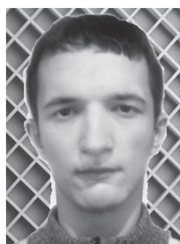
УДК 004.051, 004.272.32, 519.713

ЕФЕКТИВНИЙ ПОШУК ДАНИХ З ВИКОРИСТАННЯМ РЕГУЛЯРНИХ ВИРАЗІВ І SIMD-АРХІТЕКТУРИ ІС



Т. З. Цюра

Постановка проблеми

Стрімкий розвиток науки і техніки зумовив значне зростання обсягів корисної інформації і ускладнив схеми її кругообігу. З'явилася потреба вдосконалення існуючих і створення нових концепцій її обробки.

Важливою функцією будь-якої інформаційної системи є пошук даних. Для надання йому гнучкості доцільно використовувати можливості регулярних виразів. Динаміка мови, властива регулярним виразам і лаконічність їхнього запису дають змогу ефективно вирішувати неординарні завдання пошуку. Тому вони широко застосовуються в різноманітних галузях науки і техніки, зокрема: у бізнес-аналітиці, системах розпізнавання зображень, керування базами даних, добування даних (Data-Mining), NLP (natural language processing), системах виявлення вторгнення, пошукових механізмах, антивірусах, синтаксичних аналізаторах, компіляторах, для надання й обробки XML тощо.

Зважаючи на широке використання, важливість ефективної реалізації даного методу і наявність новітніх розробок серед апаратного забезпечення, уявляється доцільним здійснити його інтерпретацію для виконання на SIMD-архітектурі (NVidia CUDA) і провести аналіз отриманих результатів.

Аналіз останніх досліджень і публікацій. Сучасні дослідження, що були спрямовані на вирішення цього завдання:

- «Regular Expression Matching on Graphic Hardware for Intrusion Detection» – на GPU (graphics processing unit) NVidia GeForce 9800 GX2 з використанням текстурної пам'яті досягнуто пропускну спроможність під час пошуку – 16 Gbit/s, з використанням глобальної пам'яті – 8 Gbit/s [2];
- «Gregex: GPU based High Speed Regular Expression Matching Engine» – реалізовано бібліотеку Gregex, яка досягла показника 126.8 Gbit/s пропускної спроможності текстурної пам'яті на GPU NVidia GeForce GTX 260, що у 210 разів перевищує результат CPU (central processing unit) і в 7.9 раза перевищує попередній результат [5];
- «CANSCID-CUDA» – пропонується рішення, що здійснює також і класифікацію вхідних потоків. Пропускна спроможність обчислення – 600 Mbit/s на GPU Tesla [3];
- «Gnort: High Performance Network Intrusion Detection Using Graphics Processors» – на GT 8600 – бібліотека Gnort – 2.3 Gbit/s [4];
- «Small-Ruleset Regular Expression Matching on GPGPUs: Quantitative Performance Analysis and Optimization» – на GTX 280 із використанням FLEX – 6.92 Gbit/s [6].

Подібні завдання вирішували:

- «GrAVity: A Massively Parallel Antivirus Engine» – створення антивірусної програми з використанням регулярних виразів. На NVidia GeForce GTX 295 досягнуто 20-40 Gbit/s, на 8800 GT - 10 Gbit/s [7];
- «Evaluating GPUs for Network Packet Signature Matching» – у цій публікації регулярні вирази надаються у вигляді XFA, виграш ефективності на GT 8800 GTX відносно CPU P4 3 GHz – 8,6-36 разів [8];
- «iNFAnt: NFA Pattern Matching on GPGPU Devices» – тут регулярні вирази надані у вигляді NFA, пропускна спроможність на GeForce 260 GTX становить 3.5 Gbit/s [9].

Виділення невирішених раніше частин загальної проблеми і постановка завдання

Серед вищезгаданих публікацій, роботи [2] і [5] на першому місці за показником пропускної спроможності механізму пошуку. Регулярні ви-

рази в них надані у вигляді DFA (deterministic finite automata) і пошук збігів виконується над даними, які розміщені в текстурній пам'яті GPU. Реалізація з використанням глобальної пам'яті зустрічається у [2].

Ці типи пам'яті фізично нероздільні, проте пропускна спроможність у публікаціях значно відрізняється. Це зумовлюється тим, що під час доступу до елементів текстурної пам'яті активно використовується кеш мультипроцесорів GPU. Доцільним уявляється також дослідити природу цього процесу і розробити механізм доступу до елементів глобальної пам'яті з ефективним застосуванням кеша.

Як абсолютний лідер за показником швидкодії пошуку, публікація [5] містить результати виконання тестів на сучасному обладнанні (GPU NVidia GeForce GTX 260) з швидкісною шиною даних і підтримкою набору інструкцій версії 1.3.

Серед матеріальної бази не виявилося достойної заміни вищезгаданого GPU, щоб реалізувати поставлене у цій роботі завдання. Ця обставина відкрила для дослідження новий напрям – адаптації й оптимізації для виконання обчислень на GPU, які підтримують набори інструкцій лише версій 1.1 і нижче з метою підвищення їхнього ККД.

Виклад основного матеріалу дослідження. Класичний механізм пошуку з використанням можливостей регулярних виразів [1] передбачає два етапи: побудову (компіляцію) скінченного автомата, на основі регулярного виразу і його виконання для вхідних даних.

Перший етап цього механізму включає в себе:

- здійснення синтаксичного аналізу регулярного виразу;
- надання операндів і операторів виразу у специфічній структурі даних, яка відображала б семантику;
- методи побудови вузлів автомата на основі членів цієї структури;
- об'єднання цих вузлів між собою з урахуванням специфіки операторів;
- кінцеве надання автомата, придатне до подальшої обробки.

Реалізація цього етапу є доволі складною (особливо у випадку DFA), про що свідчить незначна кількість якісних готових програмних продуктів, які б вирішували це завдання. Крім того, вона передбачає часте використання рекурсивних і послідовних алгоритмів – особливо під час здійснення синтаксичного аналізу і взаємодії з вищезгаданою структурою даних для надання семантики виразу. Це зміщує пріоритети під час вибору платформи для реалізації на користь CPU.

Синтаксичний аналіз. Як правило, регулярний вираз записується в інфіксній формі. Необхідно перетворити його у постфіксну форму запису з використанням стека. На цьому етапі остаточно визначаємо властивості операцій, які підтримуватиме механізм компіляції:

- нежадібний унарний правоасоціативний квантифікатор «*», який відображатиме операцію замикання Кліні на множині рядків;
- нежадібний унарний правоасоціативний квантифікатор «+», який відображатиме операцію плюса Кліні на множині рядків;
- бінарна лівоасоціативна елементарна кон'юнкція «&»;
- бінарна лівоасоціативна елементарна диз'юнкція «|»;

Також необхідно визначитися з пріоритетом цих операцій (у порядку спадання):

I. «*», «+»;
II. «&»;
III. «|».

Стек також надасть можливість опрацювання дужок довільного рівня вкладеності у вхідному виразі. Як операнди використовуватимемо латинські літери. За необхідності розширення набору операндів і можливість групування їх у класи легко додається.

Семантичне дерево. Стек, отриманий на попередньому етапі, містить постфіксну форму запису виразу. Необхідно сформувати семантичне дерево виразу. Для цього потрібен новий стек, який збиратиме вузли дерева з операндів і операцій виразу й з'єднуватиме вузли між собою.

Перший операнд відповідає лівій вітці вузла, другий – правій, а операція – вершині. У випадку унарної операції використовуватимемо лише ліву вітку вузла. З отриманої множини вузлів збираємо дерево в порядку спадання пріоритетів операцій. По закінченні цього етапу вершина стека міститиме основу семантичного дерева.

Необхідно спроектувати таку модель надання визначеного скінченного автомата в пам'яті ІС, яка адекватно його відображатиме й легко розширюватиметься відповідно до семантичного дерева виразу.

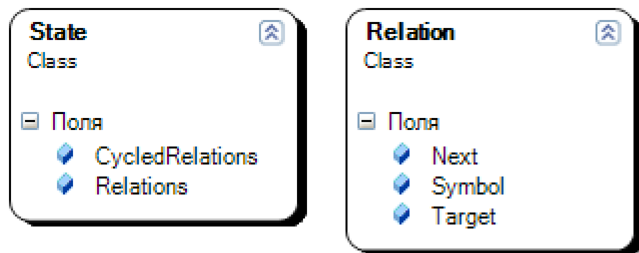
Її будова визначатиме не лише методи взаємодії з деревом, а й загальний функціонал створюваного програмного продукту. Внесення додаткових можливостей на пізніх етапах проектування, яке вимагатиме зміни надання визначеного автомата, є вельми трудомістким, оскільки торкатиметься також механізмів взаємодії з ним. Отже, це найвідповідальніше рішення на даному етапі про-

ектування.

Необхідною і достатньою для виконання цих вимог є модель розподілу кінцевого автомата на стани і зв'язки між ними. Оскільки вираз може містити квантифікатори «+» та «*», ця модель також повинна підтримувати зациклені зв'язки довільної довжини.

Опишемо два класи: «State», який являтиме собою окремий стан кінцевого автомата і «Relation» – зв'язок між станами (див. рисунок).

Клас «State» містить вказівник «Relations» на перелік зв'язків типу «Relation» з іншими станами і лічильник («CycledRelation») зациклених зв'язків.



Класи, які описують модель надання автомата

Клас «Relation» містить вказівник «Next» на наступний зв'язок у списку, символ переходу «Symbol» і вказівник «Target» на сполучуваний стан.

Надання автомата в пам'яті базуватиметься на цих двох класах. Усі операції з ним, у тому числі деініціалізація і звільнення пам'яті, здійснюватимуться шляхом рекурсивного обходу станів автомата. Для цього необхідно розробити механізм виходу із зациклених зв'язків автомата.

Якщо лічильник «CycledRelation» відмінний від нуля, виконаємо рекурсивний обхід відповідних йому зв'язків автомата з поверненням. Рекурсія припиняється, коли у функцію надійшов уже відвіданий стан.

Для побудови вищезгаданого надання в пам'яті необхідно обійти в глибину семантичне дерево виразу. Залежно від оператора або квантифікатора, який міститься в поточній вершині дерева, характер взаємодії з моделлю змінюється. Так, у випадку:

- «*» – копіюємо частину автомата, яка відповідає операнду, формуємо циклічний зв'язок і зв'язок оригіналу з копією;
- «+» – лише формуємо циклічний зв'язок;
- «&» – будуємо новий зв'язок між операндами;
- «|» – доповнюємо список зв'язків одного з операндів зв'язками іншого і знищуємо останній.

Якщо частина надання автомата, яка слугує операндом, не повністю уточнена, рекурсивно виконаємо її побудову. Кожен символ вхідної абетки розглядатиметься як атомарна одиниця і формуватиме один зв'язок між станами.

З отриманої моделі надання необхідно сформувати матрицю, яка вказуватиме наступний стан автомата на основі поточного стану і вхідного символу. Її розмірність по вертикалі рівна кількості елементів вхідного алфавіту регулярного виразу, по горизонталі – кількості станів у наданні автомата.

Заповнимо матрицю перехідних станів з допомогою рекурсивного обходу моделі автомата в пам'яті: для кожного стану знайдемо відповідність між станами, на які вказують його зв'язки, і загальним індексом станів і додамо її до матриці.

Отримана таким чином матриця перехідних станів є кінцевим продуктом компіляції регулярного виразу й використовуватиметься безпосередньо для пошуку даних.

Оскільки другий етап пошуку включає в себе виконання автомата над вхідними даними й аналіз його поточного стану, ефективність цих процесів є критичною. Це зумовлює важливість їхньої оптимізації і паралельної реалізації на SIMD-архітектурі ІС.

Основними напрямами підвищення ефективності обчислення алгоритму на графічному прискорювачі є оптимізація доступу до даних і збільшення числа мультипроцесорів, які виконують обчислення.

Одна когерентна трансакція доступу до глобальної пам'яті займає 400-600 тактів мультипроцесора [10], при цьому можна прочитати від одного до шістнадцяти (для набору інструкцій версії 1.1) машинних слів відразу. Тому виконаємо читання 16 слів прямо у регістри чи спільну для потоків пам'ять і використовуватимемо її як кеш, уникаючи конфлікту банків (bank conflict).

При доступі до текстурної пам'яті активно використовується текстурний кеш графічного процесора [11]. Реалізуємо і цей підхід оптимізації для порівняння результатів.

Отже, у порівнянні показників ефективності братиме участь такий перелік програмних реалізацій пошуку:

- з використанням глобальної пам'яті:
  – без оптимізації;
  – з когерентним доступом і кешем;
- з використанням текстурної пам'яті.

Для підвищення ККД графічного прискорювача здійснимо розподіл потоків у межах блоку й

блоків у межах сітки ядра з оглядом на кількість регістрів, які використовує алгоритм, і версію підтримуваних інструкцій. Це завдання легко вирішується з допомогою програмних засобів NVIDIA CUDA Occupancy Calculator та NVIDIA Visual Profiler. Для кожної з наведених вище реалізацій застосуємо найефективнішу (з отриманих) модель розподілу потоків.

Для дослідження показників пропускної спроможності обробки вхідних даних використовувалася ІС з GPU – NVIDIA GeForce 9800 GT [13] і CPU – AMD Phenom II X4 940 3.0 GHz [14].

- Без оптимізації доступу до глобальної пам'яті на GPU досягнута середня пропускна спроможність обробки 13.3 Gbit/s, на CPU – 3.53 Gbit/s.
- Завдяки когерентному доступу до глобальної пам'яті та її кешуванню в регістри досягнуто 54.2 Gbit/s; з допомогою текстурної пам'яті – 61.6 Gbit/s.

Порівняємо ці результати (див. таблицю) з отриманими в статті «Gregex: GPU based High Speed Regular Expression Matching Engine» [5]:

Порівняння підходів оптимізації пошуку з використанням регулярних виразів

Найменування	Шляхи оптимізації	Пропускна спроможність	
		пам'яті, GB/s	пошуку, Gbit/s
NVIDIA GeForce GTX 260	Текстурна пам'ять, уникнення конфлікту банків	111.9 [12]	126.8 [5]
NVIDIA GeForce 9800 GT	Те ж саме	57.6 [13]	61.6
NVIDIA GeForce 9800 GT	Когерентний доступ до глобальної пам'яті, кешування і регістри	57.6 [13]	54.2
NVIDIA GeForce 9800 GT	–	57.6 [13]	13.3
AMD Phenom II X4 940 3.0 GHz	–	–	3.54

Висновки

Проведено аналіз сучасних досліджень, спрямованих на ефективний пошук даних з використанням регулярних виразів і можливостей SIMD-архітектур. Спроектовано й реалізовано програмне рішення цієї проблеми, яке ефективно використовує ресурси GPU з підтримкою наборів інструкцій версій 1.0 і 1.1 і уможливлює застосування глобальної пам'яті для обміну даними.

Основною сферою його застосування є неоднорідні розподілені обчислювальні системи, які містять різні типи графічних прискорювачів, зокрема: BOINC, FoldingHome, SETI@Home, GIMPS, UGRID та ін.

Отримані під час дослідження результати засвідчують конкурентоспроможність бюджетних GPU перед рішеннями середньої й вищої цінової категорії і сприяють зниженню собівартості проведення масштабних обчислень. Крім того, вони демонструють високу ефективність GPU як невід'ємної частини високонавантаженої пошукової системи.

Перспективним напрямом подальших досліджень є адаптація і застосування підходів оптимізації, які описані у цій публікації, до новітніх GPU з метою підвищення їхнього ККД.

ЛІТЕРАТУРА

1. Введение в теорию автоматов, языков и вычислений / *Хопкрофт Д., Мотвани Р., Ульман Дж.* – М.: «Вильямс», 2002. — 528 с. — ISBN 0-201-44124-1.

2. Regular Expression Matching on Graphics Hardware for Intrusion Detection [Електронний ресурс] / [Vasiliadis G., Polychronakis M., Antonatos S., Markatos E. P., Ioannidis S.]. – Institute of Computer Science, Hellas, Greece. – 2010. – Режим доступу: http://www.ics.forth.gr/dcs/Activities/papers/gnort-regexp.raid09.pdf. – Назва з екрана.

3. CANSCID-CUDA [Електронний ресурс] / [Steffen M., Allada V., Jones P., Zambreno J.]. – Iowa State University, Electrical and Computer Engineering, Ames, IA, USA. – 2010. – Режим доступу: http://viper.eng.iastate.edu/veerendra/files/canscid-battery.pdf. – Назва з екрана.

4. Gnort: High Performance Network Intrusion Detection Using Graphics Processors [Електронний ресурс] / [Vasiliadis G., Polychronakis M., Antonatos S., Markatos E. P., Ioannidis S.]. – Institute of Computer Science, Hellas, Greece. – Режим доступу: http://www.ics.forth.gr/_pdf/brochures/gnort.raid08.pdf. – Назва з екрана.

5. Gregex: GPU based High Speed Regular Expression Matching Engine / [Wang L., Chen S., Tang Y., Su J.]. – Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), Fifth International Conference, Seoul. – 2011. – Бібліогр.: С. 336–370. – ISBN 978-1-61284-733-7.

6. Small-Ruleset Regular Expression Matching on GPGPUs: Quantitative Performance Analysis and Optimization / [Naghmouchi J., Scarpazza P. D., Berekovic M.]. – ICS '10 Proceedings of the 24th ACM International Conference on Supercomputing, ACM New York, USA. – 2010. – Бібліогр.: С. 337–348. – ISBN 978-1-4503-0018-6.

7. GrAVity: A Massively Parallel Antivirus Engine [Електронний ресурс] / G. Vasiliadis, S. Ioannidis // Institute of Computer Science, Foundation for Research and Technology – Hellas, Greece. – 2010. – Режим доступу: http://dcs.ics.forth.gr/Activities/papers/gravity-raid10.pdf. – Назва з екрана.

8. Evaluating GPUs for Network Packet Signature Matching

[Електронний ресурс] / [Smith R., Goyal N., Ormont J., Sankaralingam K., Estan C.] // IEEE International Symposium on Performance Analysis of Systems and Software. – 2009. – Режим доступу: http://research.cs.wisc.edu/vertical/papers/2009/ispass09-xfagpu.pdf. – Назва з екрана.

9. iNFAnt: NFA Pattern Matching on GPGPU Devices [Електронний ресурс] / [Cascarano N., Rolando P., Risso F., Sisto R.] // Politecnico di Torino, Turin, Italy . – Режим доступу: http://ccr.sigcomm.org/drupal/files/p21-2v40n5d2-cascaranoA.pdf. – Назва з екрана.

10. CUDA C Best Practices Guide v4.0 [Електронний ресурс] // NVIDIA Corporation. – С. 54. – 2010. – Режим доступу: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Best_Practices_Guide.pdf. – Назва з екрана.

11. CUDA C Programming Guide v4.0 [Електронний ресурс] // NVIDIA Corporation. – С. 97. – 2010. – Режим доступу: http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf. – Назва з екрана.

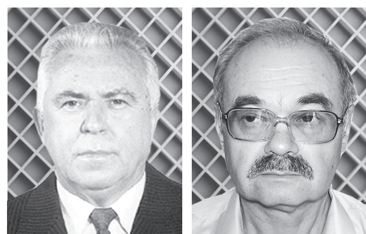
12. GeForce GTX 260 Overview [Електронний ресурс] // NVIDIA Corporation. – 2010. – Режим доступу: http://www.nvidia.com/object/product_geforce_gtx_260_us.html. – Назва з екрана.

13. GeForce GT 9800 Overview [Електронний ресурс] // NVIDIA Corporation. – 2010. – Режим доступу: http://www.nvidia.com/object/product_geforce_9800gt_us.html. – Назва з екрана.

14. AMD Processors for Desktops: AMD Phenom™ [Електронний ресурс] // Intel Corporation. – 2010. – Режим доступу: http://products.amd.com/pages/DesktopCPUDetail.aspx?id=509&AspxAutoDetectCookieSupport=1. – Назва з екрана.

УДК 004.75:004.9:004.7:004.738.5

ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ І МОДЕЛЬ ПОСЛУГ



В.І. Гриценко, *канд. техн. наук,*
О.А. Урсатьєв, *канд. техн. наук*

Вступ. За час, що минув від зародження *Internet,* відбулося переосмислення *web*-технології і завершився її розвиток від механізму надання інформації до надання різного роду послуг, як-от доставка програм кінцевому користувачу, платформ для бажаючих створювати додатки, інфраструктури тощо; з'явилася нова бізнес-модель – модель виробництва і споживання ІТ-послуг [1]. Створено розміщувані прикладні послуги за вимогою, зокрема: *IaaS* – послуга з надання уніфікованих апаратних і програмних ресурсів; *PaaS* – послуга, котра надає можливість розроблювати, тестувати і впроваджувати користувацькі додатки; ПЗ як послуга *SaaS* (*software-as-a-service*) чи ПЗ за вимогою *SoD* (*software on demand*) та ін. з оплатою у міру використання – доступ до ресурсів за моделлю почасової оплати (*pay-as-you-go*) [2–4].

В основі *SaaS* лежить принцип підписки: програмне забезпечення не продається як продукт, а надається в оренду, оплата залежить від кількості користувачів, обсягу трансакцій та інших кількісних показників. Можливість взяти в оренду ПЗ знімає питання про необхідність капітальних інвестицій в інфраструктуру. Головні характерні властивості моделі *SaaS* такі:

- ПЗ працює на боці провайдера;
- умови використання ПЗ поєднують у собі правила ліцензування* і хостингу;
- доступ до програми здійснюється через будь-який браузер чи тонкий клієнт;
- програма підстроюється під специфічні вимоги користувача, а не одноразово конфігурується спеціальним способом.

Таким чином, модель *SaaS*** має очевидні переваги над класичною моделлю розповсюдження корпоративного ПЗ: вона економічніша, не потребує інсталяції і подальшої підтримки ПЗ у середовищі розташування користувача, вкладення значних засобів захисту даних, пошуку й утримання ІТ-персоналу, який мав би підтримувати і розвивати інфраструктуру.

Технологія і середовище прикладних послуг за вимогою

Концепція прикладних послуг за вимогою заснована на сервіс-орієнтованій архітектурі (*SOA*), яка передбачає структуру з трьох елементів (рис. 1): провайдери служб, котрі розміщують

* На відміну від традиційного ліцензування програмних продуктів за «програми за вимогою» клієнт платить за принципом підписки, зобов'язуючись вносити регулярні платежі. Вважається, що головна перевага *SaaS* – низька вартість володіння і запуску в експлуатацію.

**За даними [5], залежно від кількості користувачів і виду ПЗ можна заощадити до 85% сукупної вартості володіння рішенням упродовж трьох років. Також послуга дає можливість перевести частину витрат на ІТ з капітальних в операційні.