МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Матеріали наукової конференції студентів, магістрантів та аспірантів

«Інформатика та обчислювальна техніка – ІОТ-2017»

23 травня 2017 року

(кафедра «Обчислювальної техніки»)

ТЕЗИ

Київ 2017

Оргкомітет конференції:

Голова:

О.А. Павлов – декан факультету, д.т.н., професор.

Заступник голови :

О .М. Долголенко – заст. декана по науковій роботі, к.т.н., доцент

Співголови організаційного комітету:

С.Г. Стіренко – завідувач кафедрою ОТ, д.т.н., професор

Члени організаційного комітету:

Ю.О. Кулаков- д.т.н., професор кафедри ОТ

В.І. Жабін – д.т.н., професор кафедри ОТ

В.П. Симоненко - д.т.н., професор кафедри ОТ

Г.М. Луцький – д.т .н., професор кафедри ОТ

О.П. Марковський – к.т.н., доцент кафедри ОТ

Секретар конференції:

Н.Є. Куц Пров. інженер кафедри ОТ

ЗМІСТ

УДК 683.519

ВЕЦКО В.І.

ПРОГНОЗУВАННЯ СТАНУ БАТАРЕЇ ЕЛЕКТРИЧНИХ ТРАНСПОРТНИХ ЗАСОБІВ ДЛЯ ОЦІНКИ ЖИТТЄЗДАТНОСТІ ПРИ КАРШЕРІНГУ

Каршерінг компанії впроваджують електричні транспортні засоби (EVs) в свій автопарк. Однак дані свідчать про те, що в даний момент при використанні електромобілів, не вдається досягти задовільної комерції. Потенційною причиною цього є більш нагружене використання транспортного засобу, що характерне для короткочасної оренди автомобілів, а також наслідки нагруженого використання транспортного засобу для стану батареї (SoH). У цій статті здійснено прогнозування SoH двох однакових електромобілів, що використовуються в різних практиках автомобільного обміну. Для цього використовуються отримані дані, що від зарядних станцій і різних датчиків EV. Розуміння користувачами правил водіння та поведінки зарядки батареї може служити цінним орієнтиром для системи короткочасної оренди автомобілів. Зокрема, результати прогнозування показують, що в залежності від умов експлуатації, момент досягнення батареєю електромобіля свого теоретичного кінця життя може відрізнятися на чверть часу.

Car-sharing companies are introducing electric vehicles into their fleet. At this point shared electric vehicles systems are failing to reach satisfactory commercial viability. A potential reason for this is the effect of higher vehicle usage, which is characteristic of car sharing, and the implications on the battery's state of health (SoH). In this paper, we forecast the SoH of two identical EVs being used in different car-sharing practices. For this purpose, we use real life transaction data from charging stations and different electric vehicles sensors. The results indicate that insight into users' driving and charging behavior can provide a valuable point of reference for car-sharing system designers. In particular, the forecasting results show that the moment when the battery of an electric vehicle reaches its theoretical end of life can differ in as much as a quarter of the time when vehicles are shared under different conditions.

Ключові слова: ПРОГНОЗУВАННЯ СТАНУ; ОРЕНДА АВТОМОБІЛІВ; ЕЛЕКТРИЧНИЙ ТРАНСПОРТНИЙ ЗАСІБ (EV); ВОДІННЯ І ПОВЕДІНКА ЗАРЯДКИ; СТАН ЗДОРОВ'Я БАТАРЕЇ (SoH); ДЕГРАДАЦІЯ БАТАРЕЇ; СПІЛЬНА ЕКОНОМІКА

1. Вступ

В останні роки відбулося переосмислення особистої мобільності. Є дві основні мотивації для цього. По-перше, після десятиліть використання автомобілів, ми досягли точки, де транспорт відповідає за 23% світових викидів. Очевидно, що теперішня система мобільності є нестійкою в її нинішньому вигляді і що необхідні нові, більш стійкі та енергоефективні рішення. По-друге, дослідження показали, що особисті транспортні засоби використовуються в середньому близько години в день [1,2]. Припарковані більшу частину часу, вони займають цінний простір для суспільства. Цей ефект особливо помітний в міських районах, де населення продовжує зростати. За даними Всесвітньої організації охорони здоров'я, 54% від загального світового населення проживає в міських районах. Тільки в Європі, в міських районах проживає понад дві третини населення Європейського Союзу.

Одним з рішень є спільне використання автомобілів. Оренда автомобілів дозволить задовольнити потребу в особистій мобільності, при цьому забезпечуючи більш низькі витрати для фізичних осіб і більш високу зручність використання транспортних засобів, що робить автомобілі більш економічно ефективним [3]. Fellows і Pitfield аналізуючи витрати і вигоди для оцінки оренди автомобіля виявили, що люди отримують економічну вигоду за рахунок скорочення подорожей

за ціною до 50%, а економіка в цілому виграє за рахунок зменшення пробігу транспортних засобів, збільшення середньої швидкості та економії в паливі, зменшення аварій та викидів. Більш систематичні результати в географічно більшому масштабі можна знайти в дослідженні Шахін і Коена [2], які визначили, що кожен автомобіль спільного користування зменшує потребу на 4-10 приватних транспортних засобів у Європі, 6-23 в Північній Америці, і 7-10 в Австралії.

В цій статті порівнюється вплив двох різних практик використання автомобілів спільного користування на продуктивності батареї. Це робиться за рахунок докладних даних електрокарів і даних про підзарядку для прогнозування SoH батарей двох однакових електромобілів в різних практиках. Основні дослідницькі матеріали роботи можуть бути розташовані в наступних областях: (1) детальний аналіз спільного водіння користувачів електрокарів і частоти підзарядок, сподіваючись таким чином забезпечити додакову інформацію для планування транспортної системи; (2) вплив двох різних методів обміну електрокарів на SoH батареї, оцінити життєздатність цих методів.

2. Практики каршерінгу та їх користувачів

Ми порівнюємо два однакових електромобілі, що орендуються за двох різних практик каршерінгу. Перший автомобіль є васністю компанії, що здає в оренду більш ніж 800 електричних і звичайних транспортних засобів. Дані автомобілі доступні для оренди більш ніж 24000 користувачів. Правилами користування прописано, що після використання користувач зобов'язаний підключити автомобіль для підзарядки, що гарантує максимально заряджений акумулятор для наступного користувача. В системі бронювання користувачі вказують приблизно скільки кілометрів і часу триватиме їхня оренда, для спрощення планування обслуговування. Другий автомобіль знаходиться в спільній власності житлового комплексу, де жильці займаються доглядом спільного майна. Підхід заснований на понятті спільної економіки. Автомобіль активно експлуатується серед 35 членів. Система бронювання передбачає вказування часових інтервалів, протягом якого буде експлуатуватися автомобіль. Підзарядка транспортного засобу здійснюється тільки з ініціативи користувача. В обох випадках автомобіль повинен бути повернутий в початковий пункт. Обидва автомобілі експлуатуються в однакових кліматичних умовах та регіоні.

3. Розуміння споживачів – манера водіння та повернення

Для того, щоб забезпечити якомога повне розуміння того, як були використані електромобілі, детально досліджувалися манера водіння і зарядка обох груп користувачів. Дані зарядки було зібрано від зарядних станцій. Дані водіння було зібрано з транспортних засобів за допомогою GPS та CAN шини (швидкість автомобіля, струм, напруги, часові мітки, заряд, стан двигуна).

З огляду на використання загальних електромобілів, автомобілів членів однієї компанії, їх використовують в основному в якості другого автомобіля, з високою частотою в другій половині дня і у вихідні дні (22% поїздок всіх водіїв були зроблені у суботу). Члени спільного житла були більш схильні використовувати загальний EV в ранкові години (рис. 1).

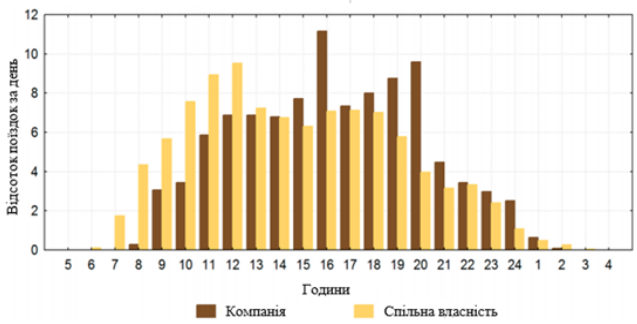

Рис. 1. Поведінка водіння, розподіл щоденних поїздок

При розгляді питання про тривалість зроблених поїздок, 72% поїздок користувачів спільно для будинку були коротше, ніж в 10 км. Згідно з офіційною статистикою стверджує, що середня поїздка автомобіля у Фландрії становить приблизно 34,4 км; середня поїздка зроблені електромобілі належать

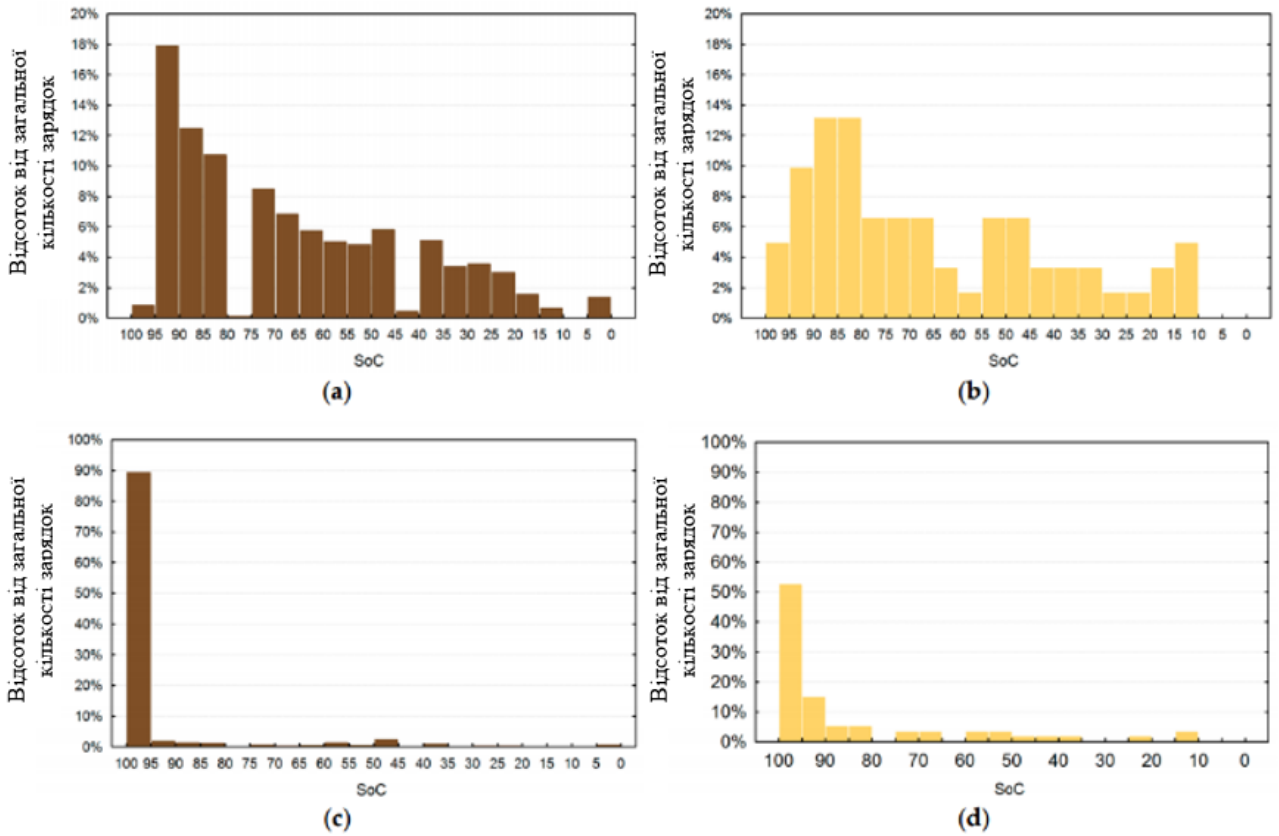

Рис. 2. Поведінки зарядки - стан заряду (SoC) на початку і в кінці зарядки; (a,c) SoC на початку та кінці зарядки для каршерінг компанії; (b,d) SoC на початку та кінці зарядки для автомобіля в спільній власності.

автомобіля обміну компанії було 32,56 км, а шляхом спільного житла, 8,4 км.

З огляду на перезарядку, члени спільного житла, як правило, підзаряджали автомобіль зранку або пізно ввечері. Автомобіль каршерінгової компанії підключався після кожного використання. Таким чином автомобіль не буде від'єднано від зарядного пристрою до наступного користування, що іноді може зайняти більше тижня. В цілому, 7,6% підзарядок були зроблені з super charge. Всі перезарядки членами спільного житло були коротше, ніж день, де 33% з них були мене півгодини, і 4% з super charge.

Правила оренди також впливають на кількість зарядок в день. Для автомобілів каршерінгової компанії, число перезарядок переважно відповідає кількості користувачів в день (переважно один). Для спільного користування, транспортний засіб часто заряджається більше, ніж один раз в день. Беручи до уваги SoC батареї, то в 20% випадках автомобілі компанії заряджалися, але SoC батареї було вище, ніж 90%. У більшості випадків, батарея залишалася до повної зарядки. Для членів спільного користування, батарея повністю заряджалася тільки в половині випадків, в той час як значення SoC, при якому акумулятор був підключений на підзарядку, був більш рівномірно розподілений (рис. 2).

4. Методологія

Для того, щоб оцінити вплив практики каршерінгу, поведінку водіння та зарядки батареї електрокару, досліджувалися SoH, що визначаються як різниця між корисною місткістю і кінцевою місткістю [6]. SoH зазвичай виражається у вигляді відсотка від номінальної потужності і є мірою довгостроковості батареї [4,5]. У порівнянні з SoH, SoC визначається як відсоток від доступної ємності і є мірою короткочасної здатності батареї. Більш детально, SoC показує залишковий заряд батареї в даний момент, в порівнянні з енергією при повному заряді, що дає уявлення про те, скільки батарея працюватиме до наступної перезарядки. В аналогії зі звичайними автомобілями, SoC відповідає паливному манометру, в той час як SoH буде відповідає здатності паливного бака зберігати паливо. У цій аналогії,

паливний бак матиме змінний доступний обсяг.

У даній роботі SoH розраховується з даних експлуатації, в той час як SoC зібрана з CAN шини. Дані з CAN шини засобів зібрані з частотою 10 Гц. На рис. 3. показано миттєва передача потужності батареї (W) з плином часу під час експлуатації.

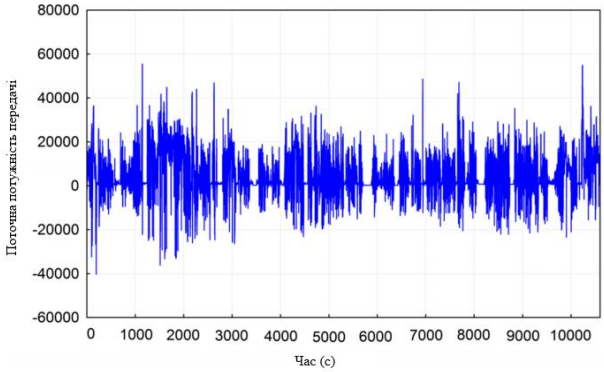

*Рис. 3. Графік V*I(W)*

На основі миттєвої передачі потужності батареї сумарна чиста енергія подається від акумуляторної батареї може бути обчислена шляхом трапецієподібної чисельного інтегрування струму батареї (I) і напруги (V) з плином часу, як показано рівнянням:

$$E_{te} = \int V * I \, dt$$

Крім того, знаючи SoC, на початку (SoC_1) і в кінці (SoC_2), SoH може бути визначений на основі рівняння:

$$SoH = \frac{E_{te}}{SoH_{100\%} * (SoC_1 - SoC_2)}$$

5. Вплив каршерінгу на стан батареї

Для обох електромобілів, використовуючи обчислення SoH з секції методології, ми визначили значення SoH для кожного циклу розрядки. Кількість записаних циклів трохи розрізнялися, для транспортного засобу, що належить компанії, було 59 і для автомобілю, що є спільній власності – 63. У літературі вказується, що протягом перших 500 циклів або близько того, ємність змінюється лінійно, обчислені значення SOH були використані для визначення цього лінійного тренда.

Для оцінки лінійного тренда і його застосовності для прогнозування майбутніх SoH елементів живлення, ми розрахували найменше квадратичне відхилення, середнє відхилення, відносну квадратичну похибку та відносне абсолютне відхилення:

$$LSD = \frac{\sum_{i=1}^{N}(E_i - O_i)^2}{N-1}$$

$$AD = \frac{\sum_{i=1}^{N}|E_i - O_i|}{N-1}$$

$$RSE = \frac{\sum_{i=1}^{N}[(E_i - O_i)/E_i]^2}{N-1}$$

$$RAD = \frac{\sum_{i=1}^{N}|E_i - O_i|/E_i}{N-1}$$

де: N – число спостережень або сума ваг; E_i – передбачене значення випадку i.

Таблиця 1 містить більш детальний аналіз оцінки лінійного тренда для обох автомобілів.

Табл. 1. Достовірність лінійних оцінок

Достовірність виміру	Спільна власність	Каршерінг компанія
LSD	1.605182	1.699434
AD	0.949703	1.07033
RSE	0.000184	0.00030
RAD	0.010116	0.01410

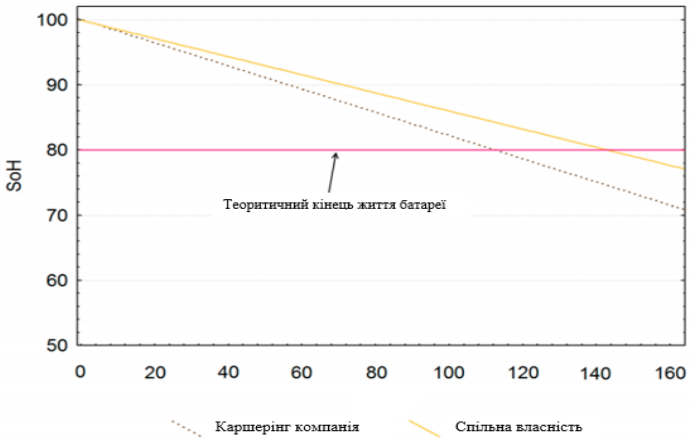

Рис. 4. Лінійна екстраполяція SoC тенденції і теоретичного кінця життя батареї

Як правило, це прийнято в автомобільній промисловості, час життя батареї вимірюється кількістю повних циклів її заряду-розряду (розряд батарея наступає,

коли її номінальна потужності падає нижче 80% від її первісної номінальної потужності [7]). Екстраполяція лінійних трендів для EV батареї, показана на рис. 4.

7. Висновки

Дані від зарядних станцій і датчиків електрокарів можуть бути успішно використані для прогнозування SoH батареї. В статті збагачені існуючі імітаційні моделі в області оцінки SoH з емпіричним підтвердженням аналізу. Аналіз оснований на даних з двох однакових електромобілів, використання яких відрізнялося в середньому SoC, DoD і відсотком використання super charge. Результати вказують на те, що відстрочення зарядки і менше використання supper charge можуть уповільнити процес деградації батареї.

Крім того, деградація батареї пов'язана з вартістю батареї і вартістю автомобіля (вартість батареї складає близько 54% від загальної вартості автомобіля [8]). Дані висновки можуть бути цінним довідником для каршерінгу електрокарів і можуть бути інтегровані в існуючі системи каршерінгу.

Список літератури

1. Meijkamp, R. Changing consumer behaviour through eco-efficient services : An empirical study of car sharing in the Nerherlands. Bus. Strategy Environ. **1998**, 7, 234–244.
2. Shaheen, S.A.; Cohen, A.P. Growth in worldwide carsharing: An international comparison. Transp. Res. Rec. J. Transp. Res. Board **2007**, 1992, 81–89.
3. Fellows, N.; Pitfield, D. An economic and operational evaluation of urban car-sharing. Transp. Res. D Transp. Environ. **2000**, 5, 1–10.
4. Nikolian, A.; Firouz, Y.; Gopalakrishnan, R.; Timmermans, J.-M.; Omar, N.; van den Bossche, P.; van Mierlo, J. Lithium ion batteries—Development of advanced electrical equivalent circuit models for nickel manganese cobalt lithium-ion. Energies **2016**, 9, 360.
5. Le, D.; Tang, X. Lithium-ion battery state of health estimation using Ah-V characterization. In Proceedings of the Annual Conference of Prognostics and Health Management (PHM) Society, Montreal, QC, Canada, 20–23 June 2011.
6. Marra, F.; Træholt, C.; Larsen, E.; Wu, Q. Average behavior of battery-electric vehicles for distributed energy studies. In Proceedings of the 2010 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe), Gothenburg, Sweden, 11–13 October 2010.
7. Magnor, D.; Gerschler, J.B.; Ecker, M.; Merk, P.; Sauer, D.U. Concept of a battery aging model for lithium-ion batteries considering the lifetime dependency on the operation strategy. In Proceedings of the European Photovoltaic Solar Energy Conference, Hamburg, Germany, 21–25 September 2009.
8. Cars 21. How to Reduce EV Production Costs? EV Battery Tech USA. 2011. Available online: http: //www.cars21.com/news/view/670 (accessed on 18 October 2016).

УДК 681.325

КРАВЧУК А.А.
САВЕРЧЕНКО В.Г.

УНИФИКАЦИЯ ВЫЧИСЛЕНИЙ В ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ КОНВЕЙЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Рассматриваются вопросы унификации вычислений в проблемно-ориентированных конвейерных вычислительных системах на основе введения обобщенной процедуры, эквивалентной последовательности базовых операций. При этом любая операция рассматривается как частный случай выделенной обобщенной процедуры, получаемой на основе системы правил подстановок. Это достигается путем доопределения потоков данных при помощи подстановки в обобщенную процедуру нулей и единиц вместо тех значений

операндов, которые не определены в заданной операции. Введение в машинный язык обобщенных процедур рассматривается как предпосылка для повышения эффективности проблемно-ориентированных вычислительных систем.

The questions of unification of computing in problem-oriented conveyor computer systems based on the introduction of a generalized procedure, the equivalent sequence of basic operations. At the same time, any operation is considered as a special case of the isolated generalized procedure, obtained on the basis of the system of rules of substitutions. This is achieved by adding data streams by substituting zeros and ones for the generalized procedure instead of those values of operands that are not defined in the specified operation. Introduction to the machine language of generalized procedures is considered as a prerequisite for improving the efficiency of problem-oriented computing systems.

Ключевые слова: проблемно-ориентированные конвейерные вычислительные системы, унификация вычислений, обобщенная процедура.

1. Введение

При построении современных высокопроизводительных вычислительных средств обработки информации широкое распространение получили конвейерные и матричные процессоры, использующие совмещение и распараллеливание операций соответственно.

В конвейерных процессорах, каждая строка конвейера перед началом либо во время вычислений настраивается на выполнение определенной операции. В матричных процессорах процессорные элементы работают параллельно и выполняют одну и ту же операцию.

Для повышения эффективности вычислений при решении широкого круга задач необходимо организовать связи между процессорными элементами таким образом, чтобы наиболее полно обеспечить загрузку процессорных элементов. При этом разнообразие выполняемых операций усложняет как систему управления вычислениями, так и в ряде случаев уменьшает производительность системы в целом [1].

2. Постановка задачи

Представляется актуальным поиск таких системотехнических решений, которые позволяли бы реализовать обработку информации, эффективно сочетающих совмещение и распараллеливание операций, и учитывающих специфику конкретного применения [2].

Один из наиболее эффективных подходов к построению проблемно-ориентированных вычислительных систем (ПОВС), базируется на унификации вычислений и связан с выбором минимального набора базовых операций, реализующих оптимальным образом любую задачу, принадлежащую данному классу.

В общем случае алгоритм решаемой задачи может быть представлен в виде последовательности элементов, принадлежащих к множеству допустимых операторов входного языка ПОВС. При этом каждому оператору ставится в соответствие определенная группа одиночных команд внутреннего языка, которые обычно называются машинными командами.

Каждая машинная команда определяет выполнение одной из множества базовых многоместных операций, эквивалентных некоторой последовательности бинарных операций. Иными словами, выполняется отображение $S: A_i \to B^{(i)}$, где $B^{(i)} = \{B_1^{(i)}, \dots, B_r^{(i)}, \dots, B_{N^{(i)}}^{(i)}\}$ – последовательность базовых операций, соответствующая оператору A_i; $B_r^{(i)} = \{B_{r,l}^{(i)} \mid l = \overline{1, N^{(r)}}\}$ – базовая многоместная операция, эквивалентная последовательности $N^{(r)}$ бинарных операций; r и l – порядковые номера базовой и бинарной операций в цепочке операций соответственно; $B^{(i)} \in B$ ($i = \overline{1, N}$, $r = \overline{1, N^{(i)}}$).

3. Методика выделения обобщенных процедур

С унификацией вычислений будем связывать такое сокращение машинных

команд, определенных для конкретной ПОВС, при котором длина последовательности базовых операций не увеличивается. В предельном случае ($M=1$) каждому оператору входного языка ПОВС ставится в соответствие одна и та же обобщенная машинная команда, которая определяет операцию вычисления некоторой обобщенной процедуры (ОПР), эквивалентной последовательности базовых операций. При этом любая i-я операция, выполняемая в ПОВС, рассматривается как частный случай выделенной ОПР, получаемой на основе i-ой системы правил подстановок. Это достигается путем доопределения потоков данных при помощи соответствующих подстановок в ОПР кодов операндов из множества {0,1} вместо тех значений операндов, которые не определены в заданной операции. Другими словами, при унификации вычислений осуществляется отображение $S': A_i \to L^{(i)}$, где $L^{(i)}$ – i-я реализация ОПР L, аргументами которой могут быть как переменные, так и постоянные величины, причем $L=\{L_i \mid t=\overline{1,K}\}$ – цепочка K бинарных операций, вызываемых обобщенной машинной командой.

Введение ОПР позволяет унифицировать способ выполнения операций, а с ним и процесс вычислений в ПОКС. Однако для реализации этой возможности необходимо учитывать специфические особенности ОПР, к которым в первую очередь следует отнести относительно высокую ее сложность.

Степень сложности ОПР, определяемая числом бинарных операций, составляющих эквивалентную последовательность, может достигать. Поэтому при выборе структуры ПОВС следует учитывать возможности выполнения с ее помощью сложных вычислений. В этой связи для эффективной организации вычислений в ПОВС необходимо обеспечить выполнения следующего условия: $T(L) \approx T(B_r^{(i)})$, где $T(L)$ $T(B_r^{(i)})$ – время выполнения ОПР и базовых операций соответственно, $r=\overline{1,N^{(i)}}$, $i=\overline{1,N}$. К структурам, удовлетворяющих данному условию, относятся ПОВС, основанные на конвейерном методе обработки информации.

Унификация вычислений в конвейерных ПОВС определяет необходимость специальных преобразований для оптимального отображения операций из заданного набора в эквивалентную им ОПР.

Методика этих преобразований включает следующее:

– выделение ОПР из заданного набора операций;

– приведение ОПР к виду, удобному для реализации в ПОВС;

– минимизацию требуемой избыточности вычислений;

– реализацию ОПР в ПОВС.

Операции, выполняемые в ПОВС, приводятся к одной из следующих схем вычислений:

$$\sum_{i=1}^{n} A_i \prod_{j=1}^{n-i+1} \frac{X_j}{Y_j} \quad (1),$$

$$\sum_{i=1}^{n} \prod_{j=1}^{k} \frac{X_{i,j}}{Y_{i,j}} \quad (2),$$

$$\prod_{i=1}^{n} \sum_{j=1}^{k} \frac{X_{i,j}}{Y_{i,j}} \quad (3),$$

$$B_0 + \bigcup_{i=1}^{n} \frac{A_i}{B_i} \quad (4),$$

где символы $\bigcup_{i=1}^{n} \frac{A_i}{B_i}$ определяют сокращенную запись цепной дроби.

Реализация схемы вычислений (1), (2), (3) или (4), наиболее близкой по структуре к выбранным ОПР, осуществляется на основе конвейерных ПОВС, допускающих организацию вычислений в автономном, полуавтономном и неавтономном режимах [3].

4. Заключение

Рассмотренная организации вычислений позволяет процесс выполнения операций сделать однотипным и легко реализовать его аппаратурными средствами. Использование конвейерных процессоров, настроенных на оптимальное выполнение ОПР, соответствующих специфике конкретного применения, существенно повышает производительность и уровень внутренних языков конвейерных ПОВС.

Это обусловлено, прежде всего тем, что реализация ОПР на уровне машинного языка позволяет:

–осуществить реализацию многоместных операций;

– сократить объемы оперативной памяти для представления программ;

– сократить количество обращений к памяти, необходимых для реализации соответствующих алгоритмов обработки данных;

– управление выполнением операций свести к формированию начальных условий.

Недостатком унификации вычислений на основе ОПР является некоторое недоиспользование процессорных ресурсов ПОВС, так как данный подход предполагает избыточность применительно к конкретным вычислениям. Однако благодаря успехам в области интегральной технологии и возможности уменьшения избыточности за счет расширения набора ОПР указанный недостаток можно считать несущественным.

Таким образом, введение ОПР во внутренний язык конвейерных ПОВС создает предпосылки для дальнейшего повышения производительности вычислительных средств, расширения средств внутренних языков, обеспечения наиболее эффективного использования оперативной памяти и существенного упрощения вопросов организации вычислений.

Список литературы

1. Представление задач в системах распараллеливания с изменяемой зернистостью/ Г. М. Луцкий, С. Г. Стиренко, А. И. Зиненко, Д. В. Грибенко // Вісн. Нац. техн. ун-ту України "КПІ". Сер. Інф-ка, упр. та обчисл. техніка. - 2012. - Вип. 55. - С. 11-17.

2. Самофалов К. Г., Луцкий Г. М. Основы теории многоуровневых конвейерных вычислительных систем Радио и связь Москва 1989 271 с.

3. Самофалов К. Г., Луцкий Г. М. Основы построения конвейерных ЭВМ. Киев: Вища шк. Головное изд-во. 1981. 224 с.

УДК 004.04

КАЛЮЖНИЙ І.Г.,
ДУБІНСЬКИЙ Є.В.

МЕТОД ТА ЗАСОБИ ВЕРИФІКАЦІЇ ТА ВІДОБРАЖЕННЯ СТАТИСТИЧНИХ ДАНИХ В КАРТОГРАФІЧНОМУ ФОРМАТІ

В даній статті було розглянуто метод верифікації картографічних даних з послідуючим відображенням на мапі на прикладі сервісу для оренди нерухомості. Для прикладу була обрана локація міста Києва та всіх його районів. Також було проведено порівняльну характеристику засобів досягнення даної мети та їх опис. Засоби відображення та верифікації порівнювались по параметрам швидкодії і простоті у використанні.

This article examined the method of verification mapping data with subsequent display on a map. As the example service for rental was picked. Kyiv and all its regions location were selected as a territory example. It was also conducted comparative characterization of tools which was needed to achieve this goal. These mapping and verification tools were compared by performance and “ease to use” parameters.

1. Вступ

В даній статті зроблено огляд методу верифікації статистичних даних з послідуючим наданням їм картографічного формату і послідуючого відображення на мапі. Для демонстрації універсальності методу було обрано місто Київ та всі його регіони. Статистичні дані а також всі довготи і широти було взято з ресурсів які знаходяться у вільному доступі в мережі Інтернет.

2. Огляд засобів верифікації і відображення даних

Для збереження статистичних даних було обрано формат JSON, так як цей формат найбільше підходить для збереження даних у картографічному вигляді. Виходячи з даних умов, для верифікації даних було обрано технологію для валідації JSON Schema. JSON Schema це потужний інструмент для перевірки структури даних JSON. Детальну інформацію по даному інструменту та його використанню можна знайти в [1]. Також було обрано технологію jQuery validation plugin, який дозволяє верифікувати дані на стороні користувача. Застосунок для відображення обирався серед додатків Яндекс.Карты, 2ГИС та Google Maps. З недоліків Яндекс.Карт можна відзначити відсутність деталізації на рівні будівель, відсутність відображення у режимі 3D та слабке покриття міст і регіонів по всьому світу. Перевагами даного застосунку є найкраще покриття території Росії і наявність режиму «нічний». 2ГИС карти мають гарну деталізацію об’єктів але слабке покриття у всьому світі. Google Maps покривають весь світ на високому рівні деталізації. Так як 2ГИС і Яндекс.Карты поступаються Google Maps в більшості порівняльних характеристик (покриття світу, деталізація відображення), а універсальність і точна робота методу націлена на будь-яку територію, то було обрано Google Maps API[2] як технологію відображення картографічних даних. Детальну порівняльну характеристику наведено у табл. 1.

3. Верифікація статистичних даних та надання їм картографічного вигляду

Збір статистичних даних по нерухомості відбувається один раз на день за допомогою виконання сценарію який створено на мові PHP. Він розбирає та аналізує відповідний ресурс в мережі Інтернет. Об’єкт аналізу має бути обрано коректно і відповідно до місцевості, інакше буде колізія між даними і відображуваною територією. Після аналізу відбувається верифікація отриманих даних. Для прискорення роботи алгоритму верифікації, дані

розбиваються на лексеми, які потім одна за одною перевіряються на наявність синтаксичних помилок, наявність необхідної інформації та помилок що могли статися за рахунок людського фактору під час внесення даних. Після верифікації дані зберігаються у базі даних, попередньо набувши картографічного формату. Цей формат досягається за допомогою додавання до статистичних даних довготи і широти місцевості якій ці дані належать.

4. Метод відображення картографічних даних

На початку роботи програми відображається початкова обрана місцевість (для демонстрації була обрана територія міста Київ) поділена на райони. На областях районів відображається базова статистична інформація. Координати початкової території та його районів отримуються за допомогою сервісу openstreetmap[3], а статистична інформація завантажується в результаті роботи PHP-сценарію.

Також на головному екрані присутні налаштування по відображенню даних, наприклад можна показати найдорожчі варіанти нерухомості або приховати квартири, число кімнат у яких менше за задану кількість та ін.

При натисканні на район, мапа змінюється і відображає обраний район у повному розмірі, а інформація по нерухомості стає детальнішою. Також на користувацькому інтерфейсі присутній набір фільтрів, які дають змогу показувати лише деякі із можливих даних. Фільтри не можна конфігурувати з інтерфейсу користувача, але їх можна змінювати, видаляти чи створювати у програмному коді додатку.

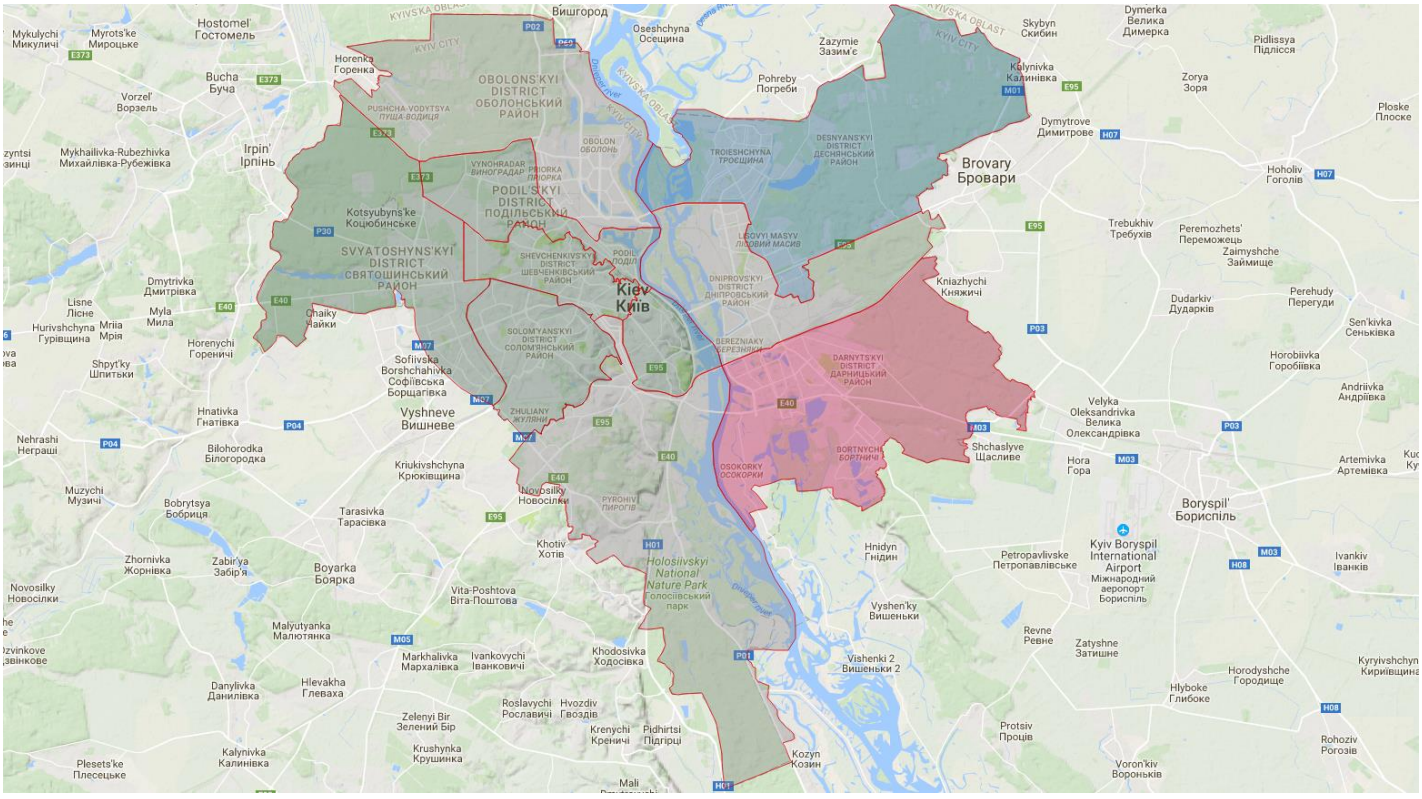

Рис. 1. Приклад інтерфейсу користувача з відображенням мапи Києва

Табл 1. Порівняльна характеристика засобів відображення картографічних даних.

Критерій порівняння	Яндекс.Карты	2ГИС	Google Maps
Покриття	Краще покриття Росії, поступається Google в покритті світу	Поступається конкурентам в покритті як в Росії, так і в інших країнах	Краще покриття всього світу
Деталізація	Хороша деталізація Росії, достатня в світі	Одна з кращих деталізацій в містах присутності	Хороша деталізація по всьому світу. На карті Росії можуть бути відсутні великі міста. У плані відображення невиразна деталізація. Об'єкти добре видно тільки при досить сильному наближенні.
Деталізація на рівні будівель	Нема	Великі торгові центри	Великі торгові центри
Можливість завантаження і використання офлайн	Є. Великий розмір даних	Є	Є. Великий розмір даних
Режим 3D	Однакова висота будівель	Є	Є
Висновок	Добре знає географію і організації по всій Росії. Наявність сервісів «Народна карта», «Панорами вулиць», голосового введення	Детальна інформація по організаціям і висока деталізація карт в містах присутності	Зручне і функціональне побудова маршрутів. Панорама вулиць, голосове введення

5. Висновки

На прикладі сервісу по нерухомості було продемонстровано метод для верифікації статистичних даних, перетворення даних у картографічний формат і відображення їх на мапі. Даний метод відображає обране місто і його райони (в даному прикладі розглядалося місто Київ) а також статистику у сфері оренди жилого приміщення. Також є можливість проглянути детальну інформацію по обраному району. На виді всього міста відображається загальна інформація по районам, а на виді детального огляду району відображається об'єкти для оренди та інформація по ним. Також є можливість фільтрації даних які відображаються. За допомогою обраних користувачем критерій виводиться специфічна інформація. Досить великий набір фільтрів (наприклад сортування по ціні, кількість кімнат, поверх, та ін.) робить систему досить гнучкою у використанні. Суть методу полягає у можливості відображати різну територію та статистику до неї представляючи необхідну інформацію. Для зміни локації відображення потрібно зробити запит з назвою бажаного міста до сервісу openstreetmap. В результаті буде завантажена інформація по обраній місцевості та її розподіл на райони. Також для коректної роботи потрібна статистика по нерухомості. В даному прикладі статистика береться з ресурсу Інтернет. Її оновлює і аналізує PHP-сценарій один раз на день. Сценарій можна налаштувати на сканування будь-якого ресурсу Інтернет. Це дозволить отримувати статистику для потрібного міста та його районів.

Список літератури

1. Michael Droettboom Understanding JSON Schema. – 2016 – С. 10 – 24.
2. Gabriel Svennerberg Beginning Google Maps API – 2010 – С. 45 – 100.
3. Yu-Wei Lin A qualitative enquiry into OpenStreetMap making – 2011.

УДК 004.048

КАПЛУНОВ А.В.
СІМОНЕНКО А.В.

ПОБУДОВА БАЄСОВСЬОЇ МЕРЕЖІ ДОВІРИ ДЛЯ КЛАСИФІКЦІЇ ЗАХВОРЮВАННЯ

У статті розглянуто практичне застосування баєсової мережі довіри для класифікації хвороби пацієнта, ґрунтуючись на симптоматиці його захворювання. Описана методика генерації структури статичних моделей баєсових мереж довіри за експериментальними вибірками значень змінних та принцип їх функціонування.

Ключові слова: баєсова мережа довіри, таблиці умовних ймовірностей, причинно-наслідкові зв'язки.

The article deals with the practical application of Bayesian believe network to classify the patient ill based on the symptoms of his disease. This workaround structure generation static models Bayesian believe network for the experiment variables and how they function.

Keywords: Bayesian believe network, table of conditional probabilities, the cause-effect relationships.

1. Вступ

Завдання класифікації представляє собою завдання віднесення зразка до однієї з декількох множин, що попарно не перетинаються. Прикладом таких завдань може бути, завдання визначення кредитоспроможності клієнта банку, рішення задач управління портфелем цінних паперів, завдання визначення життєздатних і схильних до банкрутства фірм.

Логічним наслідком цього є те, що нейронні мережі можливо використовувати з метою діагностування найбільш ймовірного захворювання пацієнта, ґрунтуючись на симптоматиці його захворювання.

Для цих цілей пропонується застосувати баєсову мережу довірі.

2. Огляд баєсової мережі довіри в загальному вигляді

Баєсова мережа довіри - використовується в областях, які можна охарактеризувати успадкованою невизначеністю. Причиною цієї невизначеності може бути:

- неповнота знань;
- випадковість характеризації завдання;
- неповнота розуміння предметної області.

Отже, баєсові мережі довіри (БМД) доцільно застосовувати для моделювання ситуацій в яких має місце певна невизначеність. Баєсові мережі довіри іноді називають причинно-наслідковими мережами, в яких випадкові події поєднані причинно-наслідковими зв'язками.

З'єднання методом причин і наслідків дозволяє відносно легко оцінити ймовірність події. У реальному світі оцінка зазвичай робиться в напрямку від "спостерігача" до "спостереження" або від "ефекту" до "наслідку". В цьому випадку провести оцінку значно важче, ніж при спостереженні від “наслідку” до “ефекту”, тобто в напрямку від наслідку.

Таким чином, баєсові мережі довіри (БМД) надають дослідникам можливість використовувати зручне графічне представлення багатовимірних статистичних розподілів. Теорема Баєса, яка лежить в основі БМД, дає можливість визначити ймовірність виконання певної події (гіпотези), коли ми володіємо лише непрямими підтвердженнями (даними), які можуть бути не точними, і визначає формулу Баєса:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

де P (A|B) - ймовірність гіпотези A при настанні події B (апостеріорна ймовірність); P (B|A) - ймовірність настання події B при істинності гіпотези A; P (A) - апріорна ймовірність гіпотези A; P (B) - ймовірність настання події B.

3. Теоретичне підґрунтя теореми Баєса

Формула Баєса дозволяє за відомим фактом події обчислити вірогідність того, що вона була викликана даною причиною. Події, які є наслідками дії «причини», називаються гіпотезами, оскільки вони є передбачуваними подіями, що викликані даними. Безумовна ймовірність справедливості гіпотези називається апріорною (яка ймовірність причини взагалі), а умовна з урахуванням факту настання подія - апостеріорною (наскільки ймовірною виявилися причина обліку даних про подію) [1].

Важливим наслідком формули Баєса є формула повної ймовірності події, що залежить від декількох несумісних гіпотез:

$$P(B) = \sum_{i=1}^{N} P(A_i)P(B|A_i),$$

де P (A_i) - ймовірність і-ї гіпотези; P ($B|A_i$) - ймовірність настання події B при істинності гіпотези A.

Баєсові мережі довіри являють собою сп рямований ациклічний граф, що має насту пні властивості:

- кожну вершину представляють як п одію, яку можна описати випадковою вели чиною, і що може мати декілька станів;
- вершини які мають "батьківські" ви значені у таблиці умовних ймовірностей (Т УЙ) або у функції умовних ймовірностей (ФУЙ);
- вершини без "батьків", мають безум овні (маргінальні) ймовірності своїх станів .

Тобто, у баєсових мережах довіри вершини є відображенням випадкових змінних, а дуги - ймовірнісних залежностей, які визначаються через таблиці умовних ймовірностей. Таблиця умовних ймовірностей кожної вершини містить інформацію про ймовірність стану цієї вершини в залежності від стану її "батьків".

Якщо на графі БМД ребро виходить з вершини A у вершину B, то вершину A називають батьківською по відношенню до

В, а В - нащадком по відношенню до А. Спільний розподіл значень в вершинах можна зручно розписати як результат локальних розподілів у кожному вузлі і в його предків:

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^{n} P(X_i | par(X_i)),$$

де X_i - ймовірність події в вершині і; par (X_i) - безліч предків вершини X_i.

Якщо у вершини Хі немає "батьків", то її локальний розподіл ймовірностей називають безумовним, у іншому випадку він є умовним. Якщо значення в вузлі отримано в результаті досвіду, то вершину називають свідком (позначену вершиною подію - свідченням).

Таким чином, БМД дозволяють точно формалізувати причинно-наслідкові зв'язки, які відбуваються всередині будь-якої системи, в тому числі при діагностування найбільш ймовірного захворювання пацієнта, ґрунтуючись на симптоматиці його захворювання.

4. Аналіз структури взаємозв'язків показників БМД

Основним завданням при побудові БМД, що моделює реальний процес, є встановлення структури взаємозв'язків показників. При цьому результатом спостереження таких процесів є набір значень експериментальних даних (навчальна вибірка):

$$D=\{d_1,d_2,\dots,d_n\},d_i=\{x_{i1},x_{i2},\dots,x_{iN}\},$$

де d_1, d_2, ..., d_n - спостереження за станом показників; n - кількість спостережень; x_{i1}, x_{i2}, ..., x_{iN} - стан змінних показників його спостереження; N - кількість показників, що беруть участь в спостереженні.

Оскільки повний перебір всієї безлічі нециклічних моделей, і вибір з них найбільш відповідної навчальної вибірки, має NP-важку обчислювальну складність - для вирішення такого завдання застосовувати його не є доцільним.

Таким чином, побудова баєсової мережі необхідно проводити евристичним методом. Для цього в першу чергу необхідно визначити структуру взаємозв'язків вершин, встановивши їх взаємний вплив в контексті аналізованого процесу. Другим етапом побудови є навчання мережі - тобто привласнення ваг взаємозв'язкам, встановлення умовних ймовірностей.

Для вирішення першого завдання, з огляду на специфіку і складність поставленого завдання, одним з кращих варіантів є побудова структури баєсової мережі з використанням досвіду експертів в даній області знань. Збір думки експертів про взаємні зв'язки показників і їх вплив на надійність проводиться у вигляді анкетування та спільної побудови графічної моделі за результатами анкетування. Однак варто враховувати, що людський фактор негативно впливає на точність оцінки, і, незважаючи на достатню кількість методик підвищення об'єктивності отриманих даних, вони можуть використовуватися лише для побудови чернетки графа взаємозв'язків баєсової мережі [2].

5. Оцінка ступеня взаємозв'язку пар змінних

Для оцінки ступеня взаємозв'язку пар змінних в безлічі показників надійності зручно використовувати значення обопільної інформації, яке пов'язане з поняттями ентропії та умовної ентропії. Для дискретної випадкової величини ентропія з функцією розподілу p (x) = p (X - x) визначається як

$$H(X) = -\sum_x p(x) \log\ p(x).$$

У свою чергу, умовна ентропія для пари дискретних випадкових величин Y і X при відомому X визначається як

$$H(Y|X) = \sum_x p(x) H(Y|X = x) =$$

$$= -\sum_x p(x) \sum_y p(y|x) \log\ p(y|x) =$$

$$= -\sum_x \sum_y p(x,y) \log\ p(y|x)$$

Отже, для випадків коли Y пов'язаний з X функціональною залежністю, H (У | X) = 0. Для всіх x: p (x)> O, значення Y визначається як y = g (x), з ймовірністю p (y | x) = 1.

Так звана mutual information показує кількість інформації, яку одна випадкова змінна містить про іншу. Для двох випадкових змінних X і Y зі спільною функцією розподілу p (x, y) обопільна інформація MI (X, Y) - це відносна

ентропія, фактично аналог кореляції, але для інформаційних показників [3]

$$MI(X,Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)p(y)} =$$

$$= H(Y) - H(Y|X) = H(X) - H(X|Y)$$

MI = 0 в тому випадку, коли оцінювані показники (вершини графа БМД) є незалежними одна від одної, так як

$$p(x,y) = p(x)p(y),$$

$$\log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x)p(y)}{p(x)p(y)} = \log 1 = 1.$$

Таким чином, MI > 0 показує наявність обопільної інформації, або інформаційної залежності, між вершинами. Підрахувати p (x) для кожної з вершин за наявною експериментальною вибіркою нескладно, при зростанні обсягу даних точність збільшується.

У разі, якщо БСД складається з N вершин, для обчислення MI між усіма парами вершин потрібно виконати (N (N-1)) / 2 обчислень, при цьому MI (x, y) = MI (y, x).

6. Висновок

Отримані за допомогою обчислень результати спільно з побудованим за результатами експертного аналізу «каркасом» БСД дозволяють отримати коректний граф, що відображає статистичні взаємозв'язки показників з надійністю ПЗ.

Початкові умовні ймовірності для вершин задаються на підставі рекурсивної формули. Їх коригування на більш пізніх етапах можливе, коли в результаті збору показників виходить нова повністю заповнена вибірка d_1. В ході подальшої роботи мережі граничні маргінальні вершини, відповідно до вимірюваних в ході оцінки метрик, максимізують значення ймовірностей, що відповідають вимірюваним показникам (тобто для вершини і $p_i(x_j) = 1$ у разі, коли $X_i = x_j$).

Таким чином, використовуючи описану методику побудови і навчання баєсової мережі, для отримання об'єктивної оцінки діагностування найбільш ймовірного захворювання пацієнта, ґрунтуючись на симптоматиці його захворювання, дослідникам залишається лише зібрати достатню кількість експериментальних даних. Згенерована модель дозволить отримувати прогнози надійності високого ступеня довіри і об'єктивності.

Список літератури

1. Heckerman D. A tutorial on learning with Bayesian networks. The MIT Press, Cambridge, Massachusetts, 1998.

2. Kahneman D., Slovic P., Tversky A. Judgment Under Uncertainty: Heuristics and Biases. Cambridge: University Press, Cambridge, 1982.

3. Chow C.K., Liu C.N. Approximating discrete probability distributions with dependence trees // IEE Transactions on information theory. Vol. IT–14. № 3. 1968.

УДК 004.056.53

КУКСА В. В.
ЖАБІН В. І.,

МЕТОД ВИЯВЛЕННЯ АТАК НА ОСНОВІ СТАТИСТИЧНОГО АНАЛІЗУ ПОКАЗНИКІВ ЗАВАНТАЖЕННЯ СИСТЕМИ

У даній статті розглянуто виявлення вторгнень із використанням статистичного аналізу показників завантаження системи. Досліджено підхід до підвищення точності виявлення аномальної поведінки із застосуванням результатів одночасного спостереження декількох різних показників завантаження системи. Була показана ефективність запропонованого методу для виявлення певних типів атак.

The subject of this paper is an intrusions detection based on statistical analysis of the system load characteristics. The approach to increase the accuracy of abnormal behavior detection using the simultaneous observation of several different system load characteristics was studied. The proposed method was shown to be efficient at detection of certain types of attacks.

1. Вступ

Важливість проблеми захисту інформації зростає із стрімким розвитком комп'ютерних технологій. Впровадження комп'ютерних мереж у таких сферах, як фінансова, виробнича, охорони здоров'я, роблять комп'ютерні системи привабливою ціллю для атак і великою вразливістю для суспільства.

Традиційно, захист комп'ютерних систем будується із застосуванням таких засобів як, шифрування, мережеві екрани (файрволи), віртуальні приватні мережі і т. п., але цього недостатньо для забезпечення повної безпеки. У зв'язку з цим постає необхідність у розробці динамічних засобів, що здатні вести постійний аудит системи і реагувати на несанкціоновану діяльність. Для вирішення цієї задачі були створені системи виявлення атак (СВА).

Дані які вказують на несанкціоновану діяльність можуть надходити із широкого різноманіття джерел: мережеві служби, журнали подій операційної системи (ОС) і користувацького програмного забезпечення (ПЗ) і т. і. Об'єм цих даних може бути дуже великим, а методи їх аналізу — мати високу обчислювальну складність. Тому постає проблема вибору найбільш ефективних джерел вхідних даних для СВА.

У даній роботі наведена характеристика сучасних типів СВА, проаналізовані потенційні проблеми джерел даних для СВА. Запропоновано підхід для підвищення точності виявлення атак на основі даних про рівень завантаження системи до задовільного рівня.

2. Типи систем виявлення атак

В залежності від джерела вхідних даних, розрізняють два основних типи систем виявлення атак: хостові та мережеві.

Хостові системи виявлення атак (ХСВА) встановлюються на окремі машини. СВА цього типу аналізують різні аспекти поведінки захищеної системи, такі як: мережеві з'єднання хоста (RealSecure Agent), цілісність файлової системи (Tripwire, AIDE), журнали подій (LogSentry), взаємодія програмного забезпечення із ядром операційної системи [1].

При своїй високій ефективності хостові СВА мають і суттєві недоліки. Перший недолік випливає з того, що СВА встановлено безпосередньо на хості, що може бути атакований, а отже, джерела інформації, якими користується СВА, можуть бути скомпрометовані. Наприклад, техніка приховування системних викликів описана у [2]. Якщо СВА працює не в реальному часі, засоби знищення системних журналів та інших слідів втручання, що входять до складу сучасних інструментів проникнення, таких як Metasploit Framework, також дозволяють уникнути виявлення атаки [3]. Іншим недоліком існуючих хостових СВА є високі витрати ресурсів захищеної системи на збір інформації (журналювання подій, перехоплення системних викликів, трафіку тощо) і її обробку.

Мережеві системи виявлення атак (МСВА) спираються на дані про використання локальної мережі. МСВА збирають і аналізують усі пакети, що надійшли на інтерфейс шлюза або мережевого адаптера, що працює у нерозбірливому режимі (захоплює усі пакети, а не лише ті, що призначені для нього) [1]. Так як МСВА обробляє не тільки пакети, адресовані певному хосту, вона забезпечує захист цілому сегменту мережі [1].

Розгортання і адміністрування МСВА простіше за ХСВА, але платою за це стає обмеження інформації, доступної системі для аналізу, лише мережевим трафіком. Об'єм трафіку, що протікає у сучасних локальних мережах, сягає від десятків мегабіт до десятків гігабіт за секунду і аудит всіх пакетів вимагає значних обчислювальних ресурсів. Вибірковий же аналіз пакетів збільшує шанси на пропуск проявів атаки. Із впровадженням шифрування трафіку МСВА або втрачають можливість аналізу вмісту пакетів, або

повинні розшифровувати трафік, створюючи тим самим нову потенційну точку атаки на шифрований трафік.

При розгляді сучасних типів систем виявлення атак і використовуваних ними джерел даних були виявлені певні проблеми із їх застосуванням. Особливо варто відзначити проблему достовірності і доступності вхідних даних. Також, відкриття таких джерел інформації, як наприклад шифрованого трафіку у мережі для СВА, робить саму СВА додатковим вразливим місцем системи.

3. Показники завантаження системи

Усі сучасні операційні системи (Windows, UNIX-подібні) у процесі виконання своїх функцій, отримують у розпорядження вичерпну інформацію про завантаженість усіх апаратних і програмних компонентів.

Для вирішення нашої задачі було обрано наступні показники завантаження: час центрального процесора (ЦП), використаний процесом на рівні користувача, час ЦП, використаний процесом на рівні ОС (виконання системних викликів), об'єм пам'яті процесу, кількість потоків процесу, кількість байтів, зчитаних з диску і записаних на диск, кількість файлів, відкритих процесом, кількість вхідних і вихідних мережевих пакетів для кожного сокета, відкритого процесом.

4. Набори даних

У дослідженні використовувалась дані, зібрані під час роботи загальновикористовуваних програм, різних за розміром і складністю, як в умовах нормального використання, так і при різних типах втручання («відмова в обслуговуванні», «виконання стороннього коду»). Інструменти втручання було взято із бази даних вразливостей https://www.exploit-db.com.

Кожен набір даних являє собою послідовність записів із значеннями вибраних показників завантаження системи одним процесом від початку його виконання до завершення, зафіксованими через рівні проміжки часу в 1 секунду.

Дані для сервера баз даних MySQL були зібрані для нормальної поведінки під змінним навантаженням, що симулювалося за допомогою утиліти mysqlslap [4], та при втручанні типу "виконання стороннього коду" під час роботи під змінним навантаженням. Вразливість, що використовувалась для втручання, зареєстрована під ідентифікатором CVE-2016-6662.

Збір даних для сервера додатків Apache Tomcat виконувався аналогічним чином. Для симуляції навантаження застосовувалася утиліта siege [5]. Втручання у роботу сервера було виконано із використанням вразливості типу "відмова в обслуговуванні". Вразливість зареєстрована під ідентифікатором CVE-2014-0050.

Для задачі навчання методів моделювання було використано приблизно 50% даних нормальної поведінки обох програм.

5. Побудова моделі нормальної поведінки

Метод, що пропонується, спирається на дані, які складно підробити у разі проникнення; їх генерація не вимагає змін у користувацькому ПЗ (журналювання подій) і додаткових ресурсів системи, що захищається.

Показники завантаження несуть багато інформації про стан процесу, але складно визначити, чи є зміна окремих показників нормальною чи аномальною, адже вона може залежати від дуже великої кількості умов. Наприклад, використання ЦП і пам'яті веб-сервером різко зростає при надходженні запитів від клієнтів і знижується майже до нуля при відсутності запитів.

Проте, спостереження дозволяють встановити, що співвідношення між певними показниками залишаються незмінними при нормальній роботі програми. Виходячи з цього, для побудови моделі нормальної поведінки програми, необхідно знайти зв'язані показники і визначити характер співвідношення між ними.

У математичній статистиці наявність і сила взаємозв'язку величин характеризується кореляцією і її мірою є коефіцієнт кореляції r [6]. У нашому випадку будемо шукати кореляцію між послідовностями значень пар показників завантаження системи програмою у часі. Так як усі показники є кількісними змінними, застосуємо лінійний коефіцієнт кореляції [6]. Перед побудовою моделі обирається порогове значення коефіцієнта кореляції $r_{\text{п}}$,

при перевищенні якого вважається, що між парою показників існує зв'язок.

Виявивши зв'язані показники, необхідно знайти характер залежності між ними. Для спрощення вважатимемо залежність між ними лінійною, а отже, можемо застосувати метод найменших квадратів для обчислення коефіцієнтів лінійної регресії [7].

Отже, база даних нормальної поведінки програми складається із пар пов'язаних показників і коефіцієнтів лінійної регресії, що описує зв'язок між ними.

6. Тестування поведінки програми

У процесі тестування через задані інтервали на основі поточного значення кожного показника у базі даних нормальної поведінки обчислюється значення відповідного пов'язаного показника. Для різниці між реальним значенням показника і обчисленим за даними бази нормальної поведінки встановлене порогове значення Δ, при перевищенні якого реєструється неспівпадіння. Оскільки показники можуть мати дуже різні границі виміру, їх значення нормується із урахуванням цих границь, що є або фіксованими (використання ЦП), або можуть бути отримані від ОС (об'єм пам'яті, максимальна кількість відкритих файлів), або визначені емпірично в процесі формування профілю нормальної поведінки. Завдяки цьому, можна задати єдине порогове значення Δ.

На кожному кроці перевірка однієї пари показників вимагає константного часу. Отже, загальний час перевірки на кожному кроці пропорційний $O(P)$, де P - кількість пар зв'язаних показників у базі даних.

Для боротьби із хибно позитивних результатів необхідно враховувати декілька неспівпадінь. Як правило, під час реальних атак аномальна активність проявляється на коротких проміжках часу. Тому мірою аномальності поведінки було обрано кількість неспівпадінь n на фіксованому проміжку часу t. В кожен момент тесту підтримується інформація про те, скільки неспівпадінь спостерігалося за минулий проміжок часу фіксованої тривалості.

Для міри аномальності встановлюється поріг N, до якого поведінка програми вважається нормальною. Щоразу, коли міра досягає або перевищує поріг, фіксується атака. Поріг різниці між очікуваним значенням показника і реальним Δ і міри аномальності N є головними регуляторами чутливості методу. Нижчі порогові значення ведуть до збільшення успішно виявлених атак, але також і збільшення кількості хибно позитивних спрацьовувань, і навпаки, вищий поріг зменшує кількість як істинно позитивних так і хибно позитивних результатів.

7. Результати дослідження

Ми протестували кожен із трьох методів моделювання на кожному з наборів даних при декількох різних порогах чутливості. Хибно позитивні результати підраховувались для частини даних нормальної поведінки, не використаної в процесі навчання, а істинно позитивні – для записів аномальної поведінки.

Метод показав дуже відмінні результати на кожному з наборів даних. На рисунку 1 представлені середні результати для кожної комбінації набору даних порогу чутливості. Вісь ординат на рисунку 1 показує загальну здатність методу до виявлення аномалій. Вісь абсцис показує частоту хибно позитивних результатів. Частота хибно позитивних результатів визначена як відношення кількості хибно позитивних результатів до загальної кількості вимірів за час тестування нормальної поведінки, і її значення лежить у діапазоні від 0 до 1.

Невелика частка виявлених атак типу "виконання стороннього коду" пояснюється ти, що зловмисницький код не впливає суттєво на функціювання ураженої програми. В той же час, точність виявлення атак типу "відмова в обслуговуванні" є задовільною, оскільки порушення основної функції програми внаслідок вторгнення веде до суттєвих змін у споживанні нею системних ресурсів.

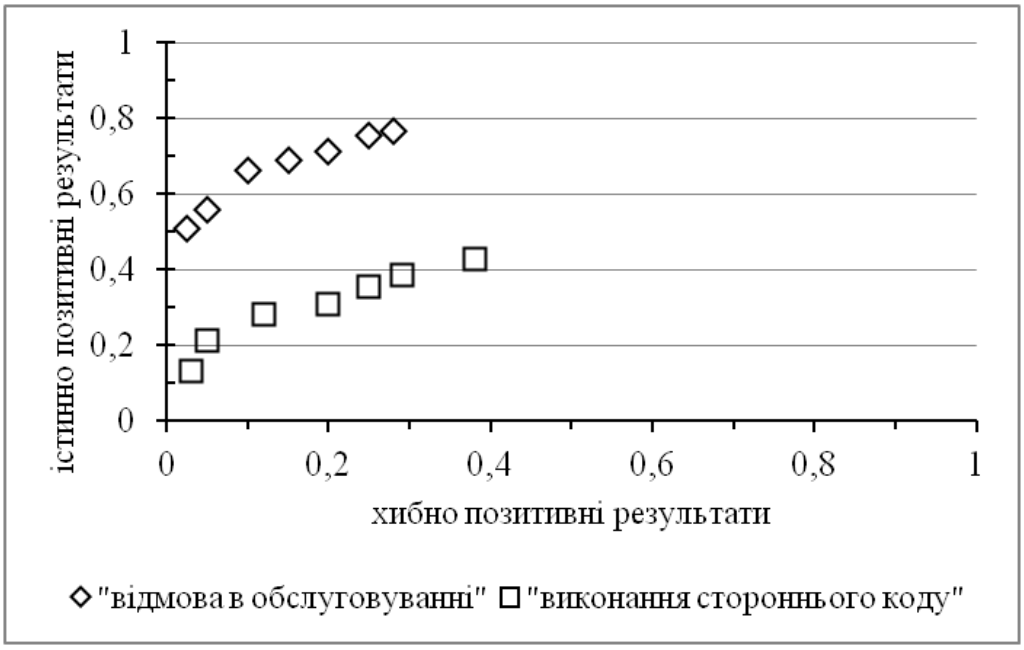

Рис. 1. Об'єднані результати для типів атаки «відмова в обслуговуванні» і «виконання стороннього коду»

8. Висновки

Запропонований підхід до використання показників завантаження системи для вирішення задачі виявлення вторгнень має наступні переваги: висока достовірність і доступність вхідних даних, висока швидкість обробки даних та незалежність методу від типу комп'ютерної системи та її застосування.

Розроблений метод був протестований на наборах даних, що відповідають різним програмам і технікам вторгнення. Метод показав невисоку ефективність проти атак типу "виконання стороннього коду". У той же час, була досягнута задовільна точність виявлення атак типу "відмова в обслуговуванні", що пояснюється суттєвими змінами у споживанні атакованою програмою системних ресурсів.

Таким чином, запропонований метод дозволяє підвищити точність виявлення певних типів вторгнень на основі показників завантаження системи.

Список літератури

1. Kazienko P. Intrusion Detection Systems (IDS) Part 2 – Classification; methods; techniques [Електронний ресурс] / P. Kazienko, P. Dorosz // TechGenix Ltd.. – 2004. – Режим доступу до ресурсу: http://techgenix.com/ids-part2-classification-methods-techniques/.

2. Srivastava A. System Call API Obfuscation (Extended Abstract) / A. Srivastava, L. Andrea, G. Jonathon // Recent Advances in Intrusion Detection – 11th International Symposium, RAID 2008, Cambridge, MA, USA, September 15-17, 2008, Proceedings / A. Srivastava, L. Andrea, G. Jonathon. – Berlin Heidelberg: Springer-Verlag, 2008. – (Security and Cryptology; т. 5230). – С. 421 – 422.

3. Event Log Management [Електронний ресурс] // Offensive Security. – 2015. – Режим доступу до ресурсу: https://www.offensive-security.com/metasploit-unleashed/event-log-management/.

4. mysqlslap — Load Emulation Client [Електронний ресурс] // Oracle Corporation. – 2017. – Режим доступу до ресурсу: https://dev.mysql.com/doc/refman/5.7/en/mysqlslap.html.

5. Fulmer J. Siege Manual [Електронний ресурс] / Jeffrey Fulmer. – 2012. – Режим доступу до ресурсу: https://www.joedog.org/siege-manual/.

6. Собственно-корреляционные параметрические методы изучения связи. Оценка существенности корреляции / Р. А.Шмойлова, В. Г. Минашкин, Н. А. Садовникова, Н. А. Шувалова // Теория статистики / Р. А.Шмойлова, В. Г. Минашкин, Н. А. Садовникова, Н. А. Шувалова. – Москва: Финансы и статистика, 2004. – С. 361–376.

7. Парная регрессия на основе метода наименьших квадратов и метода группировок / Р. А.Шмойлова, В. Г. Минашкин, Н. А. Садовникова, Н. А. Шувалова // Теория статистики / Р. А.Шмойлова, В. Г. Минашкин, Н. А. Садовникова, Н. А. Шувалова. – Москва: Финансы и статистика, 2004. – С. 333–342.

УДК 004.75

МАРТИНЮК Р.О.
ЛУЦЬКИЙ Г.М.
ВОЛОКИТА А.М.
РОТЕНБЕРГ О.В.

СПОСІБ РОЗПОДІЛЕНИХ ОБЧИСЛЕНЬ З ВИКОРИСТАННЯМ WEB БРАУЗЕРІВ

В даній роботі описано використання WEBбраузерів для створення системи розподілених обчислень на основі технологій JavaScript та AJAX. Описане рішення має дві основні переваги: воно не потребує встановлення додаткових програм для обчислення, працюючи у фоновому режимі, тим самим не заважаючи клієнту.

This article is describes how to use WEB browsers to create a system of distributed computing through technology JavaScript and AJAX. The described solution has two main advantages: it does not require additional software to calculate and working in the background without disturbing a client.

Ключові слова :розподілені обчислення, web-браузер, клієнт-серверна архітектура.

1. Вступ

Розподілені обчислення – це спосіб, який використовують для розв'язання задач, котрі можливо розділити на декілька частин, названих підзадачами; кожна з них вирішується окремо, зазвичай за допомогою незалежних комп'ютерів або процесорів. Після завершення обчислень збираються результати, щоб сформувати остаточне рішення первинної задачі.

Більшість проектів, на основі розподілених обчислень, були розроблені протягом останніх років. Це, наприклад, проект Folding@home Хімічного факультету Стенфордського університету, або SETI@home Лабораторії Космічних досліджень Каліфорнійського університету Берклі, також серед таких проектів LHC@home [1].

Загальний аспект проектів, описаних вище, це використання клієнт серверної архітектури, для якої необхідно встановлювати додаткове ПО на комп'ютер користувача (клієнта).

Робота, яка описана в даній статі, презентує розробку системи розподілених обчислень на основі клієнт-серверної архітектури, яка використовує велику кількість клієнтів, з'єднаних через мережу Інтернет. Основною інновацією системи є ідея реалізації розподілених обчислень без встановлення будь-якого додаткового ПО на стороні клієнта, а з використанням лише прямих можливостей web-браузера. Ми використовуємо клієнтський web-браузер для того, щоб виконувати обчислення під час роботи користувача. Код, який описує реалізацію підзадачі на стороні клієнта, вбудований у web-сторінку завдяки технологіям AJAX та JavaScript. При кожному відвідуванні сайту, користувачі будуть виконувати певні розрахунки у фоновому режимі[1]. На даний момент web-браузери не дозволяють напряму контролювати завантаженість системи на стороні клієнта, без встановлення додаткових розширень.

Головна мета цієї роботи – перевірити фактичну реалізацію цього підходу і виміряти можливості архітектури в процесі виконання розподілених обчислень на великій кількості клієнтів.

2. Реалізація архітектури системи

Система розподілених обчислень, яку ми розробили, основана на використанні web-браузера користувачем. Його класичний опис в цьому контексті більш широкий: браузер – це не лише програма для перегляду веб-сторінок, але і інструмент, який має можливість виконувати обчислення задач.

Завдяки технологіям JavaScript та AJAX клієнту не потрібно встановлювати додаткове ПО, так як код, для виконання задач, він буде отримувати з серверу. Таким чином, число потенційних клієнтів набагато більше порівняно з випадком, коли клієнт встановлює спеціальні програми для комунікації з сервером[3].

Під час перегляду сторінки, користувач, завдяки технології AJAX, у фоновому режимі отримує код підзадачі, котра почне виконуватись, і по завершенню відішле результат на сервер. Додаток має збережені на сервері дані та збережену на стороні клієнта логіку, тому важливим етапом є комунікація клієнта з сервером, передача даних.

Сучасні web-браузери пропонують декілька інструментів для зберігання інформації на стороні клієнта. Крім стандартного SessionStorage, ми можемо використовувати WebWorkers, LocalStorage, IndexedDB, WebSQL. Разом це приблизно 50 мб пам'яті, в котрій ми можемо зберігати інформацію для наших обчислень, на стороні клієнта.

Архітектура системи побудована за принципом клієнт-сервер. Клієнти, вбудовані у веб-сторінку, надсилають запити на отримання підзадач, і відправляють результати на сервер. Цей процес повторюється при активному підключенні користувача, у нашому випадку при відвідуванні сайту. На стороні сервера вирішується декілька задач: розділення первинної задачі на підзадачі, отримання результатів від клієнтів, консолідація результатів, формування комплексного рішення та моніторинг статусу підзадач, що враховують відключення клієнтів, та не правильне виконання.

Задачі для клієнта необхідно формувати на мові JavaScript. Використовуючи можливості AJAX та XMLHttpRequest, клієнт звертається до серверу щоб отримати скрипт, в якому міститься код рішення для отриманої підзадачі.

У свою чергу сучасні браузери реалізують підтримку технології WebWorkers, яка дозволяє виконувати код в декількох потоках, тобто виникає можливість паралельного виконання задач на стороні клієнта. Для цього, під час ініціалізації, отримаємо кількість процесорів, і, якщо можливо, розділимо підзадачу на менші частини, котрі будуть оброблювати WebWorkers. Після завершення роботи кожного з них, в головному потоці збираємо результат, опрацьовуємо його, і відправимо на сервер.

У випадку, якщо браузер не підтримує технологію WebWorkers, необхідно контролювати виконання задачі в одному потоці, не використавши весь ресурс процесора, і дати користувачу можливість продовжити роботу, без затримок.

Сторона сервера займається розподіленням задач на підзадачі, комунікацією з клієнтами через http-протокол для передачі та отримання результатів, їх обробку. Логіка сервера розділена на сервіси, з котрими працюють клієнти через GET та POST методи, використовуючи AJAX. Сервіси роблять запити до бази даних для отримання параметрів підзадач та збереження результатів. Статус виконання задач, їх результати, та дані користувачів необхідно зберігати для моніторингу статистики, прогресу, кількості невиконаних підзадач, та рейтингу клієнтів.

3. Приклад реалізації задачі

Задача, для якої була проведена оцінка архітектури:

• Факторизація числа методом Полларда:

Даний ρ-алгоритм не є найефективнішим на даний момент, але його можливо розділити на незалежні субзадачі, що дає можливість запустити його на нашій системі для її оцінки.

Реалізація алгоритму має наступний вигляд:

```
function Pollard (N){
var x = random(1, N-2);
var y = 1; var i = 0; var stage = 2;
  while (NOD, Math.abs(x - y)) == 1){
   if (i == stage ){
    y = x;
    stage = stage*2;
   }
   x = (x*x - 1)%N;
   i = i + 1;
  }
  return NOD (N, abs(x-y));
}
```

Дана функція є однаковою для всіх клієнтів, але початкове число x, генеруємо не випадковим чином, а отримуємо набір з серверу. У такому випадку сервер контролює, щоб згенеровані набори не повторювались для клієнтів.

Уявимо, що є P однакових клієнтів. Якщо ми використовуємо P різних послідовностей F(x), то ймовірність того, що перші kчисел в цих послідовностях будуть різні по модулю p буде

$\exp(-k^2P/2p)$. Таким чином максимальне прискорення можна оцінити як $P^{1/2}$ [4].

У свою чергу підзадача, котру ми отримуємо з сервера, може бути виконана в декілька потоків. На стороні клієнта ми можемо реалізувати це за допомогою технології WebWorkers. Для реалізації необхідно виділити JavaScript файл, з необхідною функцією. Він буде виконуватись у незалежному потоці, і після завершення, результат буде переданий до головного потоку.

Деякі ітерації можуть займати багато часу, і не матимуть можливості бути виконаними за одну сесію користувача. У даному випадку необхідно зберігати статус виконання, та дані на стороні клієнта. Сучасні браузери дають змогу використовувати WebSQL - базу даних, яка може зберігати необхідну для нас інформацію.

Таким чином, клієнт отримує від сервера скрипт з кодом функції, та набір даних. Для кожного потоку буде виділятись даний код, та частина даних. Після виконання всіх потоків, результат буде зібраний, і надісланий на сервер.

4. Метрики та результати

В даному розділі представлені метрики, котрі дають змогу вести оцінку як задачі, так і самої системи. Виділимо наступні характеристики:

D : розмір запиту на отримання та надсилання даних

t_D : час на виконання D запиту

t_{calc} : час виконання підзадачі клієнтом

Завдяки вищезгаданим характеристикам можна визначити деякі метрики для того, щоб оцінити продуктивність системи в різних ситуаціях і вирішити в яких випадках доцільне застосування додатку.

Однією з найголовніших метрик є коефіцієнт ефективності обчислення підзадач на стороні клієнта T_c (2). Якщо коефіцієнт близький до одиниці, або навіть більше його, то застосовувати дану систему для вирішення підзадачі неефективно, так як час передачі інформації більший за час обчислення:

$$T_c = \frac{t_D}{t_{calc}} \quad (2)$$

Даний приклад архітектури обмежується одним сервером, і відповідно всі клієнти, котрих може бути багато тисяч, або мільйонів, ведуть обмін даними з ним в онлайн режимі. Тому необхідно розраховувати навантаження на мережу (3), щоб зменшити час черги на отримання даних. Для кожної задачі необхідно ввести параметр обрахунку кількості пакетів даних, до часу виконання задачі D_s. Якщо складність задач не є лінійною, і чим більше даних, тим більший час виконання, то ймовірно, збільшення кількості переданих вхідних даних, може збільшити коефіцієнт, що зменшить навантаження на мережу, але збільшить час виконання підзадачі на стороні клієнта, і відповідно зменшить ймовірність її виконання.

$$D_s = \frac{D}{t_{calc}} \quad (3)$$

При розподіленні задачі на підзадачі важливою характеристикою є час(4), за який головна задача повинна бути виконана, так званий T_d - дедлайн реального часу. Позначимо кількість підзадач як N. Таким чином можемо вивести залежність, як:

$$T_d = (t_D + t_{calc}) * N \quad (4)$$

Дана залежність є ідеальною, адже тут не враховується кількість відмов, і час виконання підзадачі для всіх клієнтів рівний. В реальних умовах, кожен клієнт має свою продуктивність, і час виконання може бути різний. В такому випадку необхідно ввести певний коефіцієнт k_d, який буде змінювати час (5), виданий на обчислення. Також необхідно оновлювати статус підзадач, котрі були отримані клієнтами, але не виконані. Необхідно ввести певний максимальний час виконання T_m, після проходження якого підзадача повертається у статус невиконаної.

$$T_m = (t_D + t_{calc}) * k_d \quad (5)$$

Індекс T_c вказує на відношення між часом передачі даних та часом обчислення задачі. Чим вище коефіцієнт, тим гірше система підходить для вирішення задачі. Адже час передачі даних перевищує час виконання підзадач.

Нарешті коефіцієнт D_s показує використання даних отриманих клієнтом: оптимальне значення наближене до 0. В алгоритм факторизації Пірсона значення, даної характеристики, близькою до мінімальної, адже вхідні дані мають малий розмір, а час на їх обрахунок порівняно великий. Таким чином,

даний індекс дає загальне уявлення про продуктивність і не є надто важливим при порівнянні для різних задач, але дає уявлення про використання даних, і можливу оптимізацію розміру задач.

5. Висновок

В даній роботі був запропонований модифікований спосіб розподілених обчислень з використанням web браузерів, що дасть змогу, теоретично, отримати багатомільйонну групу клієнтів, кількість яких з кожним роком зростає. Це спричинене мінімальними затратами при підключенні користувача до системи. Необхідно лише відкрити web сторінку у своєму браузері. Даний спосіб дає змогу виконувати підзадачі в декілька потоках, завдяки технології WebWorkers, що дозволяє у фоновому режимі виконувати обчислення, не заважаючи користувачу продовжувати свою роботу в web браузері.

Для достовірності обрахунків необхідно ввести певний рейтинг оцінки користувачів. Підзадачі виконуються на стороні клієнта, і це дає змогу змінити результати обчислень, і надіслати недостовірну інформацію на сервер. Для цього необхідно вести рейтинг користувачів, перевіряючи їхні обчислення на інших клієнтах. Це збільшить час виконання головної задачі, за рахунок повторних однакових обчислень, але забезпечить достовірність результатів.

Інша проблема – це втрата результатів обчислень, у випадку коли клієнт отримав підзадачу, але результат не був надісланий. Необхідно виділити середній час, і у випадку, коли ми не отримаємо результат за цей час, вважати задачу не виконаною, і віддати цю підзадачу іншому клієнту. Даний підхід збільшить кількість повторного виконання, але результат буде більш достовірний.

Система розподілених обчислень, описана в даній статі може бути використана для благодійних обчислень, так як її використання для комерційних цілей не раціональна.

Список літератури

1. Boldrin F., Taddia C., Mazzini G. Distributed computing through web browser //Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th. – IEEE, 2007. – С. 2020-2024.
2. Заикин О. С., Семёнов А. А., Посыпкин М. А. Процедуры построения декомпозиционных множеств для распределенного решения SAT-задач в проекте добровольных вычислений SAT@ home //Управление большими системами: сборник трудов. – 2013. – №. 43.
3. Templin C., Templin J., Shearer A. Encryption system using web browsers and untrusted web servers : пат. 9537864 США. – 2017.
4. Pollard, J. Theorems on factorization and primality testing. Mathematical Proceedings of the Cambridge Philosophical Society, 76(3), 521-528. doi:10.1017/S0305004100049252.

УДК 004.75

ОВЧАРЕНКО П.О.,
ПОДРУБАЙЛО О.О.

АНАЛІЗ МЕТОДУ ГОРИЗОНТАЛЬНОГО МАСШТАБУВАННЯ СИСТЕМ ТА ЙОГО ОСНОВНІ ОСОБЛИВОСТІ

У даній статті розглянуто види масштабувань додатків, а також особливості побудови розподілених масштабованих додатків та основні проблеми, що виникають при їх розробці.

The subject of the article is an application scaling types and features of scalable distributed applications and basic problems that arise in their design.

1. Вступ

У час швидкого розвитку IT технологій, розростання додатків, та ресурсів для їх підтримки виникає проблема, коли один сервер вже не може впоратись з покладеною роботою. Основними

функціями серверів є відповідь на HTTP-запити, але, якщо даних для обробки або самих запитів забагато, один сервер може відмовити, зупинивши роботу всього додатку, для того, щоб уникнути таких ситуацій, розробники використовують масштабованість.

Масштабованість - це властивість обчислювальної системи, що забезпечує передбачуваний ріст системних характеристик, наприклад, числа підтримуваних користувачів, швидкості реакції, загальної продуктивності та іншого, при додаванні до неї обчислювальних ресурсів . Наприклад сервера СУБД можна розглядати два способи масштабування - вертикальний і горизонтальний

Вертикальне масштабування - це збільшення потужності окремого сервера СУБД і досягається заміною апаратного забезпечення (процесор, диски) на більш швидкодіючі або додаванням додаткових вузлів. Хорошим прикладом може служити збільшення числа процесорів в симетричних багатопроцесорних (SMP) платформах. При цьому програмне забезпечення сервера не повинно змінюватися

При горизонтальному масштабуванні системи з'єднуються через мережу або об'єднуються в кластер. Для міжз’єднань зазвичай використовуються стандартні мережеві технології, такі, як Fast Ethernet, Gigabit Ethernet (GBE) і Scalable Coherent Interconnect (SCI), що дають меншу пропускну здатність і більше запізнювання в порівнянні з вертикальними системами. Ресурси в цьому випадку розподіляються між вузлами, зазвичай містять від одного до чотирьох процесорів; кожен вузол має власний процесор і пам'ять і може мати власну підсистему введення-виведення або використовувати її спільно з іншими вузлами. На кожному вузлі працює окрема копія ОС. Ресурси розширюються за рахунок додавання вузлів, але не додавання ресурсів в вузол. Пам'ять в горизонтальних системах розподілена, тобто у кожного вузла є власна пам'ять, до якої безпосередньо звертаються його процесори і підсистема вводу-виводу. Доступ до цих ресурсів з іншого вузла відбувається набагато повільніше, ніж з вузла, де вони розташовані. Крім того, при горизонтальній архітектурі відсутній узгоджений доступ вузлів до пам'яті, а використовувані додатки споживають відносно небагато ресурсів, тому вони "поміщаються" на одному вузлі і їм не потрібен узгоджений доступ. Якщо ж з додатком буде потрібно кілька вузлів, то воно саме повинно забезпечити узгоджений доступ до пам'яті.[1]

Якщо горизонтальна система задовольняє вимогам додатків, то така архітектура краща, оскільки витрати на її придбання менше. Зазвичай вартість придбання в розрахунку на один процесор у горизонтальних систем нижче, ніж у вертикальних. Різниця в ціні пояснюється тим, що в вертикальних системах застосовуються більш потужні функції надійності, доступності та обслуговування - RAS (reliability, availability, serviceability), а також високопродуктивні міжз’єднання. Однак є ряд обмежень на застосування систем з горизонтальною архітектурою. Інший спосіб горизонтального масштабування - це великі обчислювальні системи з масовим паралелізмом (MPP), що складаються з безлічі встановлених в одній шафі невеликих процесорів, кожен з яких має власну копію ОС або копію мікроядра ОС. В даний час випускаються всього кілька систем MPP, які найчастіше представляють спеціалізовані рішення. Це, наприклад, системи Terradata виробництва компанії NCR, IBM RS / 6000SP (SP-2) і HP Tandem non-stop.[2]

Таблиця1. Типи додатків для вертикальної и горизонтальної архітектури

Вертикальні системи	Горизонтальні системи
•Великі бази даних •Бази даних транзакцій •Сховища даних •Поглиблене вилучення даних •Сервери додатків • Програми НРТС (не розбиваються на розділи)	• Web-сервери • Брандмауери • Проксі-сервери • Робота з потоковим мультимедіа • Каталоги • Обробка XML • Програми JSP • Шифрування SSL • Віртуальні приватні мережі (VPN) • Сервери додатків • Програми НРТС (розбиваються на розділи)

2. Особливості побудови масштабованого розподіленого додатку

При побудові розподіленого масштабованого додатку потрібно враховувати наступні особливості:

2.1 Сесії

Необхідно відстежувати навігацію користувачів по сайту. Для вирішення цього завдання зазвичай використовується механізм сесій, який полягає в привласненні кожному відвідувачу унікального ідентифікаційного номера, який йому передається для зберігання в cookies, або, в разі їх відсутності, для постійного "тягання" за собою через GET. Отримавши від користувача деякий ID разом з черговим HTTP-запитом сервер може подивитися в список вже виданих номерів і однозначно визначити хто його відправив. З кожним ID може асоціюватися якийсь набір даних, який веб-додаток може використовувати на свій розсуд, ці дані зазвичай за замовчуванням зберігаються в файлі в тимчасовій директорії на сервері.

Запити відвідувачів одного і того ж сайту можуть обробляти відразу кілька серверів, в такому випадку для визначення який саме сервер видав ID використовують наступні рішення:

- Централізоване зберігання сесій

Ідея проста: створюється для всіх серверів загальну "скарбничку", куди вони зможуть складати видані ними сесії і дізнаватися про сесії відвідувачів інших серверів. У ролі такої "скарбнички" теоретично може виступати і просто примонтована по мережі файлова система, але більш перспективним виглядає використання будь-якої СУБД, так як це позбавляє від маси проблем, пов'язаних зі зберіганням сесійних даних в файлах. Але в варіанті із загальною базою даних не варто забувати, що навантаження на нього буде неухильно зростати зі зростанням кількості відвідувачів, а також варто заздалегідь передбачити варіанти виходу з проблематичних ситуацій, пов'язаних з потенційними збоями в роботі сервера з цією СУБД.

- Децентралізоване зберігання сесій

Наочний приклад - зберігання сесій в memcached, спочатку розрахована на розподілене зберігання даних в оперативній пам'яті система дозволить отримувати всім серверам швидкий доступ до будь-яких сесійних даних, але при цьому (на відміну від попереднього способу) будь-який єдиний центр їх зберігання буде відсутній. Це дозволить уникнути вузьких місць з точок зору продуктивності і стабільності в періоди підвищених навантажень.

В якості альтернативи сесіям іноді використовують схожі за призначенням механізми, побудовані на cookies.

2.2 Статичний контент

Поки обсяги статичних даних невеликі - ніхто не заважає зберігати їх в локальній файловій системі і надавати доступ до них просто через окремий легкий веб-сервер, але рано чи пізно ліміт сервера по дисковому простору або файловій системі за кількістю файлів в одній директорії буде досягнутий, і потрібно перерозподілити

дані. Тимчасовим рішенням може стати розподіл даних по їх типу на різні сервера, або, можливо, використання ієрархічної структури каталогів.

Якщо статичний контент відіграє одну з основних ролей в роботі додатка, то варто задуматися про застосування розподіленої файлової системи для його зберігання. Це, мабуть, один з небагатьох способів горизонтально масштабувати обсяг дискового простору шляхом додавання додаткових серверів без будь-яких кардинальних змін в роботі самого додатка.

Альтернативою цьому підходу виступає використання так званих Content Delievery Network - зовнішніх сервісів, що забезпечують доступність Вашого контенту користувачам за певну матеріальну винагороду сервісу. Перевага очевидна - немає необхідності організовувати власну інфраструктуру для вирішення цього завдання, але зате з'являється інша додаткова стаття витрат.

2.3 Кешування

Кешування є сенс проводити на всіх етапах обробки даних, але в різних типах додатків найбільш ефективними є лише деякі методи кешування.

- СУБД

Практично всі сучасні СУБД забезпечують вбудовані механізми для кешування результатів запитів. Цей метод досить ефективний, якщо Ваша система регулярно робить одні й ті ж вибірки даних, але також має ряд недоліків, основними з яких є інвалідація кеша всієї таблиці при найменшій її зміні, а також локальне розташування кешу, що неефективно при наявності декількох серверів в системі зберігання даних.

- Додаток

На рівні додатків зазвичай проводиться кешування об'єктів будь-якої мови програмування. Цей метод дозволяє зовсім уникнути значної частини запитів до СУБД, сильно знижуючи навантаження на неї. Як і самі додатки такий кеш повинен бути незалежний від конкретного запиту і сервера, на якому він виконується, тобто бути доступним всім серверам додатків одночасно, а ще краще - бути розподіленим по декільком машинам для більш ефективної утилізації оперативної пам'яті. Лідером в цьому аспекті кешування по праву можна назвати Memcached — комп'ютерна програма, сервіс кешування даних в оперативній пам'яті на основі парадигми розподіленої хеш-таблиці.

- HTTP-сервер

Багато веб-серверів мають модулі для кешування як статичного контенту, так і результатів роботи скриптів. Якщо сторінка рідко оновлюється, то використання цього методу дозволяє без будь-яких видимих для користувача змін уникати генерації сторінки у відповідь на досить велику частину запитів.

- Reverse proxy

Поставивши між користувачем і веб-сервером прозорий проксі-сервер, можна видавати користувачу дані з кешу проксі (який може бути як в оперативній пам'яті, так і дисковим), не доводячи запити навіть до HTTP-серверів. У більшості випадків цей підхід актуальний тільки для статичного контенту, в основному різних форм медіа-даних: зображень, відео тощо. Це дозволяє веб-серверам зосередитися тільки на роботі з самими сторінками.

Кешування за своєю суттю практично не вимагає додаткових витрат на обладнання, особливо якщо уважно спостерігати за використанням оперативної пам'яті іншими компонентами сервера і утилізувати всі доступні "надлишки" під найбільш підходящі форми кеша.[4]

2.4 База даних

База даних є важливим компонентом веб застосування, яке складно масштабувати. Як вирішення цієї проблеми використовують заміну майстер сервера на master-slave з асинхронною реплікацією даних при такому підході всі операції запису виконуються лише на одному сервері (master), а інші сервера (slave) отримують дані безпосередньо від "майстра", обробляючи при цьому лише запити на читання даних. Як відомо, операції читання і запису будь-якого веб-проекту завжди ростуть пропорційно зростанню навантаження, при цьому зберігається майже фіксованим співвідношення між обома типами запитів: на кожен запит на оновлення даних зазвичай доводиться в середньому близько

десятка запитів на читання. Згодом навантаження зростає, а значить зростає і кількість операцій запису в одиницю часу. Рано чи пізно витрати операцій реплікації даних стануть настільки високі, що цей процес стане займати дуже велику частину процесорного часу кожного сервера, а кожен slave зможе обробляти лише порівняно невелика кількість операцій читання, і, як наслідок, кожен додатковий slave-сервер почне збільшувати сумарну продуктивність лише незначно, теж займаючись здебільшого лише підтриманням своїх даних відповідно до "майстра".

Тимчасовим вирішенням цієї проблеми може стати заміна master-сервера на більш продуктивний, але так чи інакше не вийде нескінченно відкладати перехід на наступний "рівень" розвитку системи зберігання даних: «sharding». Ідея полягає в тому, щоб розділити всі дані на частини по будь-якою ознакою і зберігати кожну частину на окремому сервері або кластері, таку частину даних в сукупності з системою зберігання даних, в якій вона знаходиться , і називають сегментом або shard'ом. Такий підхід дозволяє уникнути витрат, пов'язаних з реплікацією даних (або скоротити їх у багато разів), а значить і істотно збільшити загальну продуктивність системи зберігання даних. Але, на жаль, перехід до цієї схеми організації даних вимагає масу витрат іншого роду. Так як готового рішення для її реалізації не існує, доводиться модифікувати логіку додатку або додавати додатковий "прошарок" між додатком і СУБД, причому все це найчастіше реалізується силами розробників проекту. Готові продукти здатні лише полегшити їх роботу, надавши якийсь каркас для побудови основної архітектури системи зберігання даних і її взаємодії з іншими компонентами програми.

На такій стадії база даних може горизонтально масштабуватися для забезпечення вимог найважчих систем.

- Денормалізація

Запити, що комбінують дані з декількох таблиць, зазвичай при інших рівних вимагають більшого процесорного часу для виконання, ніж запит, який стосується лише однієї таблиці.

- Логічне розбиття даних

Якщо якась частина даних завжди використовується окремо від основної маси, то іноді має сенс виділити її в окрему незалежну систему зберігання даних.

- Низькорівнева оптимізація запитів

Ведучи і аналізуючи логи запитів, можна визначити найбільш повільні з них. Заміна знайдених запитів на більш ефективні з тією ж функціональністю може допомогти більш раціонально використовувати обчислювальні потужності.[4]

Варто згадати ще один, більш специфічний, тип інтернет-проектів. Такі проекти оперують даними, що не мають чітко формалізовану структуру, в таких ситуаціях використання реляційних СУБД в якості сховища даних, м'яко кажучи, недоцільно. У цих випадках зазвичай використовують бази даних з більш примітивною функціональністю в плані обробки даних, але зате вони здатні обробляти величезні обсяги інформації не чіпляючись до її якості та відповідності формату. В якості основи для такого сховища даних може служити кластерна файлова система, а для аналізу же даних в такому випадку використовується механізм під назвою MapReduce. Отже, на вході якісь довільні дані і не факт що в правильно дотриманому форматі. В результаті потрібно отримати якесь підсумкове значення або інформацію. Відповідно до даного механізму практично будь-який аналіз даних можна провести в наступні два етапи:

- Map

Основною метою даного етапу є уявлення довільних вхідних даних у вигляді проміжних пар ключ-значення, що мають певний сенс і формально оформлених. Результати піддаються сортуванню і гуртування по ключу, а після чого передаються на наступний етап.

- Reduce

Отримані після map значення використовуються для фінального обчислення необхідних підсумкових даних.

Кожен етап кожного конкретного обчислення реалізується у вигляді незалежного міні-додатки. Такий підхід

дозволяє практично необмежено розпаралелювати обчислення на величезній кількості машин, що дозволяє обробляти обсяги практично довільних даних. Для цього достатньо лише запустити ці програми на кожному доступному сервері одночасно, а потім зібрати воєдино всі результати. Прикладом готового каркаса для реалізації роботи з даними за таким принципом служить opensource проект Apache Foundation під назвою Hadoop.[5]

3. Висновок.

Використання горизонтальної масштабованості вимагає від додатку бути розподіленим, а також зробити реалізацію функціонала у відповідності вимог, які виникають при масштабуванні. В результаті отримана система являтиме собою додаток, що реагує на будь-яку кількість запитів, та об'єм оброблюваних даних і горизонтально масштабуватись у потрібний момент, що дозволить економити кошти власників додатку.

Перелік посилань

1. Кузнецов С. Д., Посконин А. В. Распределенные горизонтально масштабируемые решения для управления данными // Труды ИСП РАН. 2013.[Інтернет ресурс]. URL: http://cyberleninka.ru/article/n/raspredelennye-gorizontalno-masshtabiruemye-resheniya-dlya-upravleniya-dannymi .
2. C. Strozzi, «NoSQL: A Relational Database Management System,» [Інтернет ресурс]. URL: http://www.strozzi.it/cgibin/CSA/tw7/I/en_US/nosql/Home%20Page.
3. D. Karger, A. Sherman, A. Berkheimer, B. Bogstad, R. Dhanidina, K. Iwamoto, B. Kim, L. Matkins и Y. Yerushalmi, «Web Caching with Consistent Hashing,» MIT Laboratory for Computer Science, 1999. [Інтернет ресурс]. URL: http://www8.org/w8-papers/2awebserver/caching/paper2.html.
4. D. Obasanjo, «Building scalable databases: Denormalization, the NoSQL movement and Digg,» 2009. [Інтернет ресурс]. URL: http://www.25hoursaday.com/weblog/2009/09/10/BuildingScalableDatabases DenormalizationTheNoSQLMovementAndDigg.aspx.
5. "MapReduce Tutorial" [Інтернет ресурс]. URL: https://hadoop.apache.org/docs/r1.0.4/mapred_tutorial.html.

УДК 621.372.542

ОВЧАРЕНКО П.О.
САВЕРЧЕНКО В.Г.

ЦИФРОВА ФІЛЬТРАЦІЯ В СИСТЕМАХ ОБРОБКИ ЗОБРАЖЕНЬ

Досліджуються низкочастотна просторова та медіанна фільтрації зображень, спотворених дією імпульсних завад. Особлива увага приділяється модифікованому методу медіанної фільтрації для пошкодженого зображення, який використовує подвійну обробку апертури фільтру. Такий модифікований медіанний фільтр дозволяє суттєво поліпшити якість зображення.

The methods of low-frequency spatial and median filtering of images distorted by the action of impulse noise. Particular attention is paid to the modified median filtering method for the distorted image, which uses a double processing of the filter aperture. Such a modified median filter allows to significantly improve image quality.

1. Вступ

Актуальність вирішення задач обробки зображень загальновідома. Серед них одна із найважливіших є корекція зображень. Корекція зображень – це приведення до необхідних значень характеристик зображень. Одним із прикладів корекції може бути усунення завад [1].

Багато задач вилучення інформації з зображень можуть і повинні вирішуватися автоматичними пристроями. Спотворенні завадами зображення суттєво ускладнюють їх автоматичну обробку. Підвищення якості зображень досягається фільтрацією, що послаблює дію завад. Цифрове зображення уявляє собою двовимірну функцію. Найчастіше корисний сигнал змінюється повільніше, ніж завада. Тому підвищення якості зображень на основі цифрової фільтрації передбачає зміну атрибутів поточних пікселів в залежності від атрибутів пікселів, що їх оточують.

2. Постановка задачі

Найбільше застосування одержали фільтри “ковзного вікна”, що використовують атрибути оточуючих пікселів. На зображенні виділяють вікно розміром $N \times M$ пікселів, де значення N та M непарні числа. Тоді поточний центральний піксель вікна є деякою функцією лише елементів вікна.

Для обробки зображень з метою підвищення їх якості використовують різні процедури фільтрації, що дозволяють вилучати імпульсні завади. Найпростішим прикладом фільтра ковзного вікна є низькочастотний фільтр, коли значення центрального пікселя замінюється в залежності від маски фільтру, обчисленого по всіх пікселях вікна.

При медіанної фільтрації центральний піксель вікна замінюється медіаною всіх пікселів зображення у вікні, тобто пікселем для якого існує половина елементів вікна, менших або рівних йому за величиною, і половина елементів, більших або рівних йому за величиною.

3. Фільтрація зображень

Оскільки шум просторово декорельований, в його спектрі, як правило, містяться вищі просторові частоти, ніж в спектрі звичайного зображення. Отже, проста низькочастотна фільтрація може служити ефективним засобом згладжування шумів. Низькочастотна фільтрація зображення передбачає сканування спотвореного зображення двомірним вікном певного розміру.

Масив відфільтрованого зображення C формується за допомогою дискретної згортки масиву початкового зображення A з масивом фильтру F, який часто називається шумопослаблюючею маскою.

Низькочастотна фільтрація – це метод лінійної обробки зображень Для такої фільтрації при скануванні апертури зліва направо і зверху вниз обчислення виконуються за формулою:

$$C_{r,s} = \frac{1}{k} \cdot \sum_{i=-1}^{1} \sum_{j=-1}^{1} A_{r+i,s+i} \cdot F_{i+1,j+1}$$

де $C_{r,s}$ – матриця пікселів зображення після фільтрації; $r = \overline{1, n-2}$; $s = \overline{1, m-2}$; n та m – кількість строк та рядків відповідно; $A_{r+i,s+i}$ – значення пікселів матриці початкового зображення з завадами; $k = \sum_{i=0}^{2} \sum_{j=0}^{2} F_{i,j}$ – коефіцієнт нормалізації; F – матриця коефіцієнтів фільтру. Для низькочастотних фільтрів можна навести такі приклади масок:

$$F1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix};$$

$$F2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix};$$

$$F3 = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

Маска $F1$ являє собою усереднену фільтрацію, для якої коефіцієнт нормалізації $k = 9$.

Використання коефіцієнта нормалізації дозволяє отримати одиничний коефіцієнт

передачі, щоб процедура фільтрації шуму не викликала зміщення середньої яскравості обробленого зображення.

На основі використання таких фільтрів можуть бути розроблені інші процедури фільтрації. Наприклад, процедура порогового фільтру, обчислює пікселі наступним чином. Якщо яскравість пікселя x перевищує середню яскравість групи q_i найближчих пікселів на деяку порогову величину, то яскравість пікселя змінюється на середню яскравість. Наприклад, для вікна 3×3 пікселя $\begin{pmatrix} q_1 & q_2 & q_3 \\ q_4 & x & q_5 \\ q_6 & q_7 & q_8 \end{pmatrix}$, якщо $\left[x - \frac{1}{8}\cdot\sum_{i=1}^{8} q_i \right] \geq \varepsilon$, то $x = \frac{1}{8}\cdot\sum_{i=1}^{8} q_i$.

Метод підвищення контрасту з використанням оператора Лапласа може використовувати такі маски:

$$L1 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix};$$

$$L2 = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix};$$

$$L3 = \begin{pmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{pmatrix}.$$

Медіанна фільтрація – це метод нелінійної обробки сигналів. Одновимірний медіанний фільтр являє собою ковзне вікно, яке охоплює непарне число пікселів зображення. Центральний елемент замінюється медіаною всіх елементів зображення у вікні. Медіаною дискретної послідовності $a_1, a_2, \ldots, a_n$ для непарного n є той її елемент, для якого існує $\frac{n-1}{2}$ елементів, менших або рівних йому за величиною, і $\frac{n-1}{2}$ елементів, більших або рівних йому за величиною.

Концепцію медіанного фільтру легко узагальнити на два виміри, застосовуючи двовимірне вікно бажаної форми, наприклад, прямокутне або близьке до кругового. Очевидно, що двовимірний медіанний фільтр з вікном розміру $n\times n$ забезпечує більш ефективне усунення завад, ніж послідовно застосовані горизонтальний і вертикальний одномірні медіанний фільтри вікном розміру $n\times1$.

Медіанний фільтр більш ефективно послаблює імпульсні завади, ніж гладкі шуми.

Процедура медіанної фільтрації на мові MathCad має вигляд:

```
A := READBMP("image.bmp")
i := 0..2          j := 0..2
n := rows(A)       m := cols(A)
r := 0..n - 3      s := 0..m - 3
F_{i,j} := A_{r+i,s+j}
Fun(A,r,s) := | for i ∈ 0..2
              |   for j ∈ 0..2
              |     F_{i,j} ← A_{r+i,s+j}
              | Q ← sort(stack(F^⟨0⟩, F^⟨1⟩, F^⟨2⟩))
              | Q_4
C_{r,s} := Fun(A,r,s)
WRITEBMP("image2.bmp") := C
```

Розглянемо використання вікна розміром 3×3 на прикладі використання усередненої і звичайної медіанної фільтрації. Рис. 1 показує фрагмент початкового зображення з завадами (три середніх пікселя, якого зі значенням 4, являють собою приклад імпульсної завади). На рис. 2 та рис. 3 наведено результат застосування процедури усередненої і звичайної медіанної фільтрацій відповідно.

8	7	6	5	5	5	1
8	7	5	5	1	0	0
7	5	5	0	0	0	0
7	5	0	**4**	**4**	0	0

Рис.1. Зображення

8	7	6	5	5	5	1
8	6	5	4	2	1	0
7	5	4	3	2	1	0
7	5	3	**2**	**1**	1	0

Рис.2. Усереднена фільтрація

8	7	6	5	5	5	1
8	7	5	5	1	0	0
7	5	5	4	0	0	0
7	5	4	**0**	**0**	0	0

Рис.3. Звичайна медіанна фільтрація

Як видно з рис.2 та рис.3 результати фільтрації суттєво відрізняються. Звичайна медіанна фільтрації дає суттєво кращі результати ніж усереднена фільтрація. Медіанна фільтрація дозволяє зменшити кількість завад та більшою мірою зберегти палітру зображення . Але процедура звичайної медіанної фільтрації не забезпечують потрібної якості обробки зображень.

Використання модифікованої медіанної фільтрації зображень, яка передбачає двоетапну обробку кожного пікселя, підвищує якість зображень [3].

Сутність модифікованого способу медіанної фільтрації апертурою $k \times k$ пікселів зображення з координатами x, y полягає в тому, що попередньо виконується фільтрація всіх пікселів, що входять до апертури, центром якої вказано піксель з фіксацією результатів в окремому допоміжному масиві, після чого виконується фільтрація пікселя $a_{x,y}$ зображення з координатами x, y з урахуванням попередньої фільтрації оточуючих пікселів, що входять в апертуру.

Процедура такої фільтрації пікселя зображення з координатами x, y може бути описана наступним чином:

1. Для всіх пікселів зображення з координатами x+i та y+j (i,j∈{-k,k}, i,j≠0), що входять до поточної апертури, виконується формування допоміжного масиву:

$$b_{i,j} = \frac{a_{x+l,y+q}}{\sum\limits_{\substack{l=-k,..,k,\\ q=-k,..,k}} N(a_{x+i,y+j}, a_{x+l,y+q})} = \frac{k^2-1}{2},$$

де функція N(i,j,c) визначається наступним чином: N(d,c)=1, якщо c > d і N(d,c)=0, якщо c ≤ d.

2. Обчислюється відфільтроване значення пікселя $a^1_{x,y}$ за рахунок заміни попереднього його значення на медіану апертури з допоміжного масиву b:

$$a^1_{x,y} = \frac{b_{l,q}}{\sum\limits_{\substack{l=-k,..,k,\\ q=-k,..,k}} N(a_{x,y}, b_{l,q})} = \frac{k^2-1}{2}.$$

Результат модифікованої медіанної фільтрації має вигляд (рис. 4).

8	7	6	5	5	5	0
8	7	5	5	5	0	0
7	5	5	0	0	0	0
7	5	0	**0**	**0**	0	0

Рис.4. Модифікована медіанна фільтрація

Таким чином, модифікована медіанна фільтрація повністю усунула завади.

4. Висновки

Звичайний медіанний фільтр практично повністю усунув завади, в той час як усереднений фільтр розмазав завади по зображенню. Звичайний медіанний фільтр пригнічує імпульсні завади, тривалість яких становить менше половини ширини вікна.

Модифікована медіанна фільтрація додаткового підвищує якість фільтрації за рахунок двоетапної обробки кожного пікселя.

Перелік посилань

1. Бузовский О.В., Болдак А.А., Мохаммед Руми М.Х. Компьютерная обработка изображений. – К.: “Корнійчук”, 2001. – 180 с.

2. Гонсалес Р. Цифровая обработка изображений / Р. Гонсалес, Р. Вудс; пер. с англ. В.В. Харитонова.- М.:Техносфера, 2005. – 1072 с.

3. Марковський О.П., Невдащенко М.В., Білашевська А.М. Захищена реалізація фільтрації зображень в grid-системах. // Вісник НТУУ “КПІ”. Інформатика, управління та обчислювальна техніка: збірник наукових праць – К.: “Век+”, 2014. – № 61. – С. 103–109.

УДК 004.75

РОТЕНБЕРГ О.В.
ЛУЦЬКИЙ Г.М.
ВОЛОКИТА А.М.
МАРТИНЮК Р.О.

СПОСІБ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ МОНІТОРИНГУ ОБЧИСЛЕНЬ В РОЗПОДІЛЕНИХ СИСТЕМАХ

У даній роботі розглянуто централізовану модель розподілених обчислень. У системі є централізований сервер (так званий менеджер), що розподіляє задачі між вузлами (агентами) для виконання обчислень. В даній роботі розглянуто способи та підходи для мінімізації обчислень на стороні менеджера та підвищення ефективності перевірки результатів обчислень.

In this work was considered centralized model for distributed computing. System has a centralized server (manager), which distributes tasks between nodes (agents) to perform computing. In this work was considered methods and approaches to minimize calculations and to improve efficiency validation of computing.

Ключові слова : розподілені обчислення, централізований сервер, менеджер, агент.

1. Вступ

Сучасні задачі потребують великі обчислювальні потужності. Системи розподілених обчислень, які координують задачі між тисячами чи навіть мільйонами добровільних агентів стають дедалі популярнішими. Прикладами таких систем є SETI@home та Rosetta@home, мета яких проаналізувати величезні обсяги даних в пошуках позаземного життя чи кращого розуміння процесу згортання білка, відповідно. В цих системах кожне додаткове обчислення приносить більше користі.

У даній роботі розглядається впровадження модифікованої кредитної системи, яка нагороджує чи штрафує агента в залежності від результатів та часу обчислень.

2. Модель

Центральний сервер (менеджер), нагороджує агентів за виконану роботу. Основна ціль - зменшити кількість обчислень на стороні менеджера. В системі агенти запитують нові задачі у менеджера, але в певний момент агент може перестати слати запити на нову роботу.

Менеджер нагороджує агента за коректно виконану роботу нагородою *r*. Якщо менеджер

з’ясовує, що агент повернув некоректний результат, менеджер накладає на нього штраф *f,* який вираховується з загального рейтингу агента. Як результат, ми бачимо залежність *f/r* - чим вище співвідношення, тим менша ймовірність, що задача буде видана цьому агенту [1].

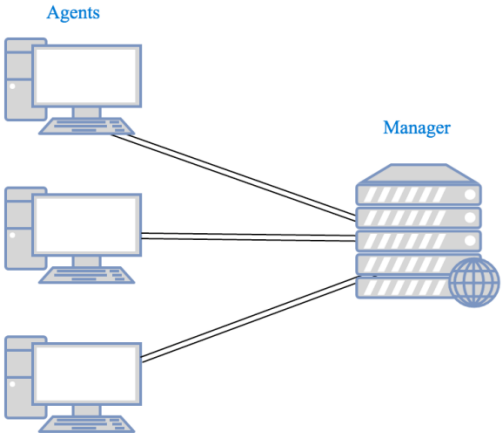

Рис. 1. Централізована модель розподілених обчислень.

Розглянемо випадок, коли менеджер видав задачу агенту. У агента є 2 варіанти. Перший - агент чесно виконає роботу і отримає винагороду r. Якщо ми визначимо вартість обчислень за допомогою алгоритму, що надав менеджер як *cost(i)*, то рентабельність обчислень *u(i)* становитиме *u(i) = r - cost(i),* де *i* – це і-та задача. В цьому випадку, ми передбачаємо, що r більша від *cost(i)*, інакше агент відмовиться виконувати роботу[1].

У другому випадку агент повертає результат, використовуючи інший алгоритм. Наприклад, агент має уявлення про вигляд відповіді, і намагається її вгадати, чи він має доступ до альтернативного алгоритму, який надає правильну відповідь з ймовірністю *q* (наприклад оптимізований клієнт SETI@home). У цьому випадку агент все ще може отримати *r*, але великий ризик отримати штраф *f*, якщо менеджер виявить, що результат некоректний.

Позначимо ймовірність того, що менеджер виявив невірну відповідь як *p.* Хоча ми не припускаємо, що менеджер кожен раз в змозі виявити некоректну відповідь та оштрафувати винного агента, оскільки перевірка на коректність може значно збільшити необхідність використання обчислювальних ресурсів менеджера[2]. Якщо менеджер встановлює співвідношення штрафу до инагороди $f/r \geq (1-p)/p$, де $p = c(1-e)$, то раціональний агент поверне коректний результат щонайменше *e* разів, де *c* - ймовірність того, що результат агента буде перевірено [1].

Такий чином, будь який раціональний агент буде використовувати найменш дорогий алгоритм, який забезпечує коректні відповіді щонайменше з ймовірністю *e.*

3. Перевірка результату

Подвійна перевірка

Найбільш простою стратегією для менеджера є випадкова подвійна перевірка отриманої відповіді з ймовірністю *t.* Але менеджер не може знати, чи правильна відповідь, поки її не перевірить, тому *c = t.* Встановлення низького значення *t* надасть можливість менеджеру зменшити кількість роботи пов’язаної з подвійною перевіркою - але так як *c* обернено пропорційне до *f/r*, то високий *f/r* може стати нездоланним бар’єром, для агентів, які шукають роботу[3].

Залучення декількох агентів

Менеджер може мінімізувати кількість перевірок надсилаючи однакову роботу декільком агентам. Менеджер виконує подвійну перевірку лише у випадку, коли у агентів різні результати. Проблема в тому, що якщо всі агенти відправлять одну й ту саму невірну відповідь, то менеджер ніяк не зможе це виявити.

Припустимо, що частка агентів *h* завжди виконують обчислення вірно – автори називають таких агентів «старанними» або «раціональними» [1]. Менеджер обирає *m* агентів випадковим чином і назначає їм одну й ту саму задачу. Опишемо *c* як ймовірність, що агент буде спійманий іншим агентом, якщо він надасть невірну відповідь.

Припустимо, що менеджер надає задачу *m* агентам, кожен з яких виконує задачу чесно з ймовірністю *h,* тоді ймовірність того, що шахрая впіймають, становить $c = 1 - (1 - h)^{(m-1)}$ [1].

Ця стратегія все ще потребує від менеджера виконання роботи, коли агенти надсилають різні результати. В системі, де всі агенти раціональні, таких ситуацій взагалі не повинно виникати. Але, якщо існують агенти в змові, чи злодії, вони можуть змусити менеджера виконувати повторну перевірку.

Гібридна стратегія

Менеджер також може застосувати гібридну стратегію : він може надати одну й ту саму задачу декільком агентам та випадковим чином повторно перевірити деякі відповіді. Так, навіть якщо всі агенти в змові нададуть хибну відповідь, менеджер зможе викрити їх.

Залучення 2 раціональних агентів

Менеджер випадковим чином обирає нову групу агентів для кожної задачі. Зробимо припущення, що дані агенти є "сумлінними", і якщо ми коректно встановимо відношення f/r, то основною стратегією для агента буде чесне обчислення.

Тут буде рівність - якщо всі інші агенти обманюють, то і раціональний агент буде також обманювати, і навпаки, якщо хоча б один агент чесний, то і раціональний агент повинен бути чесним.

Ми можемо порушити цю рівність для нечесних агентів, встановивши для них заохочення. Якщо надані різні результати, менеджер перевірить обчислення і нагородить всіх агентів, які надали правильний результат. Тепер очікувана рентабельність за сумлінне виконання задач, коли всі інші вибирають бути лінивими агентами є $u(1) = r - cost(1) + b(1 - q)$[1].

Якщо менеджер наймає двох агентів для виконання задачі, тоді він повинен встановити $f/r > 0$ і дати чесним агентам $b \geq r/(1 - q)$, де b – винагорода, коли вони зловили шахрая [1].

Шкідливі агенти

Шкідливі агенти атакують систему - їх мета знизити точність результатів та збільшити кількість повторних перевірок, які повинен зробити менеджер. Вони не раціональні, але для того, щоб утримуватися в системі, вони повинні тримати хоча б нульовий баланс рентабельності (якщо вони не можуть заплатити штраф, то вони не зможуть бути найняті менеджером) [4]. Шкідливі агенти можуть домовлятися шляхом централізованого контролю, через зовнішні комунікації та навіть через обмін ресурсами (винагорода *r*).

Навіть шкідливий агент повинен підтримувати певний позитивний баланс на своєму рахунку, так як в протилежному випадку менеджер може не надати йому задач для обчислення. Тому, шкідливі агенти, які мають намір відправляти якнайбільше некоректних результатів, повинні також певну частку часу виконувати обчислення коректно.

Модель кредитного скорингу

Якщо рейтинг агента є від'ємним, то для прийняття рішення використовується модель кредитного скорингу. Проблему кредитного скорингу можна розглядати як завдання класифікації: знаючи відповіді на питання анкети x ∈ A, визначити, до якої групи належить позичальник: x ∈ Ag для «хороших клієнтів», і x ∈ Ab для «поганих». При цьому необхідно розуміти, що абсолютно точна класифікація принципово неможлива[6].

Згідно з проведеними дослідженнями [7], до теперішнього часу не отримано відомостей про значні переваги якого-небудь з традиційних методів скорингу в точності одержуваних результатів, тобто рівні похибок при використанні цих методів є порівнянними. Так для одних методів (лінійна регресія) бажано використання рівних часток «поганих» і «хороших», тоді як інші методи (дерева класифікації, байєсовські мережі) вимагають, щоб вибірка відображала реальне співвідношення «поганих» і «хороших» клієнтів. Більшість статистичних методів призводять до побудови правила класифікації, заснованого на лінійній скоринговій функції.

При реалізації будемо розглядати 2 методи: регресійний аналіз як один з найпоширеніших методів та лінійне програмування, що дає можливість включити в програму додаткові обмеження. [8,9]

Рівняння множинної регресії (з багатьма змінними) у вигляді:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_m X_m + \varepsilon$$

де $X = (X_1, X_2, \dots, X_m)$ - вектор незалежних (пояснюючих) змінних; $\beta = (\beta_1, \beta_2, \dots, \beta_n)$ - вектор параметрів (що визначаються); ε – випадкова помилка (похибка); Y – залежна (пояснювальна) змінна. Для оцінки параметрів β використовується метод найменших квадратів.

В методі лінійного програмування є набір даних, що включає в себе відповіді на питання x_i для кожного з N клієнтів і індикатори Y_i. Лінійна функцію *s(x)*, яка розділяє «хороших» і «поганих» клієнтів, шукається з міркувань мінімізації помилки a_i.

$$\begin{cases} a_1 + \cdots + a_N \to min \\ w_1 x_{i1} + \cdots + w_m x_{im} \geq c - a_i, \;\; Y_i = 0, \\ w_1 x_{i1} + \cdots + w_m x_{im} \leq c + a_i, \;\; Y_i = 1, \\ a_i \geq 0 \end{cases}$$

Мінімізація проводиться за змінним $(w_1, \dots, w_n, c, \; a_1, \dots, a_N)$. Додаткові обмеження

можна накласти через обмеження відповідних ваг $w_i > w_i^0$.

Модифікований спосіб моніторингу обчислень агентом з часовими обмеженнями

Оптимальний час виконання задачі – це час t_{opt}, за який середньостатистичний агент виконує задачу. Час виконання задачі, це відлік часу від моменту отримання задачі агентом до моменту відправки результату менеджеру. Так як, агенти мають різні обчислювальні потужності, необхідно ввести похибку *del*, яка коригуватиме оптимальний час в залежності від кількості отриманих менеджером результатів конкретної задачі (чим більше відповідей отримано, тим менша *del* відповідно).

При отриманні правильної відповіді, виконаної за оптимальний час t, агент отримує винагороду *rk*, де k – коефіцієнт, значення якого за замовчуванням 1, і при кожній наступній правильній відповіді $k_{new} = k_{old}*(1+del)$, але поки $\frac{f}{rk} > 1$

При отримані правильної відповіді, але за час, що не є оптимальним, агент отримує винагороду *rk*, де *k* – коефіцієнт, значення якого за замовчуванням 1, і при кожній наступній неправильній відповіді $k_{new} = k_{old}*(1-del)$, але необхідно, щоб cost(rk) була меншою, ніж *rk,* так як при від'ємній рентабельності агенти не матимуть сенсу виконувати задачі.

Стратегія реального часу є додатковим стимулом для чесних агентів. Також, дана стратегія дозволяє менеджеру швидше отримувати результат, та в результаті покращити загальний час обчислення всього обсягу задач [5].

4. Висновки

В даній статті було представлено різні техніки, що можуть бути застосовані для підвищення ефективності розподілених обчислень, перешкоджаючи зайвим обчисленням менеджером або іншими агентами.

В роботі запропоновано модифікований спосіб підвищення ефективності моніторингу обчислень в розподілених системах, який за рахунок використання алгоритмів кредитного скорингу та введення додаткових часових обмежень на виконання обчилювальних задач, дозволяють підвищити рівень коректності обчислень та задавати додаткові характеристики часу виконання задач.

В статті показано використання технік, для визначення розумного *f/r,* при якому всі раціональні агенти поводитимуться чесно, та зменшаться наслідки від зловмисних агентів.

Всі ці техніки необхідні для того, щоб зменшити кількість обчислень, які необхідно виконати менеджеру. Було розглянуто ситуацію, в якій менеджер в змозі платити винагороду і штрафувати агентів.

В подальшій роботі можливі такі модифікації - до моменту, коли виконання роботи розпочалося, агент може надати заставу у вигляді штрафу, який буде стягнуто, у випадку, якщо менеджер розкриє шахрайство. Також, менеджер може надавати різні f/r для різних агентів, пропонуючи більшу винагороду, для тих, хто прийме і умови з більшими штрафами.

Список літератури

1. Belenkiy M. et al. Incentivizing outsourced computation //Proceedings of the 3rd international workshop on Economics of networked systems. – ACM, 2008. – С. 85-90.

2. K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B.A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. Multiagent and Grid Systems, 1(3):169–182, 2005.

3. F. Monrose, P. Wyckoff, and A. Rubin. Distributed execution with remote audit. ISOC NDSS, 1999.

4. P. Golle, and I. Mironov. Uncheatable distributed computations. CT-RSA, 2001.

5. Anderson D. P., Fedak G. The computational and storage potential of volunteer computing //Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on. – IEEE, 2006. – Т. 1. – С. 73-80.

6. ВОЛОКИТА, А. М., et al. Дослідження ефективності приватної хмарної системи для обчислень кредитного скорингу. *Вісник Чернігівського державного технологічного університету. Серія: Технічні науки*, 2013, 4: 115-121.

7. Гараган Сергей Александрович. Метод эмпирической скоринговой функции и его использование в кредитном процессе. // [Електронний ресурс]. Режим доступу: http://crosys.org/empirical_scoring_function.html (дата звернення 15.04.2017) - Назва з екрану.
8. Бородич С.А. Эконометрика. Учебное пособие // Минск: Новое знание, 2001. – 408 с. [Електронний ресурс]. Режим доступу: http://www.economy.bsu.by/library/Бородич_Эконометрика/Бородич_Эконометрика.pdf (дата звернення 10.04.2017) – Назва з екрана.
9. Логистическая регрессия и ROC-анализ - математический аппарат. [Електронний ресурс]. Режим доступу: http://www.basegroup.ru/library/analysis/regression/logistic/
(дата звернення 15.04.2017) – Назва з екрана.

УДК 004.2

СКОРІЧЕНКО О.В.
ЖАБІН В.І.,

СКОРОЧЕННЯ НЕОБХІДНОГО РЕСУРСУ ПЛІС ДЛЯ РЕАЛІЗАЦІЇ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ З БЕЗПОСЕРЕДНІМИ ЗВ'ЯЗКАМИ МІЖ МОДУЛЯМИ

Запропоновано метод обчислення поліномів з порозрядною обробкою операндів у надлишковому поданні в системах з безпосередніми зв'язками між обчислювальними модулями. Використання зазначеного методу дозволяє скоротити необхідний ресурс ПЛІС завдяки малому числу зв'язків між компонентами системи.

A method is proposed for computing polynomials with bitwise processing of operands in redundant representation in systems with direct connections between computational modules. The use of this method allows to reduce the required resource of the FPGA due to the small number of connections between system components.

1. Вступ

Тривалість обробки інформації у паралельних системах залежить не тільки від часу виконання операцій, але і від затрат часу на обмін інформацією між гілками алгоритмів, тобто між обчислювальними модулями (ОМ) паралельної системи. Зменшити затрати часу на обмін даними дозволяє використання потокових систем з безпосередніми зв'язками (ПСБЗ) між ОМ [1, 2].

У ПСБЗ виходи одних ОМ підключаються до входів інших ОМ відповідно до графа потоку даних (ГПД). ОМ працюють у неавтономному режимі. У процесі обчислень дані пересилаються безпосередньо від одних ОМ до інших, перетворюючись на кожному кроці відповідно до операцій, що задані вершинами ГПД. У такому випадку відсутні складні процедури пересилання даних між ОМ через загальну пам'ять, тобто зменшуються затрати часу на обмін даними між ними. Формалізовані методики переходу від ГПД до структури ПСБЗ відомі [3-5].

Використання сучасної технології проектування RSoC (Reconfigurable system on a chip – реконфігурована система на кристалі) дозволяє створювати складні системи на основі ПЛІС. Однак, з боку ресурсів мікросхем накладається ряд обмежень. При паралельній передачі інформації між мікросхемами виникають проблеми, пов'язані з можливою нестачею виводів, а в середині мікросхеми суттєво витрачається внутрішній ресурс функціональних елементів та елементів зв'язку. Зростає енергоспоживання і збільшуються габаритні розміри системи. Тому більш ефективним є порозрядний обмін даними.

Використання декількох мікросхем створює додаткові проблеми. Недоліком такої реалізації є те, що мікросхеми можуть не мати необхідної кількості виводів для забезпечення зв'язків між частинами системи. При цьому частина ресурсів мікросхем може залишатися незадіяною.

З огляду на важливість проблеми нестачі виводів мікросхем, компанією Virtual Machine Works запропонована технологія VirtualWire (віртуальні з'єднання) для побудови систем на декількох мікросхемах [6]. Ідея, закладена в основі технології, полягає у використанні незадіяного обладнання для реалізації спеціальних ланцюгів, що забезпечують почергове підключення до виводів мікросхеми інформації з різних джерел всередині мікросхеми. До недоліків даної технології слід віднести великі часові затримки просування потоків даних, що суперечить самій ідеї потокової моделі обчислень.

2. Постановка задачі

Одним з підходів до вирішення проблеми зменшення кількості зв'язків між ОМ є використання квазіпаралельних обчислювальних пристроїв, що дозволяють суміщати процеси порозрядного введення операндів і порозрядного формування результатів [5, 7]. Подання чисел в надлишкових системах числення неоднозначне, тобто одне і те ж число може бути записане різними послідовностями цифр. Наприклад, десяткове число 5/16 в двійковій системі з цифрами {-1,0,1} можна подати у вигляді: $0{,}0101$; $0{,}1\bar{1}01$; $0{,}01\bar{1}\bar{1}$ і т.д. (тут «$\bar{1}$» – інший запис цифри «-1»). Це дозволяє виконувати операції порозрядно без переносів у старші розряди. Режим роботи таких ОМ називають неавтономним, бо для виконання послідовності операцій необхідно кілька ОМ, які обмінюються інформацією у процесі роботи.

Хоча на входах і виходах ОМ числа представлені послідовним кодом, такі пристрої по внутрішній організації ближчі до паралельних пристроїв. У зв'язку з цим вони отримали назву квазіпаралельні [2, 7]. З використанням квазіпаралельних ОМ при виконанні послідовності залежних за даними операцій реалізується паралелізм на рівні обробки розрядів операндів.

У даній роботі на прикладі ПСБЗ для обчислення поліномів досліджується можливість скорочення апаратних ресурсів ПЛІС за рахунок використання квазіпаралельних ОМ відносно використання ОМ з паралельним обміном даних.

3. Організація обчислення поліномів

Процес виконання в часі послідовності операцій у неавтономному режимі ілюструється діаграмою на рис. 1.

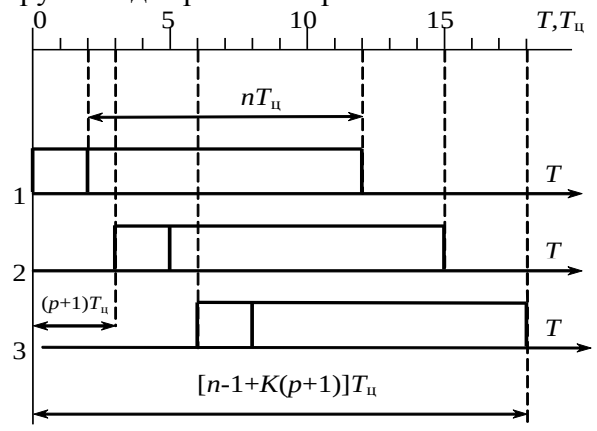

Рис. 1. Часова діаграма виконання залежних операцій в 3-х ОМ

Операції виконуються зі старших розрядів за допомогою ланцюга ОМ, що формують цифри проміжних результатів із затримкою на p циклів кожний. Наступний у ланцюгу ОМ починає свою операцію через ($p+1$) циклів відносно попереднього ОМ. Це означає, що час обчислень, необхідний для отримання остаточного результату, складає

$$T = \left[n - 1 + \sum_{j=1}^{K} \left(p_j + 1 \right) \right] t_{\text{Ц}}, \qquad (1)$$

де n – розрядність операндів і результату; k–число операцій у ланцюгу; p_j– затримка формування розрядів результату j-ї операції; $t_Ц$ – тривалість циклу формування одного розряду результату в ОМ.

Для обчислення поліномів скористаємося методом Горнера першого порядку. Наприклад, поліном 5-го порядку можна записати у вигляді

$$P(x,a_i)=((((a_5x+a_4)x+a_3)x+a_2)x+a_1)x+a_0$$

Перший підхід. Поліном можна обчислити методом певної суперпозиції операцій множення і додавання. В двійковій системі числення з цифрами {-1,0,1} алгоритм виконання додавання та множення для правильних дробових чисел в i-му циклі може бути зведений до наступних дій [7]:

$$H_i = 2R_{i-1} + F_i, \qquad (2)$$

$$z_i = \begin{cases} -1, \text{ якщо } H_i < -2^{-1}; \\ 0, \text{ якщо } -2^{-1} \le H_i \le 2^{-1}; \\ 1, \text{ якщо } 2^{-1} < H_i, \end{cases} \qquad (3)$$

$$R_i = H_i - z_i, \qquad (4)$$

де H_i, R_i – проміжні змінні; F_i – приріст функції, що залежить від i-х цифр операндів; z_i – цифра результату. Початковим є значення $R_0=0$.

Приріст F_i для операції додавання та множення обчислюється за формулами:
$F_i=2^{-p}(x_i + y_i)$,

$$F_i=2^{-p}(x_iY_{i-1} + y_iX_{i-1}), \qquad (5)$$

де p – затримка початку формування розрядів результату в циклах; x_i і y_i – цифри операндів з вагою 2^{-i}; Y_i і X_i. – операнди, що подані i старшими розрядами. В [7] показано, що для даних операцій маємо $p \ge 2$.

Якщо множення і додавання виконувати в окремих ОМ, зв'язаних послідовно (всього в ланцюжку десять ОМ), то час обчислювання в синхронному режимі згідно з (1) буде складати

$$T = (n + 29)\,t_{\text{М}}, \qquad (6)$$

де $t_{\text{М}}$ – тривалість циклу множення (більшого за цикл додавання).

Другий підхід. Кількість циклів обчислення поліномів можна зменшити, якщо використовувати ОМ, що виконують більш складну операцію.

Нехай система являє собою ланцюжок з 5-ти ОМ, кожен з яких виконує проміжну операцію $Z = XY + A$ (рис. 2).

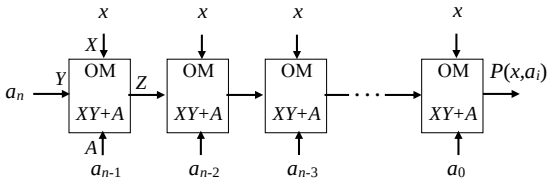

Рис. 2. Система для обчислення поліномів $P(x,a_i)$ за схемою Горнера

Будемо вважати, що операнди є нормалізованими дробовими числами і подані у формі:

$$X = \sum_{i=1}^{n} x_i 2^{-i}, \quad Y = \sum_{i=1}^{n} y_i 2^{-i}, \quad A = \sum_{i=1}^{n} a_i 2^{-i}, \qquad (7)$$

де $x_i, y_i, a_i \in \{-1,0,1\}$ – цифри операндів.

Вимагатимемо, щоб похибка результату була знакозмінною і за абсолютною величиною не перевищувала половини ваги n-го розряду після коми. Ця вимога буде виконуватися, якщо на i-му кроці цифру z_i результату вибирати таким чином, щоб мало місце співвідношення

$$Z_i - 2^{-i-1} \le 2^{-p}(X_iY_i + A_i) < Z_i + 2^{-i-1}, \qquad (8)$$

де p – число кроків затримки формування цифр результату.

Використовуючи методику, подану в [7], та формули (7), (8), можна отримати алгоритм обчислення Z у неавтономному режимі, що відповідає формулам (2)-(4). У даному випадку $p=3$. Отже, для отримання n розрядів результату після коми необхідно виконати $n+3$ кроків обчислення.

Більш докладніше алгоритм обчислення функції Z можна подати в наступній формі:

1. X_0, Y_0, R_0 надати значення 0.
2. Для $i=\overline{1,n+3}$ виконувати пункти 3-7.
3. $H_i = 2R_{i-1} + 2^{-3}X_{i-1}y_i + 2^{-3}Y_{i-1}x_i + 2^{-3}a_i + 2^{-3-i}x_i y_i$.
4. $X_i = X_{i-1} + x_i 2^{-i}$.
5. $Y_i = Y_{i-1} + y_i 2^{-i}$.
6. $z_i = \begin{cases} -1, \text{ якщо } H_i < -2^{-1}; \\ 0, \text{ якщо } -2^{-1} \le H_i < 2^{-1}; \\ 1, \text{ якщо } 2^{-1} \le H_i. \end{cases}$
7. $R_i = H_i - z_i$.

Тут H_i і R_i – допоміжні змінні.

В даному випадку час обчислення поліномів буде визначатися формулою

$$T = (n + 19)\,t_{\Phi}, \qquad (9)$$

де t_{Φ} – тривалість циклу формування розряду результату функції Z.

Значення t_{Φ} несуттєво перевищує $t_{\text{М}}$, що видно з порівняння формул для обчислення змінної H_i (див. формули (2) і (5) та п. 3 наведеного алгоритму). Це доводить ефективність використання ОМ, що реалізують функцію Z.

4. Порівняння апаратурних витрат для різних реалізацій обчислювального модуля

Для порівняльної оцінки витрат ресурсів ПЛІС при використанні пристроїв з паралельною і послідовною передачею даних при обчисленні Z проведено моделювання двох варіантів реалізації обчислювачів на базі мікросхеми EP2C35F672C6 сімейства Cyclone II фірми Altera у середовищі проектування Quartus II Version 9.1 Build 304. Результати моделювання, відповідно до звітів про компіляцію, наведені у табл. 1 (через роздільник показаний відповідно ресурс для побудови обчислювачів і загальний ресурс ПЛІС певного виду).

Табл. 1. Використовувані ресурси ПЛІС

Ресурс ПЛІС	Блок обробки операндів	
	Паралельний	Квазіпаралельний
Логічні елементи	502/33216 (1,5%)	695/33216 (2%)
Регістри	0/33216 (0%)	271/33216 (0,8%)
Блоки множення	32/70 (46%)	0/70 (0%)
Виводи	256/475 (54%)	10/475 (2%)

Для паралельного блоку помножувач 64x64 реалізований на базі вбудованих у ПЛІС швидкодіючих блоків множення.

На підставі отриманих результатів з табл. 1 можна зробити висновок, що витрати ресурсів ПЛІС більші для пристрою паралельного типу. З огляду на те, що при даній організації обчислень практично половина виводів ПЛІС використовується тільки для одного каскаду обчислювача поліномів, застосування режиму паралельного введення операндів навряд чи є доцільним. Більш реальним є режим попереднього введення у визначеному порядку 64-розрядних операндів з подальшою їх обробкою у паралельному пристрої. Час обчислення поліномів буде визначатися режимом введення операндів.

5. Висновки

Досліджена можливість підвищення ефективності систем з безпосередніми зв'язками між ОМ за рахунок реалізації неавтономних обчислень при порозрядній передачі даних між модулями системи.

У порівнянні з системами, в яких передача даних між ОМ проводиться паралельним кодом, для реалізації систем з порозрядною обробкою даних на базі квазіпаралельних ОМ потрібно менше ресурсів ПЛІС. Заощаджуються ресурси ПЛІС, збільшується ймовірність розміщення системи на одній мікросхемі, що забезпечує підвищення надійності системи, зменшення енергоспоживання та габаритів. Обмін даними між ОМ всередині мікросхеми виконується швидше, ніж між компонентами системи, що реалізовані на різних мікросхемах. Це дає потенційну можливість підвищити частоту тактування, що, у свою чергу, прискорює обробку інформації.

Таким чином, отримані результати підтверджують ефективність застосування неавтономних методів порозрядної обробки інформації в обчислювальних системах на кристалі.

Список літератури

1. Жабин В.И. Эффективность потоковых вычислений в системах с непосредственными связями, реализованных на ПЛИС / В.И.Жабин, В.В.Жабина, М.А. Безгинский // Вісник НТУУ "КПІ". Інформатика, управління та обчислювальна техніка: Зб. наук. праць. – К.: ВЕК+. – 2012. – №55. – С. 149-156.
2. Жабин В.И. Выполнение последовательностей зависимых операций в режиме совмещения / В.И.Жабин. // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка: Зб. Наук. Пр. – К.: Век+. – 2007. – №46. – С. 226-233.
3. Жабин В.И. Построение быстродействующих специализированных вычислителей для реализации многоместных выражений / В.И.Жабин, В.И.Корнейчук, В.П.Тарасенко // Автоматика и вычислительная техника. – 1981. – №6. – с. 18-22.
4. Палагин А.В. Реконфигурируемые вычислительные системы: Основы и приложения / А.В. Палагин, В.Н. Опанасенко. – К.: Просвіта, 2006, 280 с.
5. Каляев И.А. Архитектура семейства реконфигурируемых вычислительных систем на основе ПЛИС / И.А. Каляев, И.И. Левин, Е.А. Семерников // Искусственный интеллект. – 2008. - № 3. – с. 663-674.
6. Максфилд К. Проектирование на ПЛИС. Архитектура, средства и методы / К.Максфилд. – М.: Издательский дом «Додэка-XXI», 2007, 408 с.
7. Жабин В.И. Некоторые машинные методы вычисления рациональных функций многих аргументов / В.И. Жабин, В.И.Корнейчук, В.П.Тарасенко // Автоматика и телемеханика. – 1977. – №12. – С. 145-154.

УДК 004.7

СТОРОЖУК В.О.,
ДОЛГОЛЕНКО О.М.

РЕКОНФІГУРОВАНИЙ ПОМНОЖУВАЧ ЧИСЕЛ З ПЛАВАЮЧОЮ КРАПКОЮ

В даній статті описана реалізація помножувача з реконфігурованою структурою для виконання операцій з плаваючою крапкою. Помножувач має однорідну структуру. Він реалізований як набір комбінаційних схем, без використання елементів пам'яті й не потребує мікропрограмного керування.

This article describes the implementation of reconfigurable multiplier structure to perform floating point. Multiplier has a homogeneous structure. It is implemented as a set of combinational circuits without using memory elements and do not require firmware control.

1. Вступ

Операція перемноження чисел в формі з плаваючою крапкою є однією з найбільш уживаних операцій, що виконується в мікропроцесорному ядрі. У зв'язку з цим від швидкості виконання цієї операції в значній мірі залежить швидкодія всього мікропроцесорного ядра.

Безліч цілих чисел нескінченна, але ми завжди можемо підібрати таке число біт, щоб представити будь-яке ціле число, що виникає при вирішенні конкретної задачі. Безліч дійсних чисел не тільки нескінченна, але ще і безперервна, тому, скільки б ми не взяли біт, ми неминуче зіткнемося з числами, які не мають точного представлення. Числа з плаваючою комою - один з можливих способів представлення дійсних чисел, який є компромісом між точністю і діапазоном прийнятих значень.

Стандарт IEEE 754 [1] представляє числа з плаваючою крапкою як десяткові числа з основою 2 в експоненційному форматі. В IEEE числу з плаваючою крапкою (Рис.1) виділяється 1 біт на знак, 8 біт на порядок і 23 біта на мантису, або дробову частину числа. Порядок розшифровується як ціле число без знака, що допускає як додатну, так і від'ємну експоненти [1]. Дріб представляється як 24 бітове двійково-десяткове (основа 2) число, де найстарший біт відповідає значенню 1 (2^0), наступний біт ½ (2^{-1}), і так далі. Двійкова крапка в уявленні мантиси завжди розташована після біта *MSB*-найбільшого значущого біту мантиси [1]. При нормалізації результату виконання арифметичної операції мантиса результату зсувається вліво або вправо, з відповідною корекцією значення порідку [6], поки в абсолютному значенні мантиси *MSB* не буде дорівнювати 1. Це значення не зберігається в пам'яті, а відновлюється в арифметичному пристрої перед виконанням операції.

Для плаваючої крапки з подвоєною точністю на порядок виділяється 11 біт, а на мантису - 52 біт. Формат числа з плаваючою крапкою IEEE показаний на Рис. 1.

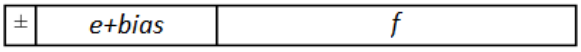

Рис. 1 Формат представлення чисел з плаваючою крапкою

2. Розробка блоку помножувача мантис з реконфігурованою структурою

Розроблюваний блок помножувача мантис орієнтований на обробку всіх форматів чисел з плаваючою крапкою, що передбачені стандартом IEEE Std 754™. Таких форматів п'ять: половинна точність (SF) - 16 розрядів, одинарна точність (F) - 32 розряда, подвійна точність (DF) - 64 розряда, подвійна розширена точність (DEF) - 80 розрядів та учетверину точність (QF) (128 розрядів). Для досягнення цієї мети блок помножувача мантис повинен мати можливість оброблювати, відповідно формату: 12, 25, 54, 66 та 114 бітні мантиси (разом із знаковим та прихованим бітом). Для побудови такого універсального швидкого помножувача мантис, без використання засобів мікропрограмного управління, найбільше підходить алгоритм *Baugh-Wooley* [2]. У алгоритмі Бо-Вулі добуток чисел в доповняльному коді записується наступним чином:

$$A\times B = -2^{2n-1} + \left(\overline{a_{n-1}} + \overline{b_{n-1}} + \right.$$
$$\left. + a_{n-1}b_{n-1}\right)\times 2^{n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j \times 2^{i+j} +$$
$$+ a_{n-1}\sum_{j=0}^{n-2} 2^{n+j-1} + b_{n-1}\sum_{i=0}^{n-2}\overline{a_i}\times 2^{n+i-1} +$$
$$+ (a_{n-1} + b_{n-1})\times 2^{n-1}$$

Часткові добутки зводяться до такого вигляду, який забезпечує максимальну регулярність масиву, що дуже зручно при мікросхемній реалізації. По ходу множення часткові добутки (ЧД), що мають знак «мінус», зсуваються до останньої ступені підсумовування. Операція віднімання ЧД заміняється додаванням їх інвертованих значень. Недоліком схеми можна вважати те, що в останньому рядку матриці необхідний додатковий суматор, через що регулярність схеми порушується. Загалом, у схемі використовується *n(n-2)+4* повних суматорів та *(n-1)* напівсуматорів.

Розглянемо роботу схеми на прикладі виконання операції помноження мантис: *p=x*y*.

Реконфігурування блоку помножувача мантис на обробку потрібного формату операндів здійснюється шляхом доповнення молодших розрядів мантис x та y нулями до розміру мантиси за форматом QF. Таким чином, в блоці помножувача мантис завжди виконується перемноження мантис *114*114*. Доповнені до формату *QF* мантиси *x* та *y* повинні подаватися на входи блока помножувача мантис так, як це показано на схемі з рис. 2.

Схема помножувача мантис складається з 114*114 двухвходових елементів *&*, на кожному з котрих обчислюється один розряд проміжних добутків (деякі з цих елементів, відповідно до алгоритма *Baugh-Wooley*, на рис. 3 змінені на елементи *NOT-&*). Обчислені таким чином розряди проміжних добутків підсумовуються за допомогою однорозрядних повних суматорів (*FA-full adder* на рис. 2).

Обчислений за алгоритмом Baugh-Wooley добуток потребує корекції. У нашому випадку корекція добутку полягає в тому, що в 0-ий та 113-ий розряди добутку повинна бути добавлена «1», як це показано на рис. 2.

На заключному етапі перемноження мантис здійснюється розповсюдження переносів в старшій половині обчисленого добутку. Для цього використовується суматор з обхідними переносами, схема якого приведена на рис. 3.

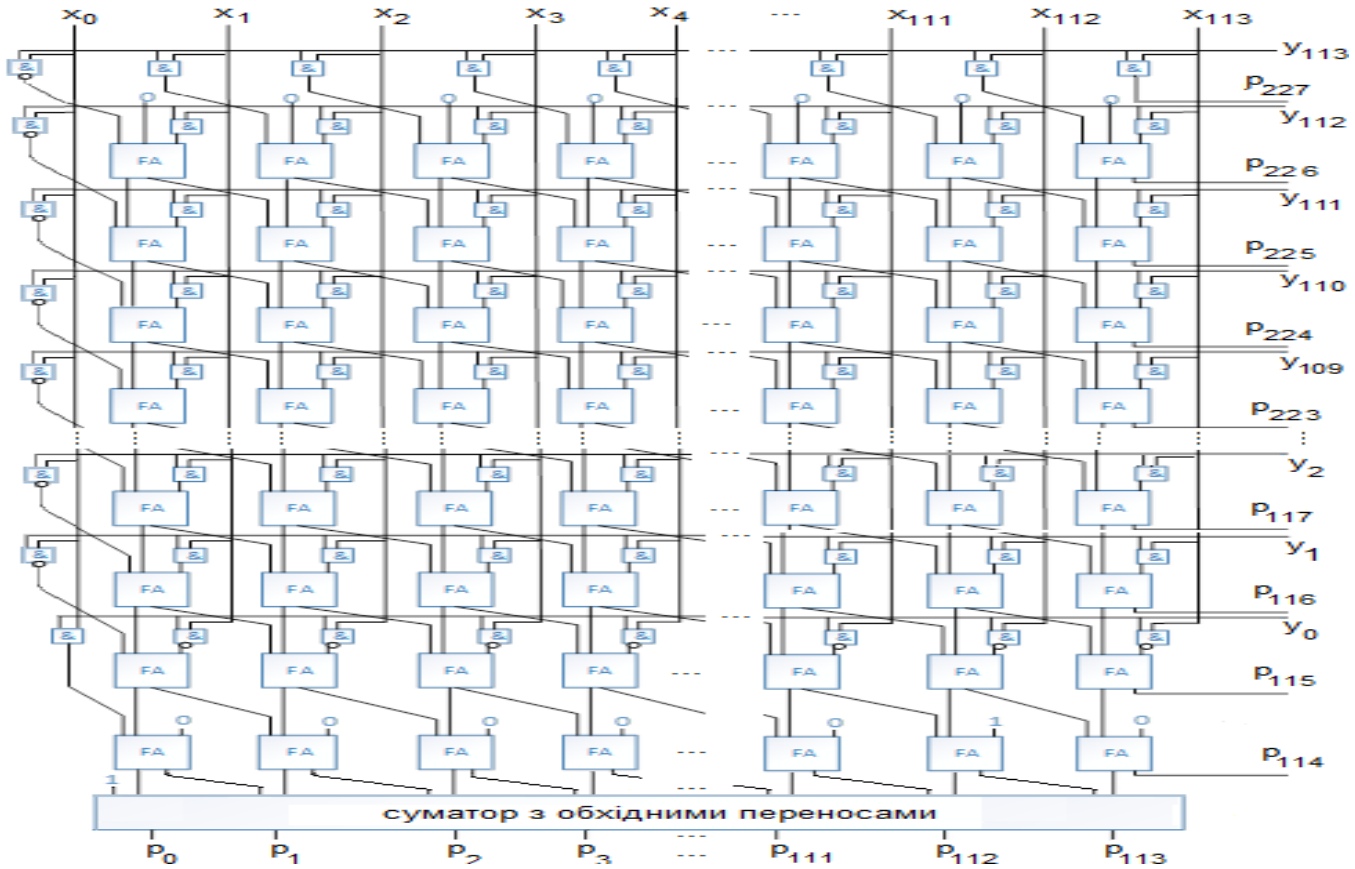

Рис. 2 Функціональна схема блоку помножувача мантис з реконфігурованою структурою

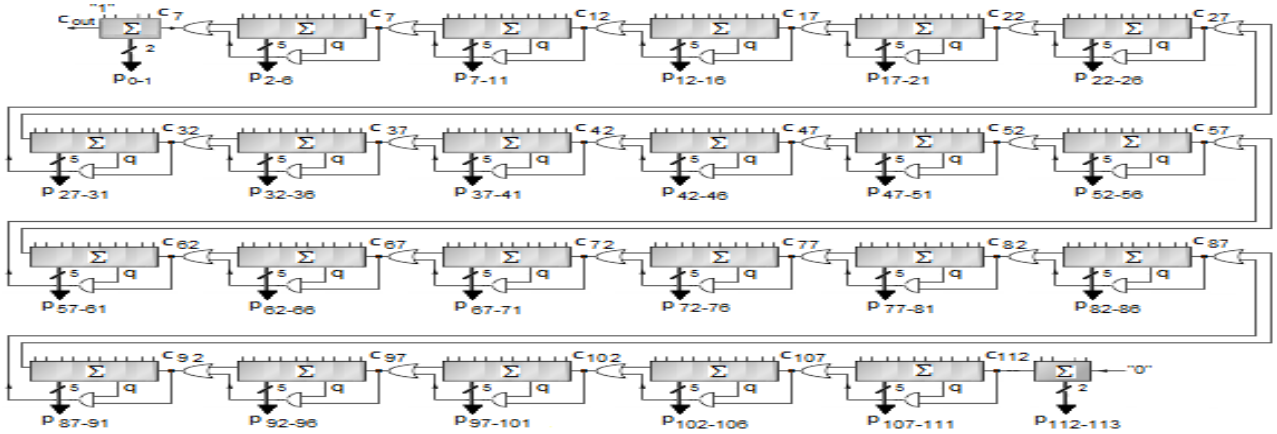

Рис.3 Функціональна схема суматора з обхідними переносами з блоку помножувача мантис

Ідея побудови такого суматора полягає в розбитті схеми *n* –розрядного суматора на групу суматорів меншого розміру, так на схемі з рис. 2 114–розрядний суматор розбитий на 22 п'ятирозрядних суматора всередині та 2 двухрозрядних суматора по краям. Кожен з таких суматорів реалізується за схемою суматора з наскрізними переносами, але відрізняється тим,

що в усіх цих суматорах, крім крайніх, додатково підраховується функція:

$$q = q_0 \& q_1 \& q_2 \& q_3 \& q_4,$$

де: $q_i = a_i \vee b_i$, a_i та b_i - двійкові сигнали на входах i – го розряду п’ятирозрядних суматорів.

Значення функції q використовується для організації обходу вхідним переносом свого суматора, одночасно з його наскрізним розповсюдженням.

Як тільки вихідний перенос з менш значущого суматора сформований, він не тільки подається на вхід перенесення наступного суматора, але також через логічний елемент *І* (якщо сформоване в цьому суматорі $q = 1$) подається і на вхід перенесення наступного суматора і т. д.

3. Блок розпаковки операндів

Схема реалізації блоку розпаковки операндів приведена на рис. 4.

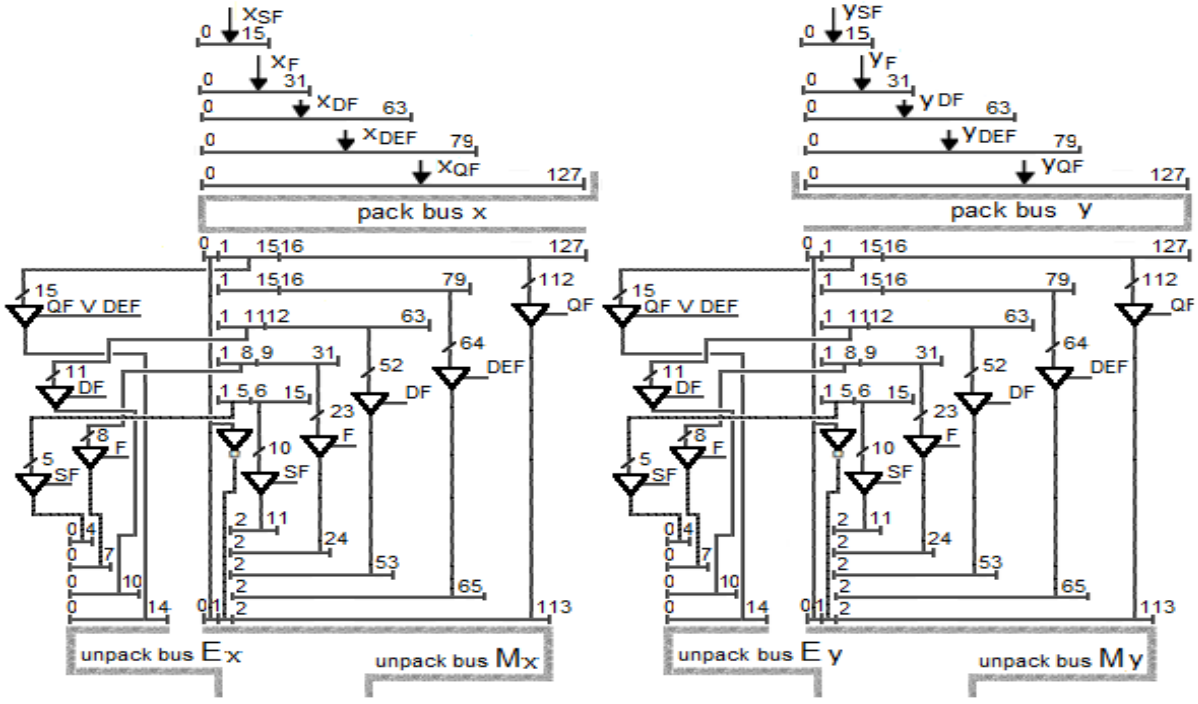

Рис. 4 Функціональна схема блоку розпаковки операндів

Основу блоку розпаковки операндів складають дві вхідні 128-розрядні шини упакованих операндів: pack bus x та pack bus y та чотири вихідні шини розпакованих операндів: 114-розрядні шини мантис операндів *unpack bus Mx* та *unpack bus My*, а також 15-розрядні шини порядків операндів unpack bus Ex та unpack bus Ey. Незалежно від формату оброблюваних операндів x та y вони завжди, на початку виконання операції перемноження, поступають із блоку регістрів з плаваючою крапкою загального призначення в старшу частину вхідної шини блоку розпаковки операндів. Розпаковка операндів здійснюється шляхом виконання потрібних зв'язків між вхідними шинами упакованих операндів та вихідними шинами порядків і мантис блоку розпаковки. Для створення потрібних зв'язків, в залежності від оброблюваного формату використовуються буфери з трьома станами, що створюють конструктивне «або» на вихідних шинах блоку. Для запобігання відмови блоку регістрів з плаваючою крапкою загального призначення, при розробці блока керування реконфігурованого помножувача з плаваючою крапкою, потрібно забезпечити умову, що в кожний момент часу, в залежності від оброблюваного формату операндів, повинен бути відкритим тільки один з буферів, що поєднуються конструктивним «або» на вихідній шині блоку.

При розпаковці операндів в 0-ий розряд кожної з вихідних шин мантис з відповідної вхідної шини напряму заноситься знак операнда, а в 1-ий розряд цих же шин, за допомогою інверторів, із знаків операндів відновлюються приховані біти.

4. Блок нормалізації та округлення результату

Якщо в результаті виконання операцій в блоках обробки мантис та порядків блок керування перемножувачем з плаваючою крапкою виявив наступні ситуації: ознака *Zero* (ознака рівності нулю мантиси результату), *NaN* («не число»), ±∞ («нескінченність»), то блок керування зразу ж ініціює виконання операції пакування числа відповідного результату, пропускаючи роботу блоку нормалізації.

В іншому випадку відбувається нормалізація результату. Функціональна схема блоку нормалізації приведена на рис. 5.

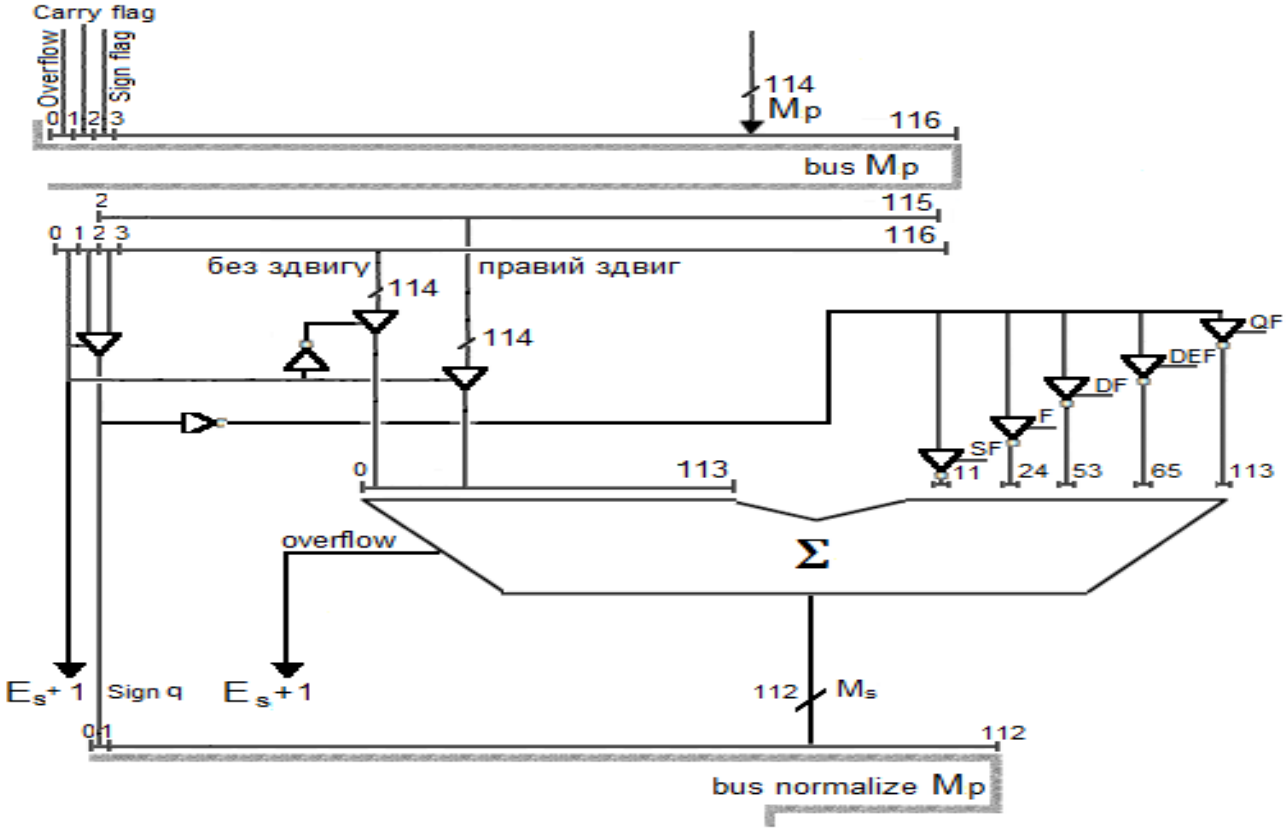

Рис. 5 Функціональна схема блоку нормалізації результату

Основу блоку нормалізації складають вхідна 117-розрядна шина bus Ms, що є вихідною шиною блоку обробки мантис і містить: в 0-му розряді сигнал *Overflow*, в 1-му розряді сигнал *Carry flag*, в 2-му розряді сигнал *Sign flag*, в 3 – 116 розрядах значення мантиси результату та 113-розрядна вихідна шина bus normalize Ms, що містить в 0-му розряді сигнал *Sign s*, а в розрядах 1 – 112 нормалізовану мантису без прихованого біта.

При перемноженні двох нормалізованих чисел з діапазону: $1 \le |x,y| < 2$ мантиса результату може знаходитися тільки в двох станах:

1. Мантиса результату є нормалізованою. При цьому мантиса результату повинна передаватися із вхідної шини на вихідну без зсуву з округленням у відкидаємому біті та видаленням прихованого біту.
2. Переповнення розрядної сітки. При цьому повинен виконуватися правий зсув мантиси на 1 розряд з видаленням прихованого біту та округленням у відкидаємому біті, а також в блок керування помножувачем з плаваючою крапкою повинен видаватися сигнал Es+1 для збільшення порядку результату на 1.

Робота блока нормалізації ґрунтується на організації безперебійної роботи конструктивного «або» на перших входах суматора. Для цього вивід з третього стану 115 – розрядного буферу, що виконує правий зсув мантиси результату, та 115 – розрядного буферу, що виконує передачу без зсуву, здійснюється відповідно, прямим та інверсним значенням одного й того ж сигналу Overflow (переповнення розрядної сітки). Така організація роботи схеми виключає одночасний вихід з третього стану обох буферів.

Знак результату передається з вхідної шини на вихідну без використання конструктивного «або». Для його передачі використовується двовхідний однорозрядний мультиплексор без третього стану, на керуючий вхід якого подається сигнал Overflow. При переповненні результату в якості його знаку передається Carry flag, в інших випадках – Sign flag.

Виконання округлення результату здійснюється шляхом подачі потрібного коду на другий вхід суматора в залежності від оброблюваної точності операндів та знаку результату.

Видалення прихованого біта забезпечується зв’язками між виходами суматора та вихідною шиною.

5. Висновки

В даній роботі описана реалізація помножувача з реконфігурованою структурою для виконання операцій з плаваючою крапкою. Розроблено моделюючу програму для дослідження швидкості та точності виконання частовживаної операції добутку чисел в арифметиці з плаваючою крапкою в розроблюваному помножувачі.

Помножувач реалізовано як набір комбінаційних схем, без використання елементів пам’яті, що не потребує мікропрограмного керування.

Виконана розробка буде корисною для подальшого вдосконалення помножувача з плаваючою крапкою та може бути використаною при побудові ядра мікропроцесора з суперскалярною архітектурою [3-6] в якості операційного пристрою.

Список літератури

1. IEEE 754: Standard for Binary Floating-Point Arithmetic [Електронний ресурс] / 3 квітень 2014. – URL: http://grouper.ieee.org/groups/754/.
2. C. R. Baugh and R. A. Wooley. A Two’s Complementpat Parallel Array Multiplication Algorithm. IEEE Transaction on Computers, C-22(12):1045–1047, December 1973.
3. Ajay D. Kshemkalyani, Mukesh Singhal «Deadlock detection in distributed systems», Distributed Computing Principles, Algorithms, and Systems, Cambridge University Press, 2008.ISBN 978-0-511-39341-9.
4. Луцький Г.М. та ін. «Розробка теоретичних основ побудови високопродуктивних комп’ютерних систем з динамічним розпаралелюванням обчислювальних процесів»- Закл. звіт по НДР № ДР 0213U004830. -Київ, 2013-165 с. – Депонований рукопис

5. K. Yeager. The mips r10000 superscalar microprocessor. IEEE Micro, 16(2):28–40, April 1996.

6. Яцун В.О., Долголенко О.М. Блок додавання та віднімання мантис з реконфігурованою структурою. Матеріали наукової конференції студентів, магістрантів та аспірантів «Інформатика та обчислювальна техніка – ІОТ-2016». Київ 25 – 27 квітня 2016, – с. 158-162 (http://fiot.kpi.ua/wp-content/uploads/2016/06/IOT-2016-OT.pdf).

УДК 004.274

СТОРОЖУК О. М.

БЛОЧНЕ РОЗВ'ЯЗАННЯ СИСТЕМ ЛІНІЙНИХ АЛГЕБРАЇЧНИХ РІВНЯНЬ ДЛЯ РЕКОНФІГУРОВНИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ

Запропоновано модифікований спосіб розпаралелювання задач лінійної алгебри для реконфігуровних обчислювальних систем, що ґрунтується на блоковому LU-розкладі для розв'язання систем лінійних алгебраїчних рівнянь. Представлено результати моделювання способу для багаторазового обчислення лінійних алгебраїчних рівнянь великої розмірності, які обґрунтовують ефективність використання реконфігуровних обчислювальних систем для розв'язання задач лінійної алгебри.

A modified method of paralleling computing tasks of linear algebra for high-performance reconfigurable computing is proposed. The results, based on modeling LU-block schedule for repeated calculation of linear algebraic equations are presented. Experimental high efficiency reconfigurable computing in calculation of linear algebra problems is shown.

1. Вступ

У більшості сучасних систем комп'ютерного моделювання для вирішення різного роду завдань виникає необхідність багаторазового розв'язання систем лінійних алгебраїчних рівнянь (СЛАР), що потребує тривалих обчислень. Фундаментальні та прикладні розрахунки лінійної алгебри пов'язані з проведенням матричних обчислень, обсяг яких залежить від складності кінцево-елементної моделі, частоти дискретизації і необхідної точності. Вимоги до цих параметрів неухильно ростуть, і якщо на зорі обчислювальної техніки великими вважалися СЛАР, що складаються з півтора десятків рівнянь, то сьогодні, наприклад, для задач квантової механіки не є рідкістю рішення систем з десятками мільйонів невідомих [1].

Розробка спеціалізованих обчислювачів на основі універсальних процесорів не є оптимальним рішенням, оскільки така спеціалізація найчастіше обмежена ефективністю в якійсь конкретній задачі або проблемної області. Зі ще більшими труднощами пов'язане створення таких систем за допомогою проблемно-орієнтований процесорів (*Application Specific Integrated Circuit, ASIC*), так як тривалість і висока ціна розробки замовних схем виправдовують себе лише в разі їх великих тиражів для задач, які не потребують частої адаптації апаратної архітектури. Ці обмеження можуть бути подолані при використанні реконфігуровних обчислювальних систем (РОС) на базі динамічних програмованих логічних інтегральних схем (ПЛІС), які забезпечують гнучку апаратну архітектуру [2].

Динамічна часткова реконфігурація – це перепрограмування частини ПЛІС в режимі *RunTime*, коли інші її частини продовжують виконувати свої завдання. Розробник має можливість незалежно конфігурувати кожний обчислювальний вузол. Використання динамічної часткової реконфігурація є ефективне, оскільки це дозволяє значно зменшити непродуктивні часові витрати під час перепрограмування кристалу ПЛІС шляхом передавання часткового бітового потоку конфігураційних даних [3].

При використанні реконфігуровних систем основні обмеження визначаються тільки об'ємом внутрішніх ресурсів кристалу ПЛІС. Застосування сучасних ПЛІС з можливістю динамічної реконфігурації дає змогу змінювати модуль, котрий розташований на деякій частині пристрою, доки решта пристрою продовжує виконувати свою задачу. Різні апаратні модулі можуть розділяти в часі спільні фізичні ресурси, і апаратура може адаптуватись під програмні задачі "на льоту", або ж до частини задачі. Це дозволяє, прискорити обчислення та ефективно використовувати місце на платі [4], [5].

Для розв'язання задач лінійної алгебри в обчислювальних системах на базі ПЛІС найважливіше значення мають три фактори: загальний обсяг програмованої логіки, наявність апаратно реалізованих множників і достатньої кількості блочної пам'яті. Перші експериментальні використання реконфігуровних мікросхем почалися після досягнення ними порога еквівалентної ємності в 1 млн. вентилів [6]. Однак в той час через відсутність в кристалі помножувачів, доводилося або вибирати для реалізації ті алгоритми, в яких множення замінювалося послідовним додаванням, або розробляти власні версії апаратно-орієнтованих алгоритмів [7]. Після появи в ПЛІС апаратних блоків множення, стало можливим вирішувати задачі лінійної алгебри більш широкого діапазону. Реалізована в кристалі блочна статична пам'ять великої ємності дозволяє мінімізувати число звернень до зовнішньої динамічної пам'яті, що значним чином позначається на продуктивності обчислень. На відміну від *CPU*, де можливості розпаралелювання обмежені особливостями «жорсткої» архітектури кристалу, ПЛІС мають потенціал для створення реконфігуровних систем та демонструють кращу масштабованість і продуктивність в вирішенні задач лінійної алгебри. Таким чином виникає задача в розробці і застосуванні нових пристосованих методів паралельної реалізації матричних обчислень, відмінних від існуючих методів розпаралелювання, що використовуються в універсальних багатопроцесорних системах.

2. Особливості апаратної реалізації блочного LU – розкладу на ПЛІС

LU-розклад – це один із способів вирішення СЛАР виду $Ax = b$, при якому вихідна матриця А розбивається на добуток трикутних матриць L і U (де L та U є нижня та верхня трикутні матриці відповідно) і методом прямої і зворотної підстановки знаходиться вектор невідомих x [8].

$$A = LU = \begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{n1} & l_{n2} & \dots & l_{nm} \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{nm} \end{bmatrix} \quad (1)$$

Декомпозиція єдина за умови, коли зафіксовані діагональні елементи однієї з матриць L або U. Це означає, що необхідно, щоб діагональні елементи нижньої чи верхньої трикутних матриць були рівні *1*. Якщо зафіксовані діагональні елементи матриці L, то факторизация називається розкладанням Дуліттла, а якщо зафіксовані діагональні елементи матриці U, то розкладанням Краута (рис. 1).

Таким чином, система рівнянь $Ax = b$ зводиться до вигляду $LUx = b$. Обчислення елементів матриць-співмножників L і U виконується відповідно із формулами:

$$u_{11} = a_{11}, \quad u_{1j} = a_{1j}, \; l_{1j} = \frac{a_{j1}}{u_{11}},$$
$$j = 2,3, \; \dots, n;$$

$$u_{ii} = a_{ii} - \sum_{p=1}^{i-1} l_{ip} u_{pi}, i = 2,3,\dots,n, \quad (2)$$

$$u_{ij} = a_{ij} - \sum_{p=1}^{i-1} l_{ip} u_{pj}, i = 2,3,\dots,n;$$

$$l_{ji} = a_{ji} - \frac{\sum_{p=1}^{j-1} l_{jp} u_{pi}}{u_{ii}}, i = 2,3,\dots,n,$$
$$j = i+1, i+2, \dots, n.$$

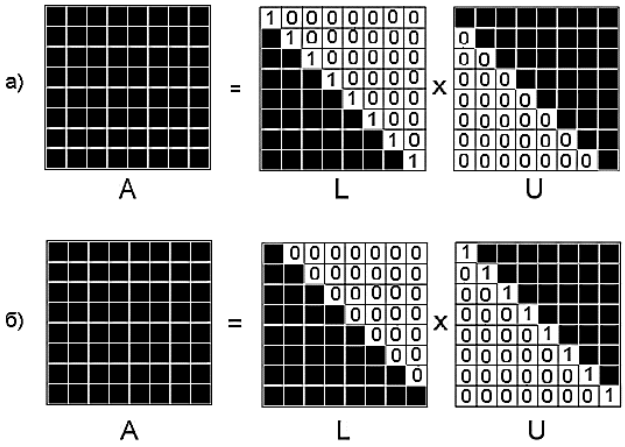

Рис. 1. Схема розкладу: а – Дуліттла; б – Краута

Після отримання мультиплікативного представлення матриці, здійснюється знаходження вектору невідомих x_i шляхом послідовного розв'язання системи рівнянь $LUx=b$, яке проходить в дві стадії. На першій стадії, що є прямим ходом, знаходяться невідомі в системі $LU=b$.

$$y_i = \frac{b - \sum_{p=1}^{i-1} l_{ip} y_p}{l_{ii}}, i = 2,3,...,n, \quad (3)$$

На другій стадії розв'язується система рівнянь $Ux = y$

$$x_i = y_i - \sum_{p=i+1}^{n} u_{ip} x_p, i = 2,3,...,n, \quad (4)$$

LU-розклад має схожість з методом Гауса, але на відміну від нього дозволяє швидко вирішувати *СЛАР* з різними векторами в правій частині.

$$Ax = b_i, i = 1,2,...,m, m \square n. \quad (5)$$

Існує велика кількість варіантів *LU*-розкладу, орієнтованих на певні обчислювальні архітектури та способи зберігання матриці в пам'яті [9]. У нашому випадку реалізовано алгоритм *LU*-факторизації (рис. 2) щільної матриці, який використовується в стандартному пакетах реалізації алгоритмів лінійної алгебри *BLAS* (*Basic Linear Algebra Subprograms*), на основі чого основуються тести реальної продуктивності комп'ютерів *Linpack Benchmark* [10]:

```
double[] LUfactorization(double[,] matrix,
                         double[] rightPart, int n){
    double[,] lu = new double[n, n];
    double sum = 0;
    for (int i = 0; i < n; i++){
        for (int j = i; j < n; j++){
            sum = 0;
            for (int k = 0; k < i; k++)
                sum += lu[i, k] * lu[k, j];
            lu[i, j] = matrix[i, j] - sum;
        }
        for (int j = i + 1; j < n; j++){
            sum = 0;
            for (int k = 0; k < i; k++)
                sum += lu[j, k] * lu[k, i];
            lu[j, i] = (1 / lu[i, i]) *
            (matrix[j, i] - sum);
        }
    }
    double[] y = new double[n];
    for (int i = 0; i < n; i++){
        sum = 0;
        for (int k = 0; k < i; k++)
            sum += lu[i, k] * y[k];
        y[i] = rightPart[i] - sum;
    }
    double[] x = new double[n];
    for (int i = n - 1; i >= 0; i--){
        sum = 0;
        for (int k = i + 1; k < n; k++)
            sum += lu[i, k] * x[k];
        x[i] = (1 / lu[i, i]) * (y[i] - sum);
    }
    return x;
}
```

Рис. 2. Псевдокод алгоритму LU-факторизації

Наведений алгоритм полягає у виконанні вкладених циклів по змінних i, j, k, кожна з яких є одним з індексів двовимірного масиву коефіцієнтів СЛАР [11]. В *LU*-розкладі для забезпечення чисельної стійкості використовується вибір і перестановка ведучого елементу для виключення появи на головній діагоналі нулів і дуже маленьких чисел. Процес обчислення *LU*-розкладу вимагає обчислювати і зберігати елементи нижньої трикутної матриці. Ця обставина

вносить свої особливості в апаратну реалізацію даного алгоритму, що стосуються не тільки управління обробкою даних, але і різної інформаційної структури. На рис. 3. приведена обчислювальна структура, що відповідає базовому підграфу *LU*-розкладу.

Функціональні блоки всередині структури розподілені наступним чином. Блок *max* знаходить максимальний по модулю елемент, що надходить на його вхід і одночасно на вхід буферизованої в *FIFO* черги 1 і-ого рядка. Блок розподілу відповідно до алгоритму виконує ділення поточного елементу рядка $a_{i,k}$ на перший елемент цього ж рядка $a_{k,k}$. Результат ділення $m_{i,k}$, що представляє остаточно сформований елемент $a_{i,k}$ розкладеної матриці, запам'ятовується в блоці пам'яті *RAM* і одночасно надходить через мультиплексор на вихід для подальшої передачі результатів розкладу. Блок множення виконує операцію нормування – множення поточного елемента $a_{k,j}$ на нормувальний коефіцієнт $m_{i,k}$. Блок віднімання віднімає із поточного елемента $a_{i,j}$ результат нормування $m_{i,k} * a_{k,j}$

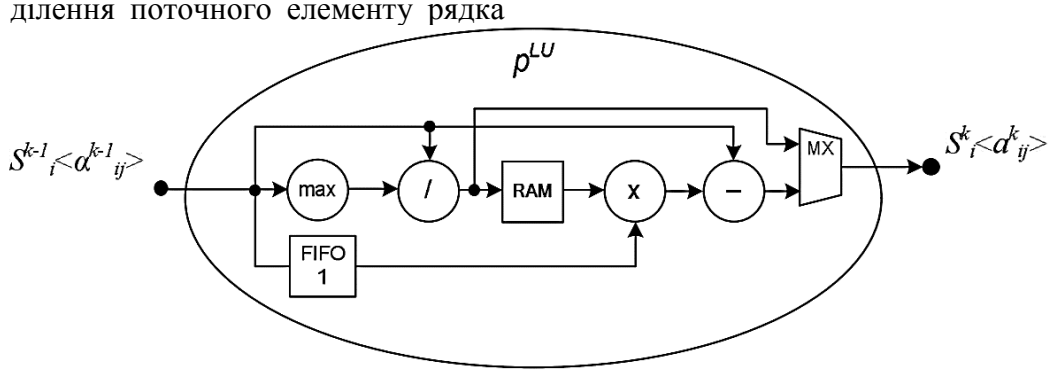

Рис. 3. Обчислювальна структура LU-розкладу

Елементи вихідного рядка *i*, розподілені таким чином, що перший з них належить першому елементу стовпчика нижньо-трикутної підматриці *L*, а всі наступні є елементами першого рядка верхньо-трикутної підматриці *U*. Таким чином, результатом роботи алгоритму на кожній ітерації *k* буде рядок матриці *U* (ведучий рядок *i* для ітерації *k+1*), стовбець матриці *L* і перелічені елементи квадратної підматриці.

Відповідно до наведеного методу була розроблена прикладна програма на мові високого рівня *C*, що реалізує рішення задачі *LU*-розкладу на мультипроцесорній реконфігуровані обчислювальній системі РОС на ПЛІС, що наведена на рис. 4.

Розроблена реконфігуровна система основана на архітектура із загальним комунікаційним середовищем «*Master-Slave*» («Головний» процесор – «Підлеглий» процесор) Дана система містить в собі спосіб управління, де головний (*Master*) процесор виконує функції управління системою та контроль поведінки підлеглих процесорів. Вона містить в основі програмовані процесорні ядра *Nios II*. Обчислення блоків матриці відбувається у програмному режимі під управлінням головного процесору *Nios II Master*. Обчислювальні вузли поєднуються загальним комунікаційним середовищем, а саме, загальною шиною.

Всі процесори підключені до спільної пам’яті через системну шину, для забезпечення безконфліктного доступу до загального ресурсу застосовується система арбітражу. Запис проміжних результатів відбувається через загальну шину в відповідності із алгоритм запису даних у пам’ять. Процесор, що в результаті реконфігурації, підключається до системної шини через систему арбітражу, генерує сигнали вимоги доступу r_i, який поступає на вхід арбітра, якщо шина вільна арбітр видає сигнал дозволу доступу q_i.

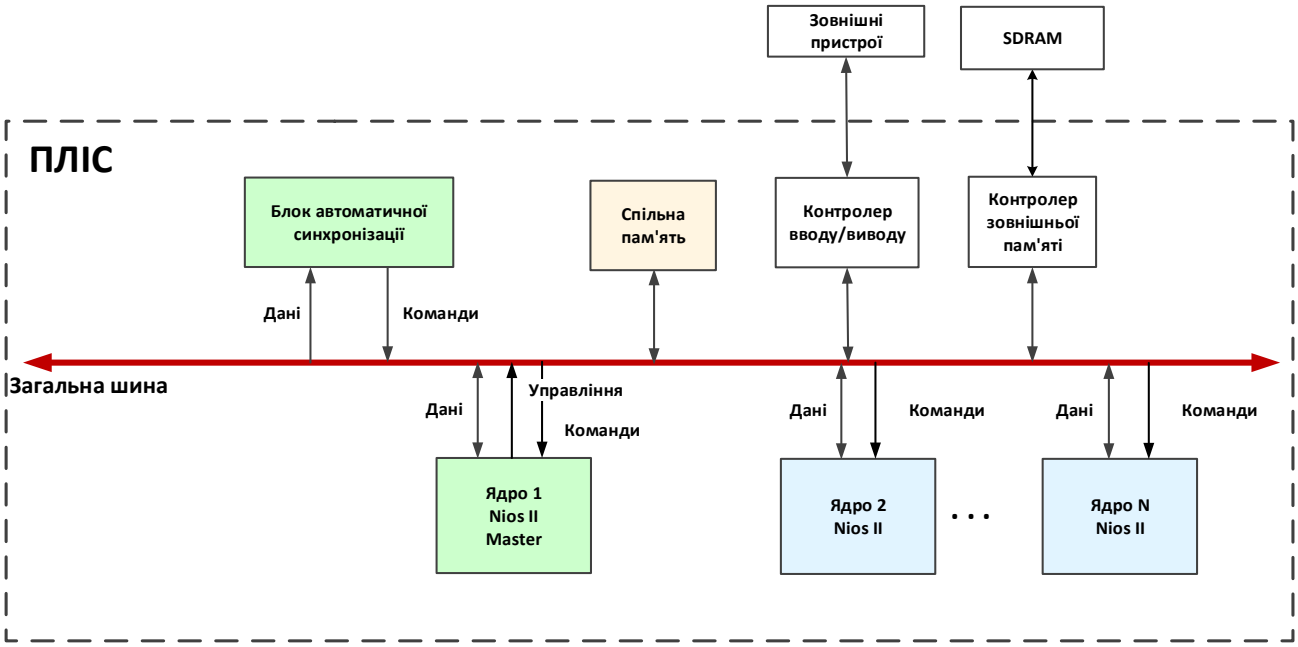

Рис. 4. Архітектура реконфігуровної обчислювальної систем-на-кристалі

Процесор, отримавши доступ до системної шини, розпочинає процедуру ініціалізації та отримання даних для виконання обчислень.

Модифікований алгоритм виконується на основі спеціалізованого модифікованого циклу запису даних, на протязі якого процесор видає на шину адреси адресу поточного блоку даних, адреса фіксується в інтерфейсі пам'яті в спеціальному регістрі адреси, далі процесор видає на шину даних масив слів, які за послідовністю сигналів процесора *w* (*write*) записуються в пам'ять. Далі процесор отримує від інтерфейсу пам'яті сигнали зворотного зв'язку *rd* (*ready*), що визначає закінчення операції запису інформації в пам'ять.

Під час звернення до загального системного ресурсу з боку активних пристроїв виникають так звані конфліктні ситуації, для вирішення яких в системі застосований централізований блок автоматичної синхронізації. Всі пристрої системи, що вимагають доступ до системної шини формують управляючий сигнал r_i, який надходить на входи блоку. У функції блоку входить перевірка зайнятості системної шини, низький рівень управляючого сигналу на шині визначає, що шина зайнята, та високий рівень сигналу – вільна. Якщо системна шина не зайнята, першому у черзі процесору надається доступ, при цьому арбітр формує сигнал зайнятості на шині, та сигнал надання доступу q_i, який надходить на управляючі входи відповідного процесора.

3. Моделювання модифікованого способу блочного LU-розкладу на ПЛІС

Обчислювальні експерименти були проведені на науковому стенді із вбудованою *ПЛІС* компанії *Altera, Cyclone II EP2C35F672C6*, що містить 33216 логічних елементів, 33216 комбінаційних функціоналів та 33216 логічних регістрів на кристалі. Загальна кількість внутрішньої пам'яті на чипі (*On-Chip Memory*) становить 483840 біт, що становить близько 59 Кбайт. На стенді вбудовано *8* Мб пам'яті *SDRAM*, *512* Кб пам'яті *SRAM*, 1 Мбайт пам'яті *Flash*, та інші інтерфейси. Підтримувана адресація зовнішньої пам'яті до 2 Гбайт.

Для розробки обчислювачів системи були обрані програмовані ядра типу *Nios II/s (Small core size)*, так як вони є оптимальним рішенням, оскільки займають не значну кількість логічних елементів в кристалі (в порівнянні із ядром *Nios II/f*), а саме до 1400 *LEs (Logical Elements)*, що становить всього 4% від всього кристалу *ПЛІС*. І при цьому забезпечують значну продуктивність – 0,74 *DMIPS/MHz*, при частоті роботи ядра 165 *MHz*, максимальна продуктивність становитиме 127 *DMIPS (Dhrystone MIPS)*.

Час виконання задачі (матриці порядку $n = 1 \cdot 10^3$) на обчислювальній структурі, що складається з 6 обчислювальних блоків (ядер) (одна ПЛІС) склало 32,1с. Програмна

реалізація *LU*-розкладу матриці тієї ж розмірності на персональному комп'ютері (ПК) (*Intel Core i5*-4690, 3.5 ГГц, 8 Гбайт ОЗУ) показало час 5,7 с.

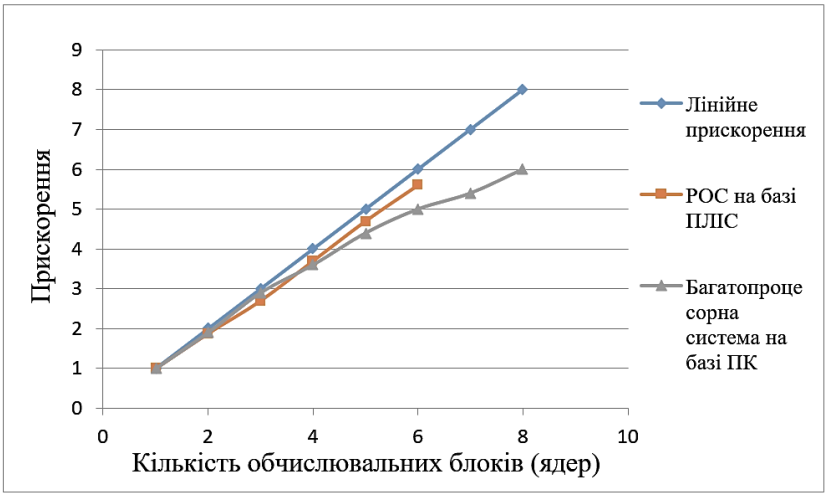

Рис. 5. Залежність прискорення LU-розкладу від кількості обчислювальних блоків на ПЛІС та ядер в ПК

Залежність прискорення від числа задіяних обчислювальних блоків сконфігурованих на ПЛІС для РОС, а також від числа процесорів в багатопроцесорній системі показано на рис. 5, де прискорення розраховується як $K_{Пр}=(T_1 - T_N)/N$, де N – кількість обчислювальних блоків (ядер), а T_1 та T_N час обчислення на одному та N ядрах відповідно.

Наведені графіки, демонструють практично лінійне зростання прискорення на РОС при фіксованому розмірі задачі, та погіршення прискорення в багатопроцесорній системі на базі ПК. Відсутність даних прискорення для 7 та 8 обчислювальних блоків на рис. 5, спричинено недостатність вбудованої пам'яті для розміщення в ній даної кількості програмованих ядер в науковому стенді *Cyclone II*, хоча тенденція графіку прискорення для меншої кількості програмованих ядер на ПЛІС показує практично лінійне зростання. Важливо відзначити, що досягнуті показники залежать тільки від внутрішніх властивостей обчислювальної системи (латентність, кількість обчислювальних блоків), пов'язаних з розмірністю завдання.

В той час як ефективність масштабування задачі в багатопроцесорній системі залежить як від розмірності задачі, так і інших чинників: конфігурація розбиття, розмір порції даних, топологія комунікаційної мережі і так далі. Ефективність *LU*-розкладу в РОС і в багатопроцесорному ПК наведені на графіках, зображених на рис. 6, де ефективність розраховується як $K_{еф} = K_{Пр} / N$, де $K_{Пр}$ – коефіцієнт прискорення, а N – кількість обчислювальних блоків (ядер).

Наведені результати показують, що середня ефективність РОС у вирішенні даної завдання складає 0,97. При збільшенні обчислювального ресурсу підвищення ефективності LU-розкладу на РОС в порівнянні з багатоядерним ПК в середньому склало 14%. Як було зазначено вище, відсутність даних ефективності для 7 та 8 блоків для обчислень на ПЛІС, на рис. 6, спричинено недостатність вбудованої пам'яті для розміщення в ній даної кількості програмованих ядер в науковому стенді *Cyclone II*, хоча з графіка видно що вже починаючи від 4 і більше програмованих ядер, обчислення в РОС показує кращу ефективність в порівнянні з багатопроцесорною системою на базі жорсткої архітектури ПК.

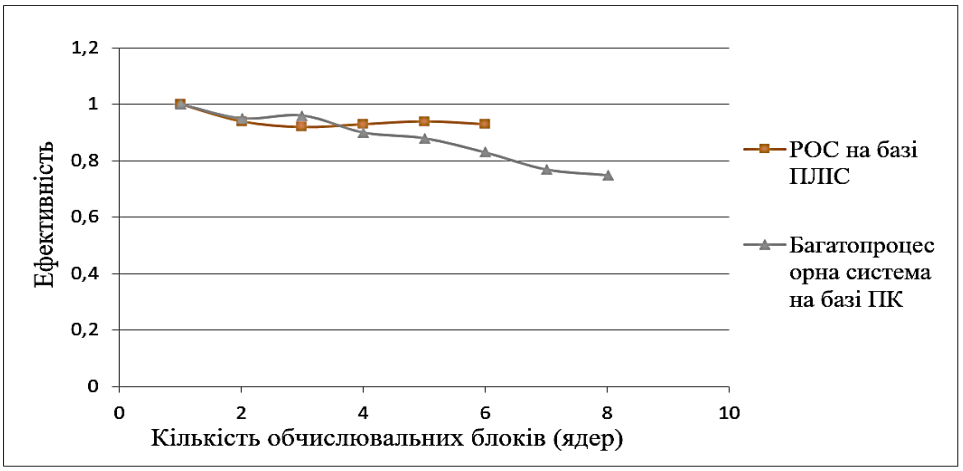

Рис. 6. Залежність ефективності LU-розкладу від кількості задіяних обчислювальних блоків на ПЛІС та ядер в ПК

4. Висновки

Використання реконфігуровних обчислювальних систем з адаптивною до вимог задач архітектурою створює передумови для підвищення ефективності розв'язання задач, в яких виникає необхідність багатократного вирішення систем лінійних алгебраїчних рівнянь великої розмірності. Застосування програмованих логічних інтегральних схем, як елементної бази для створення відповідного середовища, сприяє підвищенню швидкодії обчислень, зменшенню споживної потужності, розмірів та вартості обчислювальної системи.

Проведені в роботі експерименти показали, що можливість прискорення обробки даних за рахунок коректного розташування блоків і їх сегментації дозволяє реалізувати найбільш ефективні обчислювальні структури на ПЛІС для розпаралелювання задач лінійної алгебри.

Дослідження показали прискорення обробки даних під час багаторазових обчислень СЛАР для двовимірних і тривимірних обчислювальних структур реалізованих на ПЛІС. За результатами моделювання отримано середнє значення ефективності обчислень близько 97%, що на 14% більше в порівнянні з організацією обчислень на мультипроцесорній системі.

Ефективність використання запропонованого способу блочного *LU*-розкладу в реконфігуровних обчислювальних системах на базі ПЛІС значно вища в порівнянні з мультипроцесорними системами з жорсткою архітектурою. Ефективність обробки даних в реконфігуровних обчислювальних системах майже не залежить від масштабування обчислювального середовища, що дозволяє спростити масштабування обчислювальних систем залежно від розмірності вирішуваних задач.

Використання динамічної часткової реконфігурації, потребує коректив в підходах до проектування, що призводить до необхідності розробки та пошуку нових методів та модифікації існуючих алгоритмів розв'язання специфічних задач та методів і засобів обробки даних. Це може бути перспективним напрямом подальших досліджень в області підвищення ефективності функціонування реконфігуровних обчислювальних систем.

Список літератури

1. Жуков І. А. Обчислювальна система для розв'язку нечітких СЛАР / І. А. Жуков,vI. А. Клименко // Проблеми інформатизації та управління : зб. наук. праць. – К. : НАУ, 2011. – Вип. 3 (35). – С. 34 – 43.

2. Клименко І. А. Оптимізація реконфігурації в динамічно реконфігурованих обчислювальних системах / І. А. Клименко // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка : зб. наук. праць. – К. : Век+, 2015. – № 63. – С. 93 – 100.

3. Compton K. and Hauck S. Reconfigurable Computing: A Survey of Systems and Software // ACM Computing Surveys ,vol. 34, no. 2, pp. 171–210, June 2002.

4. Xilinx Inc. Press Release: ISR and Xilinx Roll Out Ready-to-Wear SDR. Xilinx Inc., San Jose, CA., 2006, www.fpga journal.com.

5. Programmable Logic Design Line. Xilinx honored for enabling technology in the ALICE experiment at CERN. Xilinx Inc., San Jose, CA., 2008, http://www.pldesignline.com/news/207101017.

6. Сергиенко А. М. Процессор для безошибочного вычисления обратных матриц / А. М. Сергиенко, В. Л. Лепеха, Т. М. Лесик. // Міжнародна конференція "Високопродуктивні обчислення 2012". – 2012. – С. 305–307.

7. Стрельников О. И. Разработка и исследование аппаратурно-ориентированных алгоритмов для нахождения собственных значений матриц : дис. канд. техн. наук / Стрельников О. И. – Волгоград, 2002. – 167 с.

8. Nicolas J. H. Accuracy and Stability of Numerical Algorithms / Highman Nicolas. – Philadelphia: SIAM, 1961. – 680 p.

9. Ортега Д. Введение в параллельные и векторные методы решения линейных систем / Дж Ортега., 1991. – 376 с. – (Мир).

10. Dongarra J. LINPACK Users' Guide / Dongarra J., Bunch J., Moler C. and Stewart G. W. // SIAM, Philadelphia, PA, 1979, p. 1.4.

11. Баркалов К. А. Методы параллельных вычислений / К. А. Баркалов. – Н. Новгород: Нижегородского госуниверситета им. Н. И. Лобачевского, 2011. – 124 с.

УДК 004.27

ШПАРТЬКО О. В.
КЛИМЕНКО І. А.

ОСОБЛИВОСТІ СУЧАСНОЇ МЕТОДОЛОГІЇ РОЗРОБКИ МІКРОПРОЦЕСОРНИХ СИСТЕМ

Досліджено методологію створення мікропроцесорних систем для обробки інформації, які базуються на використанні сучасних мікроконтролерів. Розглянуто питання структурного і програмного синтезу мікропроцесорних систем на базі сучасної елементної бази.

Researched the methodology of creatitg microprocessor systems for information processing based on the use of modern microcontrollers. The issue of structural synthesis software and microprocessor systems based on modern components.

1. Вступ

Протягом останніх років актуальною є тематика є використання мікроконтролерів для побудови систем обробки інформації. На сьогодні навіть найпростіші електронні пристрої обладнані мікроконтролером. А висока швидкодія і широкі функціональні можливості сучасних мікроконтролерів дозволяють створювати досить потужні системи для рішення завдань різної складності. Мікроконтролери застосовуються досить широко в якості вбудованих систем, починаючи від управління побутовою технікою та системами «*smart house»* і закінчуючи управлінням окремими технічними та технологічними процесами на виробництві. Виходячи з цього, найбільш розповсюдженою галуззю застосування мікроконтролерів є системи керування. Зростання складності розв'язуваних задач накладає певні вимоги до структури і

можливостей систем керування, які стосуються забезпечення високої продуктивності, енергозбереження, відмовостійкості, живучості, адаптивності до класів розв'язуваних задач, простоти масштабування. В свою чергу сучасна елементна база, а саме широка різноманітність мікроконтролерів *MPU* (*Microprocessor Unit*) та програмовні логічні інтегральні схеми (ПЛІС), обґрунтовує певні проблеми, які виникають під час проектування таких систем.

У статті розглядаються особливості сучасної методології створення мікропроцесорних систем (МПС) для вирішення широких класів завдань обробки інформації, зокрема завдань керування, що обумовлені використанням сучасних *MPU* і ПЛІС.

2. Огляд сучасної елементної бази для створення мікропроцесорних систем

При проектуванні мікроконтролерів доводиться дотримуватися компроміс між розмірами і вартістю з одного боку і гнучкістю і продуктивністю з іншого. Для різних додатків оптимальне співвідношення цих та інших параметрів може значно відрізнятися . Тому існує величезна кількість типів мікроконтролерів, що відрізняються архітектурою процесорного модуля, розміром і типом вбудованої пам'яті, набором периферійних пристроїв, типом корпусу і т. д. На відміну від звичайних комп'ютерних мікропроцесорів, в мікроконтролерах часто використовується Гарвардська архітектура пам'яті, тобто роздільне зберігання даних і команд в ОЗУ і ПЗУ відповідно до рис 1.

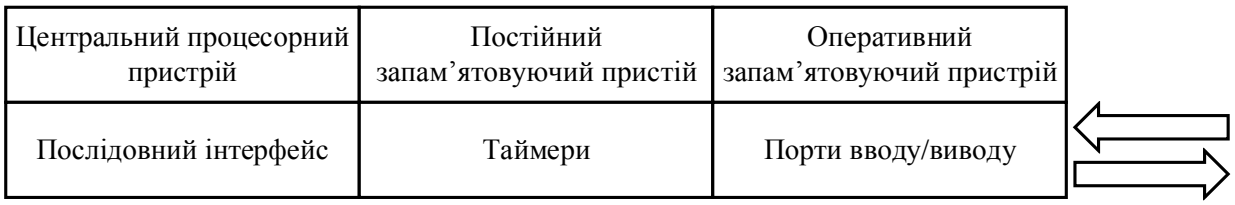

Рис. 1. Структурні компоненти мікроконтролера

Відмінність подібних систем від ПЛІС полягає у внутрішній структурі. Якщо логічна інтегральна схема складається з множини логічних схем та елементів, програмування представлене побудовою зв'язків між ними, то мікроконтролер формує сигнали виходячи з відповідних команд пам'яті програм. Сучасні мікропроцесорні системи дотримуються визначеної архітектури (рис. 2), вона досить легко простежується у керувальних пристроях відомих виробників (*Arduino, Seeeduino, Iskra*).

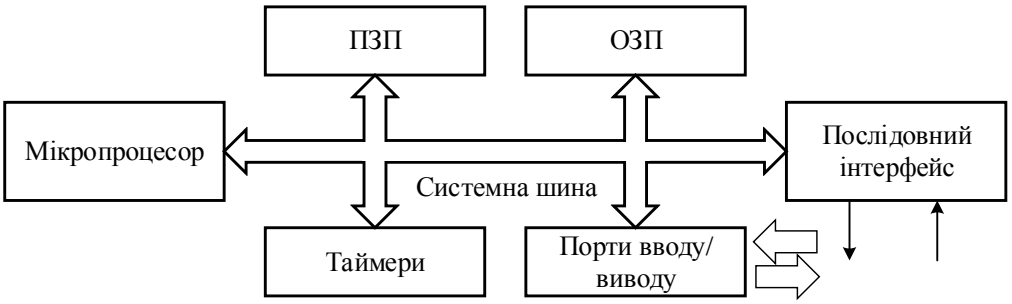

Рис. 2 . – Структура мікропроцесорної системи

Таке поєднання периферійних пристроїв є оптимальним, забезпечуючи користувача необхідними засобами для використання про цьому не обтяжуючи систему з метою підвищення швидкодії.

3. Особливості структурного синтезу сучасних мікропроцесорних систем

Основною проблемою з огляду функціональних можливостей є забезпечення сукупності критеріїв функціонування систем, які забезпечують оптимальне співвідношення проективності і витрачених ресурсів на створення системи обробки інформації. Такими критеріями є висока продуктивність, швидкодія, надійність, гнучкість архітектури, мінімізація розмірів та вартості [2]. Особливістю початкових етапів процесу проектування є аналіз, який дозволяє виявити суттєві ознаки функціонування об'єкта керування і підходи до оптимізації системи з метою отримання найкращих показників функціонування.

Наступним етапом є структурне проектування. Структурне проектування МПС полягає в реалізації регламентованої послідовності дій з проектування апаратних засобів і вибору структури програми, що дозволяють розробити системи керування, які відповідають заданим технічним вимогам. Структурне проектування МПС не завжди гарантує успішне завершення проекту, але значно збільшує його ймовірність.

Зазвичай використовують наступні етапи структурного проектування [3]:

1. Детальний аналіз вимог, що пред'являються до проектованого пристрою, на основі яких формуються технічні характеристики розроблюваного пристрою;
2. Декомпозиція проектованої системи на кілька взаємопов'язаних функціональних підсистем (модулів) з визначенням взаємозв'язків між ними.
3. Розробка структурної схеми апаратури та алгоритму її функціонування;
4. Визначення способу реалізації кожного функціонального модулю (вибір апаратних та програмних рішень);
5. Комплексне налагодження проектованого пристрою.

4. Інструментальні засоби розробки МПС

Особливістю створення і налагодження мікропроцесорних систем є те, що для програмування даної системи як правило, не достатньо одного лише мікроконтролера, це пов'язано з дефіцитом власних ресурсів системи. Для чого розробники мікроконтролерів пропонують спеціальні інструментальні засоби розробки і налагодження МПС [4].

За функціональним призначенням засоби розробки можна класифікувати наступним чином:

1. *Оціночні і демонстраційні плати*. Дозволяють в короткі терміни розробити пристрій.
2. *Схемні емулятори*. Налагоджувальні інструменти, що представляють собою набір апаратно-програмних засобів і дозволяють заміщати собою емульований мікроконтролер у реальній схемі.
3. *Програмні симулятори*. Програмні засоби для імітації роботи мікроконтролера.
4. *Налагоджувач*і. Дозволяють користувачеві контролювати хід виконання програми і бачити відповідність між вихідним текстом, образом програми в машинних кодах та станом всіх ресурсів мікроконтролера.
5. *Емулятори ПЗП*. Програмно-апаратні засоби для завантаження програми за допомогою комп'ютера через один зі стандартних інтерфейсів.
6. *Програматори*. Пристрої, що дозволяють програмувати пам'ять мікроконтролера та програмовані логічні інтегральні схеми.
7. *Інтегровані середовища розробки*. Надають універсальний інтерфейс для роботи з усіма компонентами пакету.

Реальні засоби розробки МПС часто об'єднують в собі різні функції та режими роботи. Сучасні програмні засоби розробки дозволяють розробляти програми з використанням як низькорівневих (*Assemler*), так і високорівневих мов програмування (*C, C ++, Java*), що в комплексі з великою кількістю бібліотек дозволяє з мінімальними зусиллями досягти необхідного результату шляхом виконання елементарних задач.

5. Реалізація основних етапів проектування МПС

Як правило МПС це ієрархічна система, що обумовлює її функціональну й топологічну децентралізацію. Метою функціональної децентралізації системи контролю та управління є зниження складності системи шляхом розділення функцій системи на більш дрібні, тобто на підпроцеси. При цьому поділ функцій здійснюється так, щоб забезпечити задоволення вимог представлених критеріїв. Тому помилки, допущені на етапі формування критеріальною системи оцінок, не дозволяють провести адекватну децентралізацію системи.

У ряді випадків технічний процес легко розбивається на кілька слабов'язанних підпроцесів, кожен з яких може бути реалізований на окремому мікроконтролері, завдяки чому значно знижується складність кінцевої системи обробки інформації. Однак інші технічні процеси більш складні, і їх розбиття породжує сильно взаємопов'язані підпроцеси.

Звідси випливає один з підходів до синтезу системи обробки інформації, такий як оптимальна функціональна децентралізація системи управління або моніторингу, яка має на увазі розбиття цілісного процесу на такі підпроцеси, які слабо пов'язані між собою, тобто пов'язані між собою через мінімальний інтерфейс. При такому підході до проектування "зверху - вниз" здійснюється початкове розбиття цілісної системи на окремі модулі, після чого завдання синтезу окремих локальних систем на базі мікроконтролера не представляє великої складності. Такий підхід добре використовується при завданнях технічного моніторингу, коли цілісний технічний процес контролю параметрів дійсно досить легко розбивається на окремі підпроцеси. При синтезі системи обробки інформації не завжди вдається здійснити таке розбиття. Це породжує більш складні підходи до задачі формалізації проектування системи обробки інформації.

Одним з методів проектування складної системи управління є генерування усіх можливих варіантів побудови системи з метою вибору найкращого з них за кількістю максимально можливих значень критеріїв [3]. При традиційному підході до проектування з представлених законів управління в підсистемах необхідно було б необхідно виключити ті, які при дії на підсистему зовнішніх чинників не забезпечують потрапляння в задану область розкиду вихідних змінних при заданих початкових умов. З позиції багатоваріантного підходу це призводить до необґрунтованого звуження безлічі варіантів системи і тим самим не дозволяє сформувати найкращий варіант з вже розроблених систем управління окремими операціями.

Щоб не втратити жодного варіанту системи з кращими значеннями за іншими критеріями, залежить від законів управління в підсистемах, необхідно здійснювати оцінку системи в цілому з урахуванням повної сукупності можливих варіантів її створення.

Такий підхід має місце при автоматизованому проектуванні МПС. Однак його недоліки очевидні – це висока абстрактність і складність математичного апарату дослідження, в наслідок яких часто зникає суть самої синтезованої системи управління. Часові та матеріальні витрати значно зростають, а оптимальним варіантом проектованої системи може виявитися той же самий, який міг бути отриманий при класичному системному підході до проектування.

Інший підхід до синтезу системи обробки інформації полягає в послідовному виконанні таких процедур, як розробка методів формального опису і дослідження функціонування системи і її модулів; створення узагальнених модулів функціонування комплексу технічних засобів; розробка математичних методів розрахунку параметрів комплексу; створення засобів автоматизації розрахунків; розробка методів синтезу складу і системи; розробка методів контролю результатів.

Особливий інтерес представляє методика синтезу МПС, заснована на ієрархічному підході з описом на певному рівні абстракції. Основою цієї методики проектування є процедура генерування

структурних (*S*) і параметричних (*P*) моделей при заданої функціональної (*F*) моделі системи. Опис системи здійснюється за допомогою її функціональної, структурної та параметричної моделі <*F, S, P*>. На основі цих трьох моделей можна вирішувати задачу оптимального синтезу системи за деяким критерієм. Ця методика проектування передбачає системний підхід і інтегрує такі важливі принципи: ієрархічної багаторівневої системи; багаторівневого багатофункціонального моделювання, структурного програмування; систематизованих підходів до прийняття проектних рішень і перевірки їх коректності.

6. Висновки

У даній статті було досліджено методологію та основні етапи розробки системи обробки інформації на базі мікроконтролерів. Такий підхід до створення систем управління значно підвищує швидкодію пристроїв за рахунок розподілення задачі на елементарні процеси, що виконуються на мікроконтролерах циклічно без використання додаткових програмних та апаратних ресурсів

Досвід промислового використання мікрокопроцесорних систем управління показує, що для виявлення несправностей і подальшого відновлення працездатності широко використовуються методи сигнатурного моніторингу, що забезпечують відмовостійкість системи за допомогою програмно-апаратних засобів, вбудованих на кристал або друковану плату.

Список використаних джерел

1. Новиков Ю. В. Основы микропроцессорной техники. Учебное пособие / Ю. В. Новиков, П. К. Скоробогатов., 2009. – 336 с.

2. Онипенко А. П. МЕТОДЫ АНАЛИЗА И СИНТЕЗА РАСПРЕДЕЛЕННЫХ МИКРОКОНТРОЛЛЕРНЫХ СИСТЕМ УПРАВЛЕНИЯ СЛОЖНЫХ ИЕРАРХИЧЕСКИХ ОБЪЕКТОВ / А. П. Онипенко. // Известия Южного федерального университета. Технические науки. – 2001. – С. 44. 4 р

3. Шегал А. А. ПРИМЕНЕНИЕ ПРОГРАММНОГО КОМПЛЕКСА MULTISIM ДЛЯ ПРОЕКТИРОВАНИЯ УСТРОЙСТВ НА МИКРОКОНТРОЛЛЕРАХ / Анна Айзиковна Шегал. // Издательство Уральского университета. – 2014. – С. 8.

4. Гурин А. Программно-инструментальные средства разработки и отладки [Електронний ресурс] / А. Гурин, П. Перевозчиков // РадиоЛоцман. – 2004. – Режим доступу до ресурсу: http://www.rlocman.ru/review/article.html?di=1809.

УДК 004.93(015.7)

АНТОШКИН Р.О.,
КУЛАКОВ Ю.А.

КОНСТРУИРОВАНИЕ ТРАФИКА В ПРОГРАММНО КОНФИГУРИРУЕМЫХ СЕТЯХ.

В наше время сети активно развиваются и требуют постоянного усовершенствования. Среди множества решений стоит выделить программно-конфигурируемые сети (SDN), главная идея которой заключается в отделении функций передачи трафика от функций управления. Реализация концепции SDN на практике позволит предприятиям и операторам связи получить независимый контроль над всей сетью из единого места, что значительно упростит ее эксплуатацию. Эти и остальные преимущества SDN сетей будут рассмотрены в данной статье.

Nowadays networks are actively developing and require constant improvement. Among the many solutions is the Software Defined Networking (SDN), the main idea of which is to separate the functions of traffic transmission from control functions. The implementation of the SDN concept in practice will

allow enterprises and telecom operators to get independent control over the entire network from a single location, which will greatly simplify its exploitation. These and the rest of advantages of SDN networks will be discussed in this article.

Ключевые слова: программно-конфигурируемые сети, управление трафиком.

1. Введение

Основной проблемой сети передачи данных является динамический характер сетевых приложений и их среды. Это означает, что требования к производительности передаваемых потоков данных, например, Quality of Service (QoS), могут меняться со временем. Приложения работают в широком диапазоне сред, т.е. проводных и беспроводных, с различными сетевыми устройствами. Для эффективной работы приложений базовая сеть должна быть достаточно гибкой, чтобы динамически меняться в ответ на любые изменения в требованиях приложения и их окружении. Существующие подходы основаны на статических, сверх зависимых оверлейных сетях, либо требуют изменения приложений в соответствии с производительностью сети.

Важным способом решения этой проблемы является разработка трафика (TE). Это процесс анализа состояния сети, прогнозирования и балансировки передаваемой нагрузки данных по сетевым ресурсам. Это метод, используемый для адаптации маршрутизации трафика к изменениям в состоянии сети. Однако эти методы не обеспечивают дополнительных ресурсов для трафика, который требует QoS. Традиционные методы маршрутизации не обеспечивают какой-либо механизм для распределения сетевых ресурсов оптимальным образом.

Компьютерные сети состоят из множества сетевых устройств, таких как коммутаторы, средние блоки (например, межсетевые экраны) и маршрутизаторы. Операторы сетей должны вручную настроить эти устройства для различных приложений, чтобы реагировать на различные приложения и события в сети. Часто им приходится использовать ограниченные инструменты, такие как интерфейс командной строки (CLI) и иногда инструменты для создания сценариев для преобразования этих политик конфигурации высокого уровня в политики низкого уровня. Это затрудняет управление и оптимизацию сети, что может привести к ошибкам в сети. Другие проблемы с этой архитектурой могут вызвать колебания в сети, поскольку плоскости управления устройств распределены, нововведение затруднено, поскольку поставщики запрещают модификацию базового программного обеспечения в устройствах. Чтобы преодолеть эти проблемы, была введена идея программирования сети Software Defined Networking (SDN).

2. Описание технологии Software Defined Networking (SDN)

Главная идея SDN заключается в отделении функций передачи трафика от функций управления (включая контроль, как самого трафика, так и осуществляющих его передачу устройств). В традиционных коммутаторах и маршрутизаторах эти процессы неотделимы друг от друга и реализованы в одной «коробке»: специальные микросхемы обеспечивают пересылку пакетов с одного порта на другой, а вышележащее ПО определяет правила такой пересылки, выполняет необходимый анализ пакетов, производит изменение содержащейся в них служебной информации и т д. (Рис. 1). Для определения маршрута передачи или недопущения зацикливания трафика устройства, конечно, «общаются между собой», для чего разработано множество протоколов, таких как OSPF, BGP и Spanning Tree, но при этом каждое функционирует автономно.

Согласно концепции SDN, вся логика управления выносится в так называемые контроллеры, которые способны отслеживать

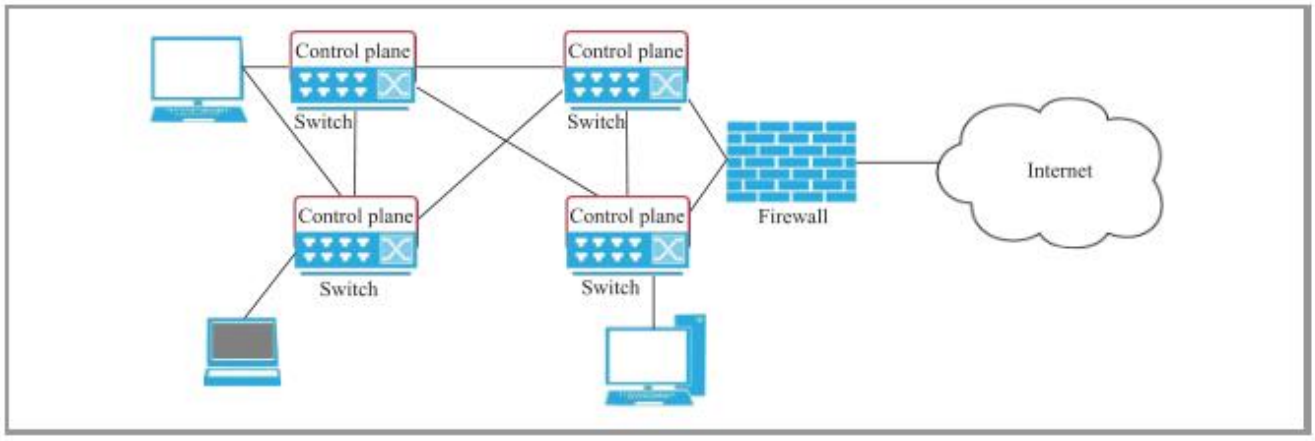

Рис. 1 Архитектура традиционной сети

работу всей сети (Рис. 2). Согласно замыслу разработчиков, SDN позволит программировать сеть как единое целое, а времени, соответственно, сроки внедрения новых приложений и сервисов значительно сократятся.

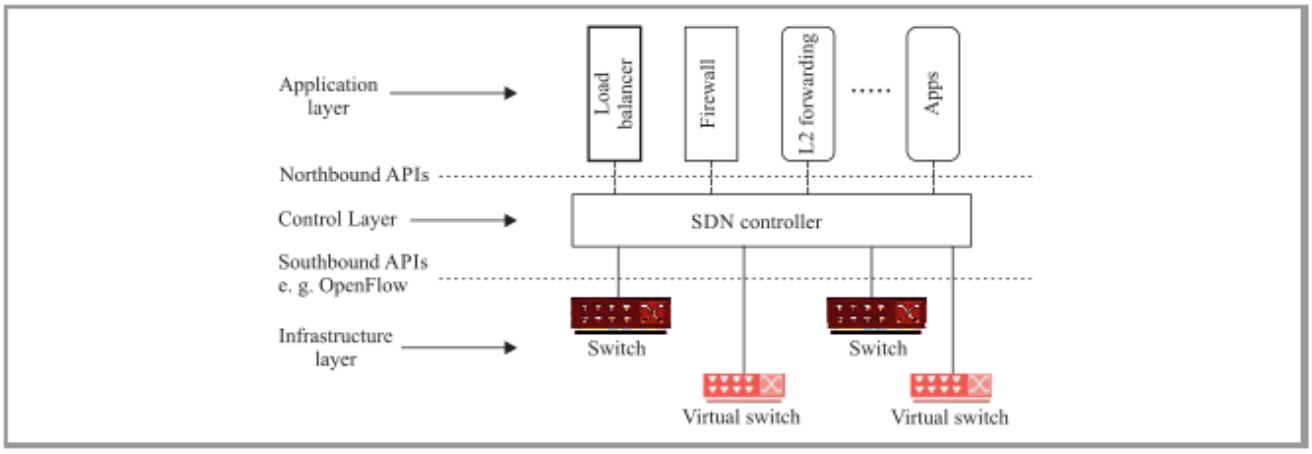

Рис.2 Архитектура SDN сети

администраторам не придется заниматься отдельными устройствами. Главным становится контроллер: он все видит, все знает и раздает сетевым устройствам инструкции по обработке трафика. Самим устройствам больше не надо разбираться в сотнях замысловатых протоколов — достаточно следовать инструкциям контроллера, а значит, они могут быть простыми и дешевыми.

Реализация концепции SDN на практике позволит предприятиям и операторам связи получить контроль над всей сетью из единого места, что значительно упростит ее эксплуатацию. Что не менее важно, конфигурирование сети сильно упростится и администраторам не придется вводить сотни строчек кода отдельно для разных коммутаторов или маршрутизаторов. Характеристики сети можно будет оперативно изменять в режиме реального

Основным элементом концепции SDN является протокол OpenFlow, который обеспечивает взаимодействие контроллера с сетевыми устройствами (Рис. 2). На «северной» стороне контроллер предоставляет программные интерфейсы (API), наличие которых позволяет владельцу сети или сторонним разработчикам создавать приложения для управления сетью. Такие приложения могут выполнять самые разные функции в интересах бизнес-задач (например, контролировать доступ, управлять пропускной способностью и т. п.), причем их разработчикам не надо знать детали функционирования конкретных сетевых устройств. Благодаря контроллеру, вся сеть, состоящая из множества разнотипных устройств разных производителей, предстает для приложения как один логический коммутатор.

Преимущества, которые дает SDN, очевидны, причем не только для сетей в центрах обработки данных, но и для других типов сетевых инфраструктур. Централизованное управление средой, значительное упрощение обслуживания и модернизации, сокращение времени на обновление программных кодов коммутаторов/маршрутизаторов и внедрение новых сервисов — все перечисленное важно, как для корпоративных сетей, так и для инфраструктур операторов связи. Однако это не повод разом отказываться от преимуществ развиваемого десятилетиями традиционного подхода, когда каждое сетевое устройство наделяется «интеллектом», достаточным для автономного функционирования.

3. Обзор классических решенийконструирования трафика

Классические методы организации трафика основаны на тонкой настройке TE и механизма маршрутизации, таких как ECMP или существующие протоколы маршрутизации, такие как IS-IS или MPL. Протоколы маршрутизации Open Shortest Path First (OSPF) и IS-IS не адаптируются к изменениям в состоянии сети, поскольку вес ссылок является статическим, и при выборе путей эти протоколы не имеют цели достигнуть максимальной производительности. Расширения технической инфраструктуры IS-IS и стандарта OSPF расширяют эти протоколы за счет включения нагрузки трафика при выборе пути. В этих подходах во время объявления состоянии канала связи маршрутизаторы объявляют нагрузку на трафик вместе со стоимостью канала. После того, как маршрутизаторы обмениваются ссылками и трафиком, они вычисляют кратчайший маршрут для каждого пункта назначения. Эти стандарты требуют изменения маршрутизаторов для сбора и обмена статистикой трафика.

Multi-Protocol Label Switching, MPLS, обеспечивает механизм туннелирования. Он создает сквозные соединения между узлами. MPLS может интегрировать метки коротких путей с механизмом IP-маршрутизации, где входящие маршрутизаторы назначают коротким фиксированным меткам для пакетов вместо длинных сетевых адресов. Сетевые устройства используют этот ярлык для пересылки пакетов к месту назначения посредством маршрута с коммутацией меток (LSP). Это уменьшает издержки на поиск таблицы маршрутизации. На основе MPLS-TE, основанной на MPLS-технологии, сначала резервируются ресурсы для сквозного маршрута, а затем передаются данные. Он устанавливает маркированный коммутируемый путь по ссылкам с достаточной пропускной способностью. Этот метод обеспечивает достаточное количество ресурсов для потока. Поскольку MPLS-TE работает с доступной пропускной способностью в одном агрегированном классе, он не поддерживает QoS. Для обеспечения возможности QoS были внедрены технологии MPLS-TE, основанные на DiffServe, которые сочетают в себе технологии управления дифференцированными услугами (DiffServ) и MPLS для обеспечения QoS. По сравнению с обычным протоколом маршрутизации MPLS является более гибким в выборе путей, поскольку он настраивает пути виртуальных каналов для отправки трафика. Недостатком MPLS является то, что сетевые операторы должны управлять распределением ресурсов по каждому пути и изменять конфигурацию сети, чтобы настроить путь согласно передаваемому трафику. Поскольку MPLS-TE передает агрегированый трафик по выделенным LSP, он страдает отсутствием гибкости. В MPLS-TE необходимо использовать резервные ссылки, чтобы в случае сбоя какой-либо ссылки трафик мог передаваться по различным путям.

Большинство обсуждаемых подходов согласуются с тем, что для эффективной разработки трафика необходим общесистемный подход. Когда в объеме трафика происходят кратковременные изменения, решение по управлению трафиком должно быстро решить, как распределить трафик по разным путям, чтобы использовать канал сбалансировано. В таких обстоятельствах, когда структура трафика изменяется часто, важно, чтобы решение по управлению трафиком было стабильным. В противном случае это может вызвать колебания. Колебания трафика могут иметь ряд нежелательных эффектов в сети,

например, переполнение буфера коммутатора, нестандартные пакеты, плохое распределение сетевых ресурсов для пользователей, задержка трафика и ухудшение обслуживания. Решения, которые не имеют вышеуказанных недостатков, трудно реализовать в традиционной сетевой архитектуре, поскольку нам необходимо иметь доступ к глобальной информации в режиме реального времени, что является утомительной работой в этой парадигме. Чтобы найти оптимальное решение, большинство предлагаемых решений основаны на локальных измерениях, т.е. требуют, чтобы сетевые устройства самостоятельно определяли, как отправлять пакеты. Как правило, в традиционных сетях цена связи сохраняется статичной в течение длительного периода времени. Поскольку стоимость канала фиксирована, трафик передается по одному и тому же маршруту, пока не изменятся затраты на связь.

Для того, чтобы техника трафика имела оптимальный эффект в сети, она должна иметь следующие характеристики:

• она должна использовать многолучевое разнесение в сети,

• она должна принимать решения о маршрутизации на основе глобального состояния сети.

• она должна учитывать значения потока.

4. Конструирование трафика в программно конфигурируемых сетях

В сетях на базе SDN контроллер может динамически изменять состояние сети, например, в традиционных сетях стоимость канала для протоколов маршрутизации, таких как IS-IS, сохраняется в течение длительного времени. Если в сети происходит перегрузка, это может привести к плохой доставке данных, пока не изменятся затраты на связь или проблема не будет решена. Однако в SDN эти значения могут быть изменены более динамично, чтобы адаптироваться к изменениям. Можно реализовать более инновационный механизм маршрутизации или изменить существующие протоколы маршрутизации, чтобы они могли динамически меняться в зависимости от состояния сети, чтобы повысить эффективность использования ресурсов, избежать сбоев и перегрузок и улучшить качество обслуживания.

Протокол OpenFlow при идентификации трафика оперирует понятием «потока». Ключевым элементом коммутатора, поддерживающего этот протокол, является таблица потоков (Flow Table). Группа столбцов в левой части таблицы формирует поля соответствия, где указаны характеристики потоков: это могут быть различные параметры, включая MAC- и IP-адреса отправителя и получателя, идентификатор VLAN, номера протокольных портов TCP и UDP, а также другая информация. Эти данные с помощью протокола OpenFlow записывает в таблицу коммутатора контроллер, он же определяет приоритет разных потоков: чем выше приоритет, тем выше соответствующая запись в таблице потоков.Входящие пакеты проверяются на соответствие указанным в таблице параметрам. Если соответствие выявлено, к пакетам применяется действие, которое указано в следующем столбце таблицы. Типичным действием является пересылка пакета на один или несколько выходных портов. Кроме того, коммутатор может изменить содержимое служебных полей пакета, сбросить его, направить для анализа контроллеру и т. д. В случае если совпадение не найдено, пакет сбрасывается или направляется контроллеру, который определит, как следует обрабатывать данный поток, и добавит соответствующую запись в таблицу. Статистика по проходящему трафику — число пакетов, байтов и пр. — помещается в соответствующие поля (на Рисунке 3 они обозначены как Count).

Используя протокол OpenFlow, контроллер добавляет, модифицирует и удаляет записи в таблице потоков. Кроме того, он может запрашивать у коммутатора его характеристики и собранную статистику, конфигурировать коммутатор и его отдельные порты.

Разделение «плоскостей» передачи и управления можно реализовать вообще не затрагивая имеющуюся физическую сеть — задействуя виртуальные коммутаторы наподобие Cisco Nexus 1000v, VMware DVS, IBM 5000v или даже Open vSwitch с открытым исходным кодом (см. Рисунок 4).

Программирование таких коммутаторов с помощью контроллера позволяет создать виртуальную сеть SDN поверх имеющейся физической инфраструктуры. Некоторые эксперты рассматривают этот подход как альтернативу развиваемому ONF, но на самом деле, поскольку описываемая схема не исключает возможности использования стандартного протокола OpenFlow, противопоставлять ее решениям ONF не стоит.

Если в такой сети обычные коммутаторы также будут поддерживать OpenFlow, то к виртуальной сети можно будет подключить и физические серверы. Управление такими коммутаторами тоже можно будет передать контроллеру, если это не войдет в противоречие с принципом разделения физической и виртуальных сетей. Этот пример показывает, что в модели SDN конкретная реализация коммутатора — будь то физическое устройство или программа на гипервизоре — не имеет принципиального значения, главное, чтобы он мог получать и исполнять инструкции контроллера.

В рамках своей стратегичи SDN ряд производителей заявили об открытии функционала операционных систем своих устройств через API. До сих пор настройка сетевых устройств производилась преимущественно через командную строку или Webинтерфейс. Но эти инструменты ограничены той оболочкой программирования, которую предлагает производитель. При наличии API можно использовать более широкий набор инструментов программирования и создавать приложения не только для настройки сетевого устройства, но и для программирования сетевой среды в целом. По сути, этот подход является альтернативой SDN, при этом он обеспечивает доступ к более широкому набору функций сетевых устройств, что потенциально позволяет реализовать больше возможностей, чем заполнение таблицы потоков.

В частности, наличие доступа через API непосредственно к функционалу сетевых устройств позволяет системам типа VMware vCenter программировать сеть — например, задавать настройки VLAN в рамках общих задач по развертыванию и обслуживанию виртуальных машин. Для многих производителей коммутаторов интеграция с системой vCenter чрезвычайно важна, поскольку она упрощает и автоматизирует процедуры конфигурирования сетей для сред виртуализации VMware.

5. Заключение

Благодаря отделению плоскости управления от плоскости данных и помещении ее в сервер, называемый контроллером, становятся доступными более гибкие пути конструирования трафика. Преимущества, которые дает SDN, очевидны, причем не только для сетей в центрах обработки данных, но и для других типов сетевых инфраструктур. Однако это не повод разом отказываться от преимуществ развиваемого десятилетиями традиционного подхода, когда каждое сетевое устройство наделяется «интеллектом», достаточным для автономного функционирования. Однако в скором будущем ожидается переход к SDN архитектуре, хотя скорее всего он будет осуществляться постепенно и предельно осторожно.

Список литературы

1. Коломеец А.Е, Сурков Л.В. Программно-конфигурируемые сети на базе протокола OpenFlow // Электронный научно-технический журнал «Инженерный вестник» МГТУ им. Н.Э. Баумана. 2014. №5. 9 с. / URL:http://engbul.bmstu.ru/doc/711486.html
2. Коломеец А.Е, Сурков Л.В. Моделирование сетей SDN в среде Nicira // Электронный научно-технический журнал «Инженерный вестник» МГТУ им. Н.Э. Баумана».. 2014. №6. 6 с. / URL: http://engbul.bmstu.ru/doc/714126.html
3. Смелянский Р.Л. Программно-конфигурируемые сети // Открытые системы. 2012. №9. URL: http://www.osp.ru/os/2012/09/13032491/

4. OpenFlow Tutorial // OpenFlow.2013.URL: http://archive.openflow.org/wk/index.php/OpenFlow_Tutorial

5. ONF Specification // Open network foundation.2014. / URL: https://www.opennetworking.org/ja/sdn-resources-ja/onf-specifications

6. Thomas D. Nadeau, Ken Gray, SDN: Software Defined Networks, O'Reilly, 2013. pp 1025.

7. Черников А. С.1, Паус А. С. Многопоточная маршрутизации в программно-конфигурируемых сетях // Радиооптика. МГТУ им. Н.Э. Баумана. Электрон. журн. 2016. № 06. С. 35–46.

8. Mohammad R. Abbasi1, Ajay Guleria2, and Mandalika S. Devi Traffic Engineering in Software Defined Networks: A Survey // Department of Computer Science and Application, Panjab University, Chandigarh, India. Journal of telecommunications and information technologies