проводилось налаштування параметрів).

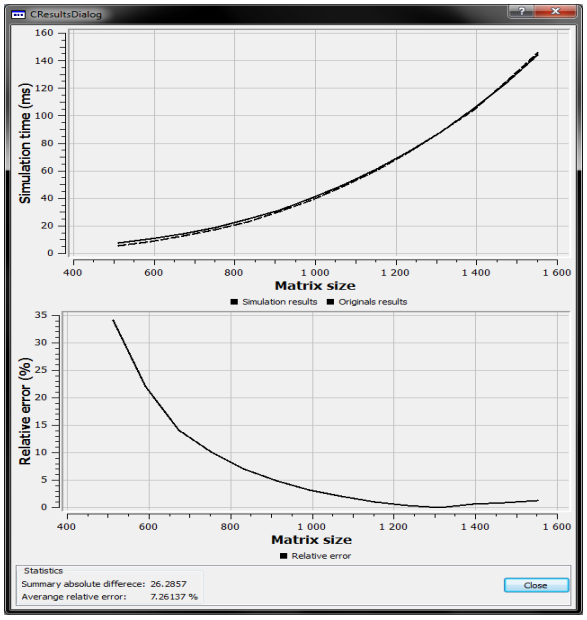


Рис. 3. Порівняння часу виконання реального експерименту і симуляції для діапазону розміру матриці [512; 1552] з інкрементом 80

На рис. 4 показано ті ж графіки для діапазону розміру матриць [512; 3760] із інкрементом 16 (для всіх практично значущих даних, що отримані шляхом виконання на реальному ресурсі).

Як видно на графіках, при великих розмірах матриці (2000–3000 елементів), відносна похибка стабілізується і становить 3–4 %, що є прийнятною точністю для подібних систем.

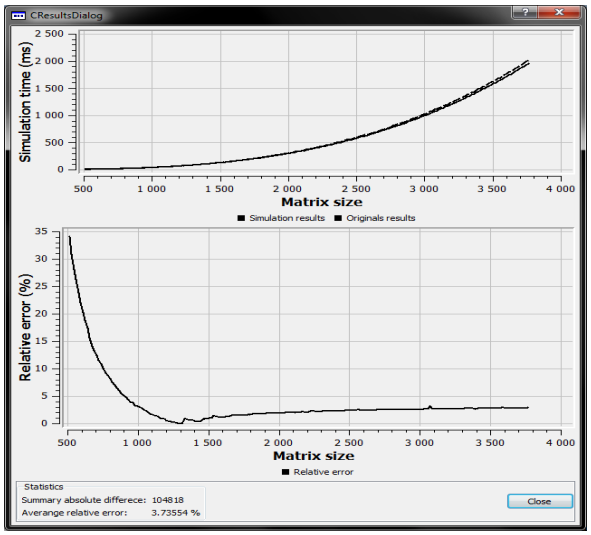


Рис. 4. Порівняння часу виконання реального експерименту і симуляції для діапазону розміру матриці [512; 3760] з інкрементом 16

У ході дослідження моделі виявлені дві особливості. При малих розмірах матриці розбіжність між результатами симуляції і реального експерименту істотно більше ніж при великих (рис. 5). Це пояснюється обмеженням фреймворку GridSim: симуляція не може тривати менше порогового часу 2 мс. Крім того, для малих розмірів матриць складно точно виміряти час виконання на GPU. Також для малих розмірів на час виконання починають впливати додаткові параметри, не описані в моделі. Тому моделі, описані в [14], також не працюють для малих розмірів матриць. Слід зазначити, що використання GPU для множення матриць малих розмірів є невиправданим, через істотні накладні витрати на передачу даних і ініціалізацію. Тому для практично значущих випадків дана розбіжність не спостерігається.

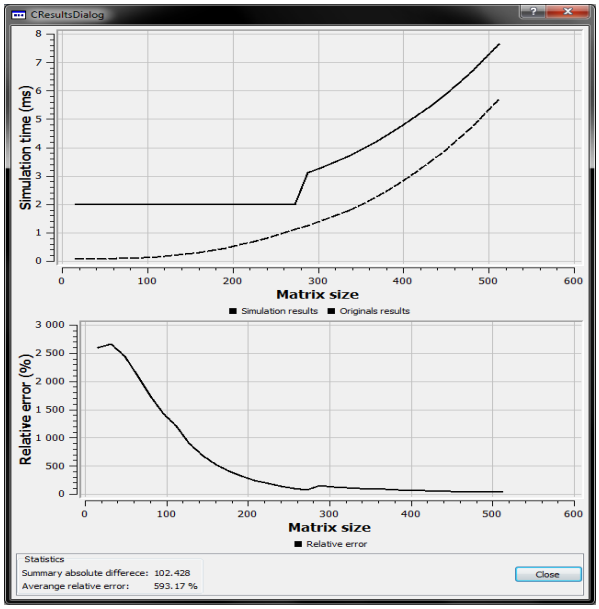


Рис. 5. Порівняння часу виконання реального експерименту і симуляції для діапазону розміру матриці [16; 512] з інкрементом 16

Друга особливість полягає у різкому підвищенні часу симуляції при певних значеннях розміру матриці, при відсутності аналогічного підвищення в експериментальних даних (рис. 6). Цю особливість можна пояснити збільшенням накладних витрат, пов'язаних із використанням додаткового обчислювального елемента (обчислювального ядра) ресурсу Грід.

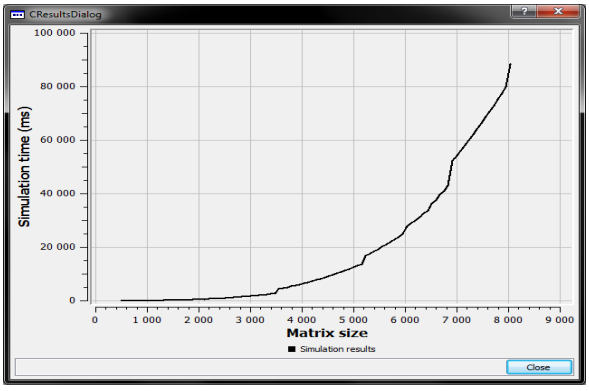


Рис. 6. Різке підвищення часу симуляції при певних значеннях розміру матриці

## Висновки

У роботі запропонована архітектура гнучкого, розширюваного і простого у використанні середовища моделювання гетерогенних Грід-систем на основі симуляційного фреймворку GridSim. На основі запропонованої архітектури розроблений прототип середовища. Проведено експериментальну перевірку розробленого прототипу на задачі множення матриць за рахунок порівняння реальних вимірів та модельного часу виконання. За допомогою модуля оптимізації констант знайдена конфігурація моделі, що має 3–4 % відхилення результатів симуляції щодо результатів експериментів на реальній паралельній системі CPU-GPU.

Розроблена система погано працює для малих розмірів задач, що є недоліком, який планується в майбутньому усунути.

Подальші дослідження в даному напрямку передбачають підвищення точності і розширення функціональності симулятора. Також планується дослідження різних задач, зокрема для визначення можливості підбору параметрів для моделювання заданого обчислювального вузла незалежно від задачі. Додатково ставиться мета включення побудованої імітаційної моделі окремого вузла CPU-GPU в більш велику модель гетерогенного кластера або Грід-системи із підтримкою відеографічних прискорювачів.

1. *Yongwei W., Likun L., Jiayin M., et al.* An analytical model for performance evaluation in a computational grid // Proceedings of the 2007 Asian technology information program's (ATIP's) 3rd workshop on High performance computing in China: solution approaches to impediments for high performance computing (CHINA HPC '07). – 2007. – P. 145–151.
2. *Javadi B., Abawajy J.H., Akbari M.K.,et al.* Analytical Network Modeling of Heterogeneous Large-Scale Cluster Systems // IEEE International Conference on Cluster Computing. Barcelona, September 25–28, 2006. – P. 1–9.
3. *Nadeem M.F., Ahmadi M., Nadeem M., et al.* Modeling and Simulation of Reconfigurable Processors in Grid Networks // Proceedings of the 2010 International Conference on Reconfigurable Computing and FPGAs (RECONFIG '10). – 2007. – P. 226–231.
4. *Шелестов А.Ю.* Подходы и средства моделирования GRID-систем обработки спутниковых данных // Проблеми програмування. – 2008. – № 2–3. – С. 713–720.

5. *Минухин С.В., Знахур С.В.* Исследование эффективности методов планирования ресурсов для различных интенсивностей потоков заданий в Грид // Радіоелектронні і комп'ютерні системи. – 2012. – № 1 (53). – С. 165–171.
6. *Петренко А.І.* Комп'ютерне моделювання грід-систем // Электроника и связь. – 2010. – № 5. – С. 40–48.
7. *Кореньков В.В., Нечаевский А.В.* Пакеты моделирования DATAGRID // Системный анализ в науке и образовании. – 2009. – № 1. – С. 1–15.
8. *Sulistio A., Cibej U., Venugopal S., et al.* A toolkit for modelling and simulating data Grids: an extension to GridSim // Concurrency and Computation: Practice & Experience. – 2008. – Vol. 20, N 13. – P. 1591–1609.
9. *GridSim*: A Grid Simulation Toolkit for Resource Modelling and Application Scheduling for Parallel and Distributed Computing [Электроний ресурс]. – Режим доступу: http://www.buyya.com/gridsim/. – 01.11.2012 p.
10. *Baghsorkhi S.S., Delahaye M., Patel S.J., et al.* An adaptive performance modeling tool for GPU architectures // SIGPLAN Not. – 2010. – Vol. 45, N 5. – P. 105–114.
11. *Hong S., Kim H.* An integrated GPU power and performance model // SIGARCH Comput. Archit. News. – 2010. – Vol. 38, N 3. – P. 280–289.
12. *Kerr A., Diamos G., Yalamanchili S.* Modeling GPU-CPU workloads and systems // Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units. – 2010. – P. 31–42.
13. *Kerr A., Diamos G., Yalamanchili S.* A characterization and analysis of PTX kernels // IEEE International Symposium on Workload Characterization (IISWC 2009). – 2009. – P. 3–12.
14. *Жереб К.А., Ігнатенко О.П.* Моделювання задачі множення матриць для відеографічних прискорювачів // Матеріали конференції "Високопродуктивні обчислення" (HPC-UA 2012). Київ, 08–10 жовтня 2012 р. – С. 174–181.
15. *Qt* Developer Network [Электронний ресурс]. – Режим доступу: http://qt-project.org/. – 01.11.2012 г.
16. *Intel* Core i7-3770k Processor [Электронний ресурс]. – Режим доступу: http://ark.intel.com/products/65523. – 01.11.2012 г.
17. *GeForce* GTX 650 Specifications [Электронний ресурс]. – Режим доступу: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-650/specifications. – 01.11.2012 г.

Одержано 21.06.2012

***Про авторів:***

*Оконський Ілля В'ячеславович,*
студент факультету інформатики
та обчислювальної техніки,
кафедри автоматики і управління
в технічних системах НТУУ "КПІ",

*Дорошенко Анатолій Юхимович,*
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень Інституту
програмних систем НАН України,

*Жереб Костянтин Анатолійович,*
кандидат фізико-математичних наук,
науковий співробітник.

***Місце роботи авторів:***

Національний технічний університет
України "КПІ",
03056, Київ-56,
Проспект Перемоги, 37.
Тел.: (044) 236 7989,

Інститут програмних систем
НАН України,
03680, Київ-187,
Проспект Академіка Глушкова, 40.
Тел.: (044) 526 1538.

e-mail: logrus.work@gmail.com,
dor@isofts.kiev.ua
zhereb@gmail.com