УДК 004.424

*А.Ю. Дорошенко, М.М. Гнинюк*

# Паралельна розподілена реалізація моделювання паралельних обчислень

Запропонована паралельна реалізація для раніше створеного інструментарію моделювання гетерогенних паралельних обчислювальних систем, побудованого на основі фреймворку GridSim. Проведена перевірка та первинне дослідження цієї реалізації на прикладі однієї прикладної задачі.

## Вступ

Сьогодні паралельні обчислювальні 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.

Слід зазначити, що актуальною є проблема моделювання саме гетерогенних Грід-систем, що включають до свого складу центральні процесори (CPU), а також як графічні прискорювачі (відеоадаптери, GPU) через їх високу продуктивність останніх, їх доступність за ціною і енергоефективність. Одним з популярних засобів моделювання паралельних систем (кластерів, Грід-систем), окремі вузли яких мають гетерогенний характер і містять як CPU, так і GPU-компоненти, є фреймворк GridSim [1]. 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.

У роботі [2] була запропонована архітектура гнучкого і розширюваного середовища моделювання гетерогенних Грід-систем gpusim на базі Java-фреймворку GridSim. Особливістю системи є такий підхід до опису гетерогенної паралельної системи, при якому CPU і GPU-компоненти розглядаються як окремі вузли віртуального Грід-середовища. Було описано прототип інструментальної системи для моделювання, заснований на використанні запропонованої архітектури, а також механізм налаштування системи на параметри конкретної паралельної системи за рахунок автоматизованого підбору параметрів моделі. Експерименти показали достатню точність побудованої моделі для великих розмірів вхідних даних та ефективність застосування системи моделювання. В роботі [3] ці дослідження були продовжені на прикладі відомої прикладної задачі гравітаційної взаємодії *N* тіл, де було поставлено більш складне питання про пошук оптимального значення параметру конфігурації обчислювальної системи для отримання більш ефективної програми.

В даній роботі виконано наступний крок у розвитку інструментарію моделювання у напрямку підвищення ефективності цього інструментарію за рахунок створення його паралельної розподіленої реалізації. Матеріал статті організований наступним чином. У розділі 1 розглянуто основні засади імітаційного моделювання, наведено огляд існуючих рішень для моделювання Грід-систем та обгрунтована необхідність розробки інструментарію gpusim. У розділі 2 описана архітектура інструментарію gpusim та його паралельна розподілена реалізація. У розділах 3 та 4 розглядається застосування паралельної версії інструментарію gpusim для моделювання конкретної прикладної задачі (блочний алгоритм множення матриць. Роботу завершують висновки та напрямки подальшої роботи.

## Засоби імітаційного моделювання

Імітаційне моделювання є найбільш потужним, а іноді і єдиним методом дослідження динамічної поведінки складних систем [4]. Імітаційна модель відтворює поведінку модельованої системи в часі, для якого використовуються три категорії часу: фізичного, модельного і процесорного. Фізичний час стосується системи, що моделюється, модельний – відтворення фізичного часу в моделі, а процесорним часом називається час виконання імітаційної моделі на комп'ютері. Співвідношення фізичного і модельного часу визначається специфікою системи і задається величиною діапазону фізичного часу, прийнятого за одиницю модельного часу.

Змістом імітаційного моделювання є просування модельного часу і виконання подій, пов'язаних з певними його значеннями, а основним завданням імітаційного моделювання є правильне відображення порядку подій в системі, що моделюється, на порядок виконання подій в моделі.

Моделювання складних систем може вимагати значних витрат процесорного часу. Тому важливим завданням системи імітаційного моделювання є зменшення процесорного часу, наприклад, за рахунок паралельного виконання подій на мультипроцесорній техніці. Хоча у деяких прикладних галузях, навпаки, може виникнути потреба у штучному збільшення процесорного часу, наприклад, для моделей тренажерів з метою наблизити віртуальне середовище, що генерується комп'ютером, до реального.

У даній роботі розглядається задача мінімізації часу імітаційного моделювання виконання Грід-програм на відеографічних процесорах за рахунок розпаралелювання програми моделювання.

На сьогоднішній день розроблено чимало інструментальних засобів моделювання Грід-систем. Серед них імітаційні моделі використовуються для більш точних вимірів продуктивності Грід-систем при конкретних значеннях параметрів. Вони дозволяють отримати довільну точність моделювання за рахунок врахування всіх необхідних деталей. Але на практиці потрібно досягати балансу між точністю моделювання та обчислювальною складністю.

Серед існуючих рішень можна 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 в 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] та [7]. На підставі порівняння можна зробити висновок, що найбільш зручним і функціональним є GridSim [1], фреймворк для мови програмування Java.

Окремим питанням є моделювання часу виконання задач на GPU, зокрема такі аспекти їх функціонування запропоновано детальну аналітичну модель як час виконання та споживання енергії [8-9], а також порівняння продуктивності CPU та GPU [10-11]. Застосування таких засобів моделювання дозволяє досягти великої точності моделювання, але потребують значних ресурсів для виконання моделі. До того ж вони не передбачають вбудовування в моделі Грід-середовища.

В роботах [2-3] були запропоновані спеціалізовані засоби імітаційного моделювання gpusim, засновані на інструментарії GridSim і спрямовані на моделювання роботи GPU з метою визначення оптимальних значень параметрів прикладних задач, що забезпечують найбільше прискорення виконання цих задач на GPU. Для збільшення ефективності використання таких засобів актуальною задачею є розпаралелювання програм самого імітаційного моделювання, що і є метою даної роботи.

## Система моделювання gpusim

**2.1. Основні концепції фреймворку Hazelcast.** Hazelcast – це платформа з відкритим програмним кодом для Java, яка використовується для побудови кластеру та масштабованого розподілення данних. Серед основних властивостей платформи є:

* Велика швидкість – тисячі операцій на секунду.
* Відмовостійкість – без втрати данних після аварій.
* Динамічне масштабування - по мірі додавання нових серверів.
* Дуже легке використання – досить додати у свій проект один jar файл.

Завдяки різним розподіленим структурам данних, розподілених можливостей кешування, еластичної природи, підтримки memcache, інтеграції з Spring та Hibernate Hazelcast є багатим на функціонал та готовий для промислового використання у якості платформи для створення кластерів.

**2.2. Можливості Hazelcast.** Hazelcast має такі можливості:

* Розподілені java.util.{Queue, Set, List, Map}
* Розподілені java.util.concurrency.locks.Lock
* Розподілені java.util.concurrent.ExecutorService
* Розподілений MultiMap для співідношення типу один до багатьох
* Розподілений “вузол” для publish/subscribe messaging
* Розподілена підтримка індексування та запитів
* Підтримка транзакцій та інтеграції з контейнером J2EE через JCA
* Шифрування на рівні сокетів для безпеки кластерів

**2.3. Опис архітектури gpusim.** Для досягнення розширюваності та гнучкості і, разом з тим простоти, було 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 С++ 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 в 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 в 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 наведена діаграма послідовностей для найбільш частого варіанта використання gpusim – проведення одного експерименту.

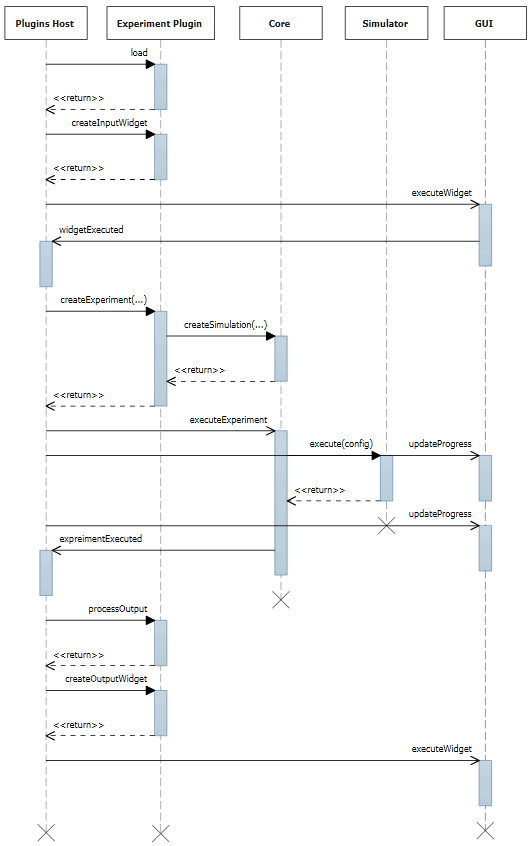


Рис. 2 – Діаграма послідовностей проведення експерименту в gpusim

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 користувачем. 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.

## Моделювання задачі множення матриць

**3.1. Задача множення матриць блочним алгоритмом.**

Для перевірки точності та актуальності розробленого середовища моделювання гетерогенних Грід-систем була обрана задача множення матриць за допомогою блочного алгоритму [12].

Результатом множення матриці  розмірністю й матриці  розмірності  є матриця  розмірності , кожен елемент якої обчислюється наступним чином:

, , .

Цей алгоритм вимагає операцій множення й додавання елементів матриць. При множенні квадратних матриць розмірності  кількість виконаних операцій має порядок . 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 , то результуюча матриця  буде тієї самої розмірності, що й вхідні матриці  і . Спочатку матриця  розбивається на прямокутні блоки розмірністю . Кожен з цих блоків обчислюється незалежно окремою задачею, для чого потрібно передати блок матриці  розмірністю  й відповідний блок матриці  розмірністю .

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 .

В [14] також побудовано емпіричну модель часу виконання блочного алгоритму на GPU. Ця модель має наступний характер: , де  та  – параметри, що визначають час завантаження та збереження даних при роботі з глобальною пам'яттю GPU. Ці параметри є частиною опису моделі і 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, cpuMaсhinePERating, gpuMaсhinePECount, gpuMaсhinePERating;
* пропускна здатність підключення між ресурсами і розподільниками завдань: resourceBaudRate, linkBaudRate;
* вартість використання ресурсів Грид – resourceCostPerSec;
* вартість операцій роботы із 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.

## Перевірка адекватності моделі та аналіз отриманих результатів

**4.1. Перевірка адекватності мо-делі.** Для перевірки адекватності створеної моделі був 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, були 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 був налаштований на роботу в діапазоні розмірів матриць [512; 1552] із інкрементом 80.

Спочатку були отримані значення попередньо встановлених параметрів в 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 було виявлено, що параметри cpuMachinePECount, cpuMachinePERating, resourceCostPerSec практично або зовсім не впливають на час симуляції, що 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, що пояснюється малим обсягом даних, що пересилаються між елементами Грід.

Вищеописаний експеримент був 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, були 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 наведені 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).

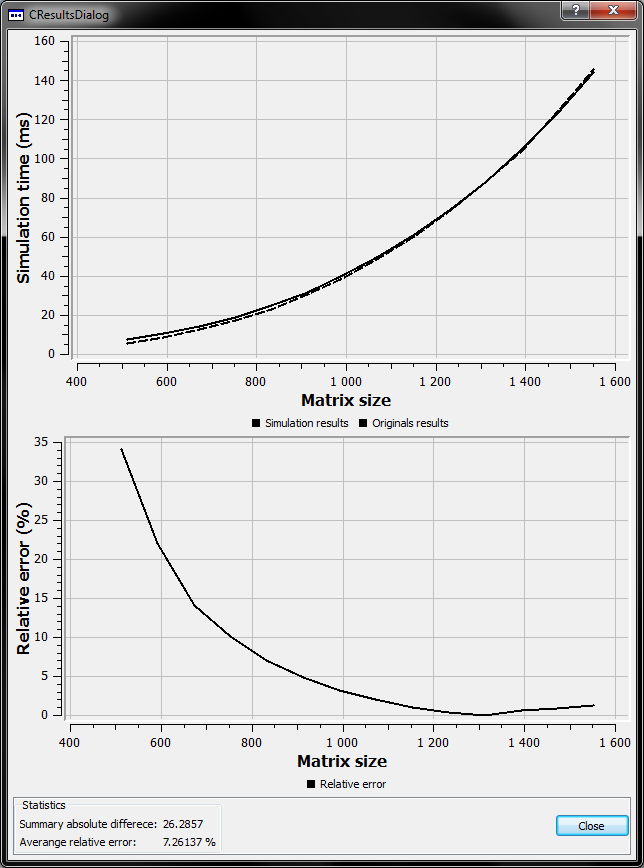


Рис. 3 – Порівняння часу виконання реального експерименту і симуляції для діапазону розміру матриці [512; 1552] з інкрементом 80

На рис.4 наведено ті ж графіки для діапазону розміру матриць [512; 3760] із інкрементом 16 (для всіх практично значущих даних, що були 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.

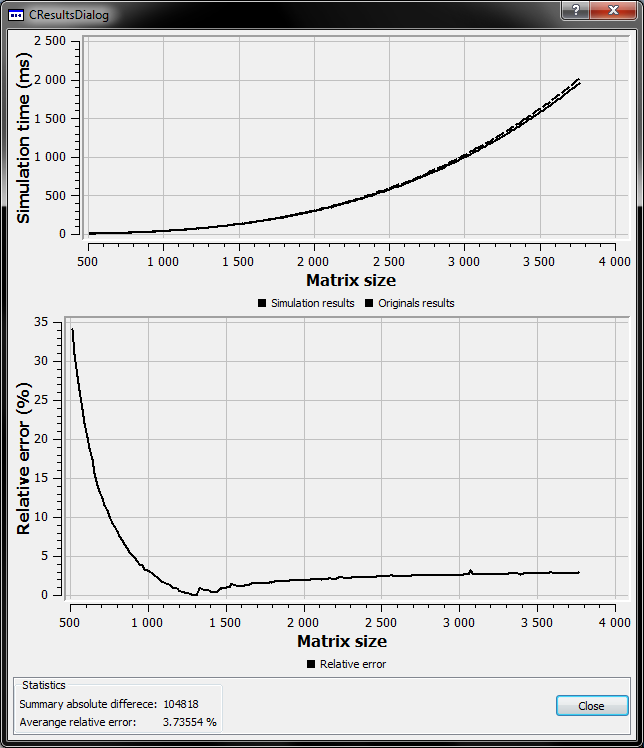


Рис. 4 – Порівняння часу виконання реального експерименту і симуляції для діапазону розміру матриці [512; 3760] з інкрементом 16

У ході дослідження моделі були 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 роботі [5], 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.

Друга особливість полягає в 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.

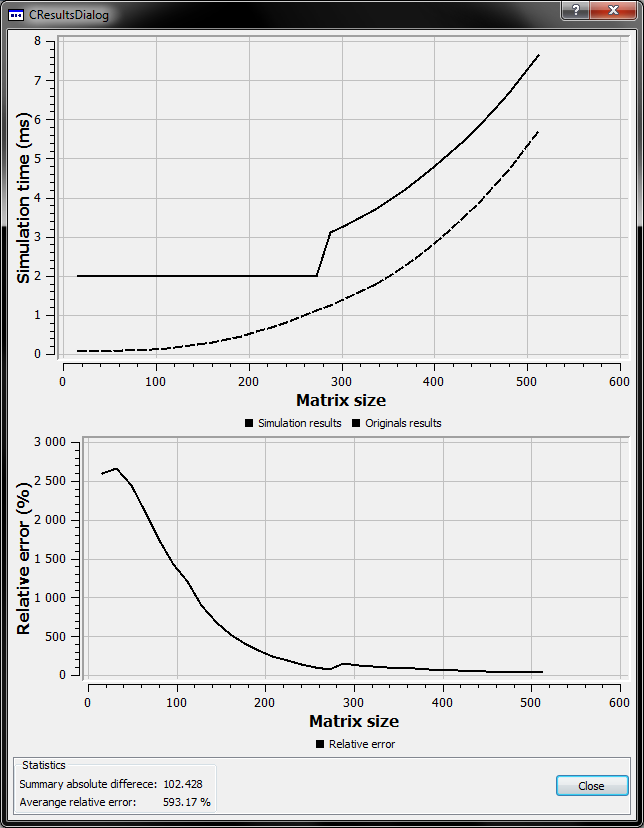


Рис. 5 – Порівняння часу виконання реального експерименту і симуляції для діапазону розміру матриці [16; 512] з інкрементом 16

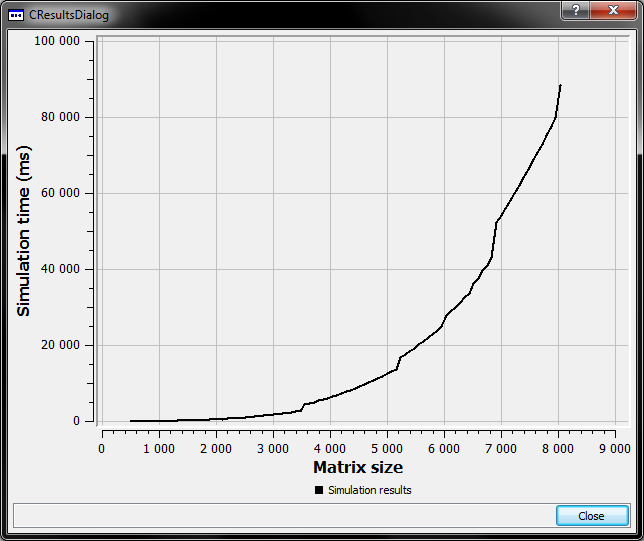


Рис. 6 – Різке підвищення часу симуляції при певних значеннях розміру матриці

## 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 був розроблений прототип середовища. Було проведено експериментальну перевірку розробленого прототипу на задачі множення матриць зар рахунок порівняння реальних вимірів та модельного часу виконання. За допомогою модуля оптимізації констант була 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. *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/. – 01.11.2013 р.
2. *Оконський І.В., Дорошенко А.Ю., Жереб К.А.* Інструментальні засоби моделювання гетерогенних середовищ заснованих на відеографічних прискорювачах // Проблеми програмування. – 2013. – № 1 С. 107–115.
3. Дорошенко А.Ю., Оконський І.В., Жереб К.А., Бекетов О.Г. Використання засобів моделювання для визначення оптимальних параметрів виконання програм на відеографічних прискорювачах // Проблеми програмування. – 2013. – № 2. – С. 23–31.
4. Лоу А.М., Кельтон А.Д. Имитационное моделирование. СПб: Издательская группа BHV, 2004.- 847 с.
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. *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.
9. *Hong S., Kim H.* An integrated GPU power and performance model // SIGARCH Comput. Archit. News. – 2010. – Vol. 38, N 3. – P. 280–289.
10. *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.
11. *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.
12. *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.
13. *Qt* Developer Network [Электроний ресурс]. – Режим доступу: http://qt-project.org/. – 01.11.2012 г.
14. *Intel* Core i7-3770k Processor [Электроний ресурс]. – Режим доступу: http://ark.intel.com/products/65523. – 01.11.2012 г.
15. *GeForce* GTX 650 Specifications [Электроний ресурс]. – Режим доступу: http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-650/specifications . – 01.11.2012 г.

Отримано ??.??.2012

***Про авторів:***

*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) 236-79-89

Інститут програмних систем   
НАН України,

03680, Київ-187,

проспект Академіка Глушкова, 40.

тел.: (044) 526 1538.

e-mail: [logrus.work@gmail.com](mailto:logrus.work@gmail.com),

[dor@isofts.kiev.ua](mailto:dor@isofts.kiev.ua)

[zhereb@gmail.com](mailto:zhereb@gmail.com)