УДК 004.7

О.П. Ігнатенко, Г.В. Парусімов, О.Б. Синецький

ОДНА МОДЕЛЬ ВИКОНАННЯ ОБЧИСЛЕНЬ У ГЕТЕРОГЕННИХ РОЗПОДІЛЕНИХ СЕРЕДОВИЩАХ

В даній роботі досліджується потокова модель паралельних обчислень у гетерогенній багатопроцесорній системі. Використання потокової моделі дозволяє обчислити «ідеальний» час закінчення, який дає нижню оцінку реального часу. Пропонується ігрова модель взаємодії користувачів, що виконують задачі у спільному обчислювальному середовищі на прикладі задачі множення матриць. Дією користувачів у даному випадку є розмір блоку на яку розрізається матриця. Проведені експерименти з використанням середовища імітаційного моделювання GridSim, що підтверджують теоретично отримані результати.

Вступ

Важливим напрямком сучасних наукових досліджень є надання сервісів доступу до обчислювальних ресурсів через мережу Інтернет. Ідея надання доступу до різноманітних обчислювальних ресурсів для за вимогою користувача або – обчислення як сервісу вже виникала в середині 90-х років у вигляді Grid інфраструктури.

Починаючи з 2008 року виникає термін – *хмарні обчислення* (Cloud computing), який представляє нову парадигму обчислень, що дозволяє отримувати доступ до програмних, обчислювальних та інформаційних ресурсів у вигляді мережевих сервісів.

З розвитком та ускладненням хмарних сервісів виникає проблема побудови ефективних алгоритмів забезпечення їх безперешкодної роботи. Одним з перспективних напрямків розвитку є теоретико-ігровий підхід, який дозволяє побудувати аналітичні моделі та дослідити динаміку системи з багатьма конфліктуючими за ресурси користувачами.

Хмарні обчислення стали новим напрямком розвитку інформаційних технологій, об'єднавши у собі ідеї комп'ютерних мереж та обчислювальних дата центрів. Власне хмара – це можливість надавати сервіси користувачам через мережу Інтернет. Існують три основні типи сервісів: програми у вигляді сервісів (Software as a Service (SaaS)), платформи у вигляді сервісів (Platform as a Service (PaaS)) та інфраструктура у вигляді сервісу (Infrastructure as a Service (IaaS)). Будемо називати такі системи – C/G системами.

Ефективність C/G системи забезпечують великі дата центри та технологія віртуалізації. Користувачі, таким чином, отримують доступ на основі принципу pay-as-you-use (сплати за використання) до найбільш сучасної інфраструктури не витрачаючи коштів на її створення та підтримку. Провайдери в свою чергу максимізують використання своїх потужностей та можуть балансувати навантаження через мережі для досягнення рівномірної загрузки.

Таким чином, ключовим елементом роботи C/G системи є ефективні алгоритми надання сервісів користувачам. Сучасна хмарна система існує у середовищі постійних змін. Змінюються технології, протоколи та програмне забезпечення. Змінюються користувачі, їх пріоритети, задачі і поведінка. Всі ці зміни є непередбачуваними. Тому теорія ігор, яка вже має історію успішного застосування до розв'язання задач маршрутизації, планування, керування потоками даних і перевантаженнями, є головним напрямком аналітичного моделювання хмарних систем.

Аналітичне моделювання дозволяє формалізувати роботу системи, поведінку користувачів і залучити їх у побудовану модель. Кожен гравець тут є автономним агентом, що прагне отримати певну частину обчислювального ресурсу. Припускаючи, що гравці діють раціонально – тобто

© О.П. Ігнатенко, Г.В. Парусімов, О.Б. Синецький, 2015

максимізують свою функцію корисності можна здійснити аналіз отриманої гри. Ключовим елементом аналізу є пошук точки рівноваги та її характеристика. Рівновага – це одна з головних ідей теорії ігор. Буквально – рівноважний стан відповідає ситуації, коли гравці досягли максимуму своїх функцій корисності. В залежності від зовнішніх факторів можливі стійкі і нестійкі рівноваги. Взагалі, концепцій рівноваги в іграх існує більш ніж два десятки.

Найбільш відомою є рівновага за Нешем: множина стратегій гравців утворюють рівновагу за Нешем, якщо ніхто з них не може покращити свою функцію корисності змінюючи окремо свою стратегію. Іншими словами кожен гравець використовує найкращу можливу стратегію у відповідь на стратегії своїх опонентів.

Задача множення матриць

Паралельне множення матриць – це одна з найбільш розповсюджених операцій, яка застосовується при обробці зображень, задачах метеорологічного прогнозу та багатьох інших. У реальних системах доводиться оперувати з матрицями великої розмірності, множення яких потребує $O(N^3)$ операцій, де N – розмірність матриць. Тому важливою проблемою є забезпечення максимальної ефективності обчислень.

Алгоритми множення матриць досліджуються вже більше 50 років. На сьогодні досягнуто значних результатів, зокрема алгоритм Штрассена дозволив зменшити кількість операцій до $O(N^{2.82})$, однак це не змінює ситуації принципово. Іншим напрямком є паралельне обчислення, яке дозволяє скоротити час виконання пропорційно до кількості задіяних ресурсів.

Пропонується розрізати матриці на блоки, які можуть обчислюватись незалежно або принаймні паралельно (обмінюючись інформацією у певні моменти часу). Кожен процесор обчислює свою частину і потім передає результат далі за визначеною схемою взаємодії. В кінці роботи результуюча матриця збирається з частин.

Особливо плідним напрямком є побудова аналітичної моделі алгоритму під конкретну конфігурацію обчислювальної системи з метою пошуку оптимального розв'язку [1]. В даному напрямку було досягнуто значних результатів, наприклад [2–4].

Архітектура обчислювальної системи. В даній роботі будемо вважати, що обчислювальні вузли не зв'язані один з одним і не можуть обмінюватись напрямку даними. Кожен вирішує свою задачу незалежно і повертає користувачу результат по закінченню.

Будемо вважати, що в системі є m обчислювальних елементів та кожен з них характеризується швидкістю роботи p_i, $i = 1,...,m$ – тобто кількістю операцій з плаваючою точкою за секунду, які він може здійснити.

Обчислювачі з'єднані лініями зв'язку з планувальником, який передає задачі та приймає від них результат. Будемо вважати, що лінії зв'язку ідентичні та мають швидкість передачі даних q.

Блочний алгоритм. Результатом множення матриці A розмірністю $m \times n$ і матриці B розмірності $n \times l$ є матриця C розмірності $m \times l$, кожен елемент якої обчислюється наступним чином:

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

Цей алгоритм вимагає $m \times n \times l$ операцій множення й додавання елементів матриць. При множенні квадратних матриць розмірності $n \times n$ кількість виконаних операцій має порядок $O(n^3)$. Необхідно побудувати паралельний алгоритм, який буде виконувати множення матриць за найкоротший час для довільної кількості неоднорідних процесорів (рис. 1).

Для експерименту було обрано наступну паралельну модифікацію послідовного алгоритму:

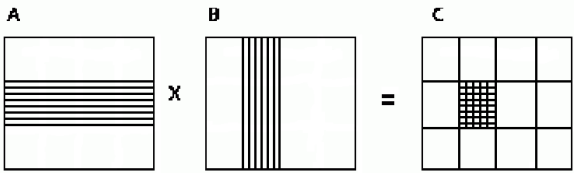

Рис. 1. Схема множення

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 $\frac{n}{n_x} \cdot \frac{n}{n_y}$, де n_x та n_y – кількість блоків по вертикалі й горизонталі. Кожен з цих блоків обчислюється незалежно окремою під-задачею, для чого підзадача потребує блок матриці A розмірністю $n \cdot \frac{n}{n_x}$ й аналогічний блок матриці B розмірністю $n \cdot \frac{n}{n_y}$.

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 – в 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 $O(n^3)$.

Кожна підзадача виконує

$$O\left(\frac{n}{n_x} \cdot \frac{n}{n_y} \cdot n\right)$$

скалярних операцій та потребує пересилки

$$O\left(\frac{n^2}{n_x} + \frac{n^2}{n_y}\right)$$

елементів.

Якщо враховувати тільки операції множення (вони дають головний вклад у час виконання задачі), то для обчислення однієї підзадачі потрібно виконати $\frac{n^3}{n_x n_y}$ операцій. Також потрібно переслати $\frac{n^2}{n_x} + \frac{n^2}{n_y}$ елементів та прийняти $n_x n_y$ елементів.

Зауваження. Описаний алгоритм множення матриць дозволяє розбивати задачу на блоки без «штрафів», тобто загальна кількість обчислень при розбитті на менші підзадачі не змінюється.

Потокова модель обчислення множення матриць. Будемо вважати, що виконуються наступні припущення.

1. Всі процесори починають роботу одночасно.

2. Планувальник здійснює призначення миттєво.

Зрозуміло, що мінімально можливі блоки досягаються при $n_x = n_y = n$. Нехай задане довільне розбиття n_x, n_y, яке забезпечує виконання задачі за час $T_k = T(n_x, n_y)$. Припустимо, що планувальник забезпечує найкраще можливе виконання, тоді задача користувача полягає у розв'язанні наступної задачі:

$$T(n_x, n_y) \to \min .$$

Функція $T(n_x, n_y)$, взагалі кажучи, може мати багато локальних мінімумів [4], оскільки існує мінімальна фіксована величина блоку, яка визначається розбиттям. При перепланування блоку з одного процесора на інший відбувається дискретна зміна часу виконання. Тому задача визначення мінімуму є складною.

Одним з розповсюджених підходів до аналізу таких задач є ідеалізація, яка полягає у дослідженні *потокової* моделі [3].

Припустимо, що користувач розділив задачу на m підзадач, кожна з яких має складність x_i, $i = 1,...,m$ та призначається процесору відповідного номера. Тоді $\sum_{i=1}^{m} x_i = n^3$. Потокова модель передбачає можливість розділення задачі на «шматочки» розміру ε, компонування з них підходящих підзадач та визначення загального часу при $\varepsilon \to 0$.

Твердження 1. Мінімальний час закінчення обчислень (без пересилок) для потокової моделі з одним користувачем дорівнює

$$T = n^3 \left(\sum_{i=1}^{m} p_i \right)^{-1}.$$

Це оцінка знизу будь-якого експериментального часу виконання.

Доведення. Нехай задано розбиття $x = (x_1,..., x_m)$, тоді час завершення роботи i-го процесора дорівнює $\frac{x_i}{p_i}$, у потоковій моделі всі процесори закінчують роботу одночасно – інакше можливо було б перерозподілити вектор x, зменшивши час. Отже

$$\frac{x_1}{p_1} = ... = \frac{x_m}{p_m} = T_c.$$

Підставивши $x_i = T_c p_i$ у $\sum_{i=1}^{m} x_i = n^3$ отримуємо результат для часу обчислення. Даний час буде мінімально можливим, оскільки не враховує мінімально можливі блоки при розбитті матриць на підзадачі.

Дискретна модель обчислення множення матриць. Нехай заданий мінімальний розмір блоку $\frac{n^2}{n_x n_y}$ (будемо вважати, що це ціле число), тоді для обчислення кожного елемента цього блоку потрібно виконати n множень, отже загальна складність мінімальної задачі дорівнює $\frac{n^3}{n_x n_y}$ операцій.

Потокова модель визначає оптимальний розподіл

$$\bar{x} = n^3 \left(\frac{p_1}{\sum p_i},..., \frac{p_n}{\sum p_i} \right).$$

Побудуємо сітку допустимих розподілів з урахуванням мінімального блоку. Позначимо множину можливих розподілів

$$K = \{(k_1,..., k_m) \mid k_j \geq 0, \sum_{j=1}^{m} k_j = n_x n_y\}.$$

Визначимо вектор $\bar{z}$ наступним чином:

$$z_i(k) = n^3 \frac{k_i}{n_x n_y},$$

даний вектор визначає обсяг обчислень, що призначений відповідному процесору. Зауважимо, що за припущенням елементи вектора $\bar{z}$ – цілі числа і

$$\sum_{i=1}^{m} z_i(k) = n^3,$$

для будь-якого $k \in K$.

Тоді для фіксованого $k \in K$ час закінчення обчислень дорівнює

$$T(z(k)) = \max_j \frac{z_j(k)}{p_j}.$$

Визначимо множину

$$R(\bar{z}(k)) =$$

$$= \left\{ y \in R^m \mid \sum_{i=1}^{m} y_i = n^3, |z_i(k) - y_i| \leq \frac{\sqrt{2} n^3}{2 n_x n_y} \right\}.$$

Твердження 2. Мінімальний час закінчення обчислень для дискретної моделі з мінімальним блоком $\frac{n^2}{n_x n_y}$ та одним користувачем дорівнює $T(z(k))$, де індекс $k \in K$ визначається включенням $x \in R(\bar{z}(k))$.

Доведення. Розглянемо всі можливі розподіли на блоки, які описується векторами $\bar{z}(k)$, $k \in K$. Множина цих точок утворює цілочисельну сітку в просторі R^m, яка належить гіперплощині

$$Y = \left\{ y \in R^m \mid \sum_{i=1}^{m} y_i = n^3 \right\}.$$

При пошуку розподілу, який відповідає мінімальному часу розглянемо процедуру руху за даною сіткою. При переміщенні з вузла $\bar{z}^1(k)$ на сусідній вузол $\bar{z}^2(k)$ одна з координат збільшується, інша зменшується а решта залишається незмінними. Відстань між сусідніми вузлами дорівнює

$\frac{\sqrt{2}n^3}{n_x n_y}$. Обчислимо множину точок з Y, відстань від яких до $\bar{z}^1(k)$ менша за відстань до будь-якої іншої точки сітки. Оскільки функція $T(x) = \max_j \frac{x_j}{p_j}$ є кусково лінійною, то мінімум функції часу $T(z(k))$ буде досягатися на точці сітки, найближчій до розв'язку потокової задачі. Цей факт описується включенням $x \in R(\bar{z}(k))$.

Некооперативна ігрова модель обчислення множення матриць. Некооперативна гра описує ситуацію прийняття рішень гравцями за умов конфлікту інтересів. Під терміном гравець тут ми розуміємо користувача, який впливає на систему шляхом вибору стратегій – дій, які він може застосовувати. В залежності від дій його та інших учасників відбувається визначення виграшів гравців. Говорять, що гравець раціональний, якщо його дії спрямовані на максимізацію власного виграшу.

Якщо у грі приймають участь хоча б двоє учасників, можлива ситуація, коли перший гравець може покращити свій виграш за рахунок зменшення виграшу інших. В такому випадку кажуть про конфліктну взаємодію. Частинним випадком конфлікту є ситуація повністю протилежних інтересів – коли виграш одного є програшем іншого. Такі ігри називають антагоністичними або іграми з нульовою сумою.

Зауважимо, що некооперативність не означає, що гравці взагалі не кооперуються, а тільки те, що немає зовнішніх причин, які б спричиняли координацію або узгодження їх стратегії. Будь-яка кооперація що може виникнути має виходити зі структури гри і визначатися функціями корисності учасників.

Гру називають статичною, якщо гравці приймають свої рішення одноразово, незалежно один від одного. В певному розумінні, статична гра не залежить від часу. Навіть якщо гравці приймають рішення протягом певного часового інтервалу, якщо вони не володіють інформацією щодо дій інших гравців, така гра є статичною.

У динамічних іграх гравці отримують певну інформацію щодо рішень інших учасників і можуть змінювати свою стратегію у часі, тобто приймають рішення більше одного разу. Динамічні ігри є найбільш складним для аналізу і відіграють важливу роль у дослідженні процесів у мережах [5, 6].

Опишемо задачу множення матриць у вигляді статичної некооперативної гри.

Основні визначення. При визначенні статичних ігор загальновживаною є стратегічна або нормальна форма, яка включає опис трьох компонентів: множини гравців, їх стратегій і виграшів.

Гравцями в даному випадку являються користувачі $\{u_i\}_{i \in L}$, L – множина індексів, які мають доступ до спільного ресурсу з m обчислювальних елементів з швидкістю роботи p_i, $i = 1,...,m$. Будемо вважати, що для кожного користувача задані квадратні матриці розмірності n_l, $l \in L$. Тоді стратегіями користувачів є допустиме розбиття матриць на блоки $k_l \in K_l$, $l \in L$. Після розбиття матриць, блоки потрапляють у планувальник, який надсилає їх на процесори згідно з певним визначеним алгоритмом роботи. Часом закінчення обчислень T_l, $l \in L$ будемо називати час закінчення останньої задачі користувача u_l. Кожен користувач намагається зменшити свій час закінчення і через обмеженість спільних ресурсів виграш одного користувача спричинить програш інших.

Середовище моделювання GridSim

GridSim розроблений в Університеті Монаш (Австралія) з метою моделювання роботи GRID-системи з можливістю імітаційного моделювання характеристик ресурсів і обчислювальних мереж при різних конфігураціях планувальників програм для розподілених обчислювальних систем, таких як кластери і Grid. GridSim

розроблений на мові програмування Java і дозволяє моделювати різні класи гетерогенних ресурсів, брокерів ресурсів (RBS), користувачів, програм і планувальників у розподіленому обчислювальному середовищі.

До основних характеристик GridSim можна віднести:

– моделювання різних конфігурацій обчислювальної мережі GRID-систем різних топологій;

– моделювання різних політик планування завдань на обчислювальному кластері, як вже реалізованих, так і розроблених користувачем;

– використання даних про завантаження реальних суперкомп'ютерів для проведення експериментів;

Опишемо архітектуру GridSim. Нижній рівень – Java Virtual Machine (JVM), основна частина виконавчої системи Java, так званої Java Runtime Environment (JRE). Віртуальна машина Java інтерпретує і виконує байт-код Java, попередньо створений з вихідного тексту Java-програми компілятором Java.

Другий рівень являє собою SimJava бібліотеку, яка використовується для управління моделюванням і взаємодією об'єктів GridSim. Третій рівень – безпосередньо GridSim. Вона моделює основні об'єкти Grid такі, як ресурси та інформаційні сервіси. Цей рівень містить підрівень моделювання ресурсів і визначення топології мережі. Статистичні дані автоматично збираються і доступні через об'єкт GridStatistics. Також на цьому рівні знаходяться об'єкти для моделювання DataGrid: ReplicaCatalogue і ReplicaManager.

Останній, четвертий рівень визначає функціональні можливості для планувальників і брокерів ресурсів. Верхній рівень призначений для конфігурування параметрів моделювання.

Опишемо виконання експерименту в GridSim. Об'єкт GridUser представляє собою користувача. Важливим параметром для користувача є пропускна здатність передачі даних (baud_rate) за яким рахується час передачі даних та повернення результату. Завдання від користувача направляється на відповідний Брокер Ресурсів (RBS) у вигляді об'єкта Gridlet. Gridlet є пакетом, який містить всю інформацію, пов'язану з завданням – довжину завдання (виражену в мільйонах операцій, MI), розмір вхідних і вихідних файлів, характеристики користувача. Ці основні параметри допомагають визначити час виконання, час, необхідний на передачу файлів і повернення обробленого об'єкту Gridlet користувачу разом з результатами обробки. RBS запитує у GRID Information Service (GIS) список доступних GridResource. GIS реєструє ресурси, що дає можливість відстежувати список доступних ресурсів GRID. Після чого, RBS виконує вибір ресурсу і направляє Gridlet на обробку, згідно з установленим сценарієм. GridResource – об'єкт, складений з однієї або більше машин (Machine). Machine містить один або більше CPU (processing elements, PE), для кожного з яких потужність задається кількістю операцій за секунду (mipsRating-millions of instructions per second (MIPS)). Після виконання Gridlet, GridResource повідомляє RBS про завершення. Слід зазначити, що ресурс може виконувати кілька завдань одночасно.

Для розглянутого прикладу задачі паралельного множення матриць блочним методом задача поділяється на незалежні підзадачі – множення блоків матриці , кожна з яких обчислюється на окремому вузлі CPU. Якщо N – розмірність квадратної матриці, m – кількість блоків по одному з вимірів матриці, то загальна кількість незалежних підзадач буде $m*m$ і розмірність блоку $n = N/m$. Тому одне завдання користувача Gridlet буде складатися з

$$length = n*n*(2*N-1)/1000000\,MI$$

операцій, розмір вхідних даних

$$file\_size = 2*n*n*8$$

та вихідних

$$output\_size = n*n*8$$

байтів.

Експериментальні дані

Для перевірки отриманих результатів та обґрунтування подальших дослі-

джень були проведені експерименти з імітаційного моделювання обчислення добутку матриць на різних Grid системах.

На рис. 2 показані результати залежності часу закінчення виконання від кількості блоків для системи з одним, двома, трьома і чотирма процесорами.

Також показані аналогічні залежності для неоднорідних систем. Отримані графіки залежностей добре узгоджуються з результатами, отриманими при експериментальних вимірах на кластері Інституту програмних систем НАН України [4].

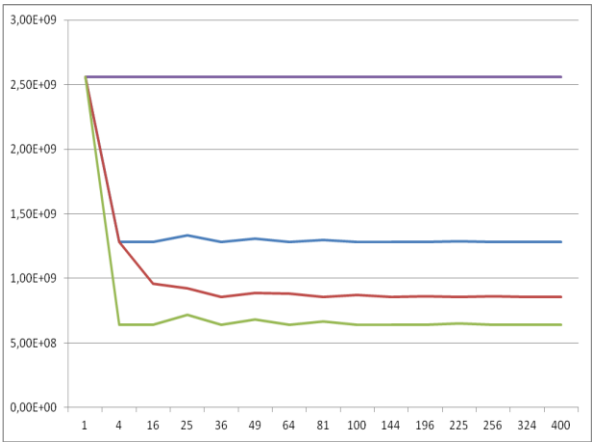

Рис. 2. Однакові процесори

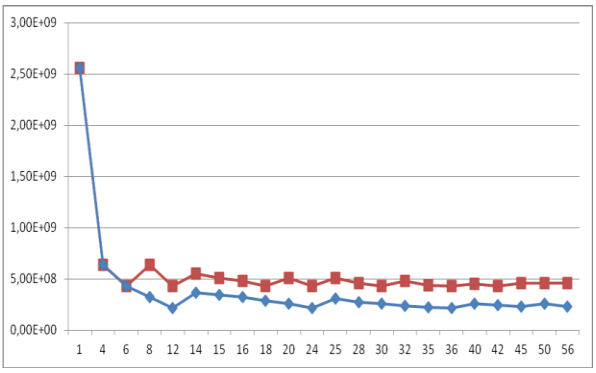

Рис 3. Неоднорідні процесори

Висновки

В даній роботі досліджується потокова модель паралельних обчислень у неоднорідних багатопроцесорних системах. Використання потокової моделі дозволяє обчислити «ідеальний» час закінчення, який дає нижню оцінку реального часу. Проведені експерименти, що підтверджують теоретично отримані результати.

1. *Srinivasa Prasanna G.N., Musicus B.* Generalized Multiprocessor Scheduling Using Optimal Control // Proc. SPAA. – 1991. – P. 216–228.
2. *Grosu D., Chronopoulos A.T.* Noncooperative load balancing in distributed systems // Journal of Parallel and Distributed Computing, 2005. – **65** (9). – P. 1022 – 1034.
3. *Nazarathy Y., Weiss G.* A Fluid Approach to Large Volume Job Shop Scheduling // Journal of Scheduling, 13(5). – 2010. P. 509–529.
4. *Дорошенко А.Ю., Ігнатенко О.П., Іваненко П.А.* Про одну модель оптимального розподілу ресурсів у багатопроцесорних середовищах // Проблеми програмування. – № 1. – 2011. – С. 21–28.
5. *Андон Ф.И., Игнатенко А.П.* Моделирование конфликтных процессов в сети Интернет // Кибернетика и системный анализ. – 2013. – № 4. – С. 153–162.
6. *Ignatenko O., Synetskyi O.* Evolutionary Game of N Competing AIMD Connections. In Information and Communication Technologies in Education, Research, and Industrial Applications. Springer International Publishing. – 2014. – P. 325–342.

Одержано 20.01.2015

Про авторів:

Ігнатенко Олексій Петрович,
кандидат фізико-математичних наук,
старший науковий співробітник,

Синецький Олександр Борисович,
аспірант,

Парусімов Генадій Вікторович,
аспірант.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, Київ-187,
Проспект Академіка Глушкова, 40.
Тел.: 526 6025.
E-mail: o.ignatenko@gmail.com