УДК 004.43 **doi: 10.26906/SUNZ.2018.6.126**

П. М. Шутка, А. М. Філоненко

Національний технічний університет «Харківський політехнічний інститут», Харків, Україна

ОГЛЯД ПРОБЛЕМНО-ОРІЄНТОВАНИХ МОВ ПРОГРАМУВАННЯ ДЛЯ ПАРАЛЕЛЬНОГО АНАЛІЗУ СТАТИЧНИХ ГРАФІВ

Предметом вивчення в статті є проблемно-орієнтовані мови програмування для паралельного аналізу статичних графів. **Метою** даної статті є огляд підходів до реалізації проблемно-орієнтованих мов програмування на прикладі Green-Marl, OptiGraph, Elixir і Falcon, призначених для аналізу статичних графів. **Завдання:** показати ефективність використання предметно-орієнтованих мов програмування в аналізі статичних графів, якими можуть оперувати не тільки спеціалісти в області програмування, а й фахівці розробки математичних моделей і алгоритмів аналізу даних, зокрема із застосуванням теорії графів; розглянути існуючі DSL для аналізу статичних графів із застосуванням паралельних і розподілених обчислень; відзначити існуючі предметно-орієнтовані мови для побудови алгоритмів обходів графа; порівняти DSL з точки зору виразності паралелізму і застосовності для генерації високоефективних паралельних програм для суперкомп'ютерів і кластерних систем у вигляді зведеної таблиці з основними властивостями мов і їх компіляторів. Використовуваним **методом** є: проведення порівняльного аналізу предметно-орієнтованих мов програмування. Отримані такі **результати:** виявлено рівень ефективності використання предметно-орієнтованих мов програмування в аналізі статичних графів; розглянуто існуючі DSL; проведено порівняльний аналіз DSL. **Висновки.** В статті були розглянуті чотири проблемно-орієнтованих мови програмування, призначених для розробки і реалізації алгоритмів аналізу статичних графів.

Ключові слова: DSL, предметно-орієнтовані, аналіз статичних графів, паралелізм, генерація, таблиця, математичні моделі, виявлення.

Вступ

Аналіз статичних графів використовується в ряді прикладних областей, таких як аналіз веб-графів, аналіз соціальних мереж, біоінформатика, інформаційна безпека та ін. Досить часто реальні графи можуть досягати дуже великих розмірів, так що обробка таких графів вимагає величезних обчислювальних ресурсів і використання технологій розподілених і / або паралельних обчислень. У контексті майбутніх тенденцій на зближення технологій обробки великих даних (Big Data) і паралельних обчислень із застосуванням високо-продуктивних обчислювальних комплексів (ВОК) [1], паралельна обробка великих графів є тією прикладною областю, де злиття цих двох технологічних парадигм виявляє себе найпомітніше. Як і будь-яка інша інженерно-технічна дисципліна, паралельне програмування вимагає певної кваліфікації, яку можна отримати тільки маючи реальний практичний досвід. З іншого боку, фахівці з аналізу даних, як правило, не мають такого досвіду, оскільки їх основне завдання - це розробка математичних моделей і алгоритмів аналізу даних, зокрема із застосуванням теорії графів.

Таким чином, між кінцевими користувачами і існуючими технологіями паралельної обробки графів існує розрив, ліквідувати який покликані спеціалізовані фреймворки, такі як Pregel [2], GraphLab [3], GraphX [4] та ін., і проблемно-орієнтовані мови програмування (DSL). Відмінність підходів в тому, що в DSL використовуються нові синтаксичні конструкції, які можуть бути реалізовані як «зовнішня» мова, тобто із залученням зовнішнього компілятора DSL, так і з допомогою перевантаження операторів базової мови, тоді говорять про вбудовуваний DSL. Другий спосіб часто застосовується при реалізації DSL на базі функціональних мов, наприклад Scala. Графові фреймворки, такі як Pregel, GraphLab, GraphX, як правило, обмежуються наданням простого і зручного інтерфейсу (API), але з досить потужною підтримкою з боку runtime-системи.

В даній статті ми розглядаємо кілька існуючих DSL для аналізу статичних графів із застосуванням технології паралельних (OpenMP, MPI, CUDA і ін.) і розподілених (Hadoop [5], Giraph [6], Spark [7] та ін.) обчислень. Для аналізу були обрано кілька існуючих DSL, зокрема Green-Marl [11, 12], OptiGraph, Elixir і Falcon. Всі перераховані DSL є відносно новими академічними розробками. Відзначимо також, що крім DSL для аналізу статичних графів існують проблемно-орієнтовані мови для побудови алгоритмів обходів графа. Прикладами таких мов є SPARQL [8], Cypher [9], Gremlin [10] та ін. Розгляд подібних DSL виходить за рамки даної статті.

Мета статті: опис та порівняльний аналіз розглянутих DSL (Green-Marl, OptiGraph, Elixir і Falcon), а також наведення міркувань щодо перспективності застосування мов і їх компіляторів для розробки інструментального середовища аналізу великих статичних графів.

Green-Marl

Green-Marl [11, 12] – проблемно-орієнтована мова розробки паралельних програм аналізу графів, розроблена в науково-дослідній лабораторії PPL в Стенфордському університеті. Мова Green-Marl призначена для розробки паралельних алгоритмів аналізу статичних графів. Мова Green-Marl є зовнішньою DSL-мовою з власним компілятором. Компілятор підтримує трансляцію програм на Green-Marl в наступні паралельні моделі програмування: OpenMP - для багатоядерних систем зі спільною пам'яттю, і Pregel - для розподілених систем (в якості реалізації Pregel підтримуються GPS і Giraph). Компілятор Green-Marl продовжує розвиватися в одній з дослідницьких лабораторій Oracle в рамках проекту PGX.D,

але вже як комерційний проект. В Green-Marl введені спеціальні типи для опису графа (тип Graph), вершини (Node), ребра (Edge), а також для опису атрибутів вершин (N_P <тип атрибута>) і ребер (N_E <тип атрибута>). Можна вказувати сусідів заданих вершин (Nbrs). Крім звичайних операторів, таких як While, Do-While, If, If-Else визначають послідовне виконання програм-ми, в Green-Marl підтримуються оператори для опису паралельних обчислень. Для опису циклів, ітерації яких можуть бути виконані паралельно і незалежно один від одного, в Green-Marl використовується спеціальна конструкція:

Foreach (Iterator: Set) (cond) {...} ,

де Set – ітераційна множина(наприклад, множина вершин графа, множина сусідів заданої вершини, колекції - впорядковані і невпорядковані підмножини вершин), it - ітератор. Додатково може бути визначена умова, яка задає які з ітерацій повинні бути виконані, тобто можна фільтрувати ітерації.

Семантика Foreach передбачає, що для кожного елемента множини Set будуть виконані операції, визначені в тілі циклу, при цьому порядок виконання ітерацій циклу не заданий. Допускається використання вкладених Foreach-циклів, при цьому операції fork / join для різних витків циклу виконуються незалежно. Приклад вкладеного циклу:

```
Foreach (I: G.Nodes)
    Foreach (J: i.nbrs)
        Foreach (K: j.nbrs) {...} .
```

Також в Green-Marl підтримується цикл For, який відрізняється від Foreach тим, що виконання циклу For з точки зору зміни пам'яті завжди однозначно визначено й існує еквівалентне послідовне виконання For, яке дає такий самий результат.

З точки зору консистентності пам'яті в Green-Marl виділяють послідовну і паралельну моделі. Послідовна модель відповідає послідовним ділянкам коду (наприклад, Do-While), при цьому результат виконання інструкції, що модифікує якусь змінну (або елемент пам'яті), буде доступний для наступних інструкцій. У паралельній моделі консистентності пам'яті порядок виконання записів і видимість результату гарантується тільки в середині паралельного фрагмента для кожного процесу окремо, відповідно, загальний порядок виконання звернень до пам'яті для всіх процесів не визначений. Проте гарантується, що на момент завершення паралельної ділянки коду всі записи будуть виконані. В Green-Marl введена підтримка редукцій, для цього в тілі циклу Foreach використовуються спеціальні оператори: + =, * =, max =, min =, && =, || =, що позначають додавання, множення, максимум, мінімум, логічне І і логічне АБО, відповідно. Результат редукції може присвоюватися як скалярним змінним, так і атрибутам вершин або ребер. Найпростіший приклад редукції в Green-Marl:

```
int sum = 0;
Foreach (I: G.Nodes) {
sum + = ix;
}
```

Крім того, в Green-Marl підтримуються вбудовані ітератори, що реалізують паралельні обходи DFS і BFS, що сильно спрощує розробку алгоритмів, так як дані операції (обхід в ширину і обхід вглибину), часто зустрічаються в різних алгоритмах. Синтаксично обхід в ширину виглядає таким чином:

```
inBFS (Iterator: G.Nodes [from |;] root)
((Filter_expr)) ([navigator_expr]) {statement_block1}
(inReverse ((filter2_expr)) {statement_block2})
```

Для виконання обходу задається коренева вершина (root), можуть бути задані фільтр для відкидання непотрібних вершин (filter_expr) і навігатор для задання точки зупину обходу (navigator_expr); якщо навігатор не заданий, то обхід завершується тільки після відвідування всіх досяжних вершин. При виконанні обходу для кожної аналізованої вершини (якщо значення фільтра для даної вершини істинно) виконується заданий блок інструкцій (statement_block1). Також можна описати дії, які будуть виконані при зворотньому обході (statement_block2). Обхід вглибину (DFS) визначається аналогічним чином.

Подання графа (тобто те, яким чином граф буде зберігатися, а також як будуть зберігатися атрибути вершин і ребер) визначається компілятором мови і конкретною платформою, в яку здійснюється трансляція. При трансляції про- грам в модель Pregel граф представляється стандартним (і єдиним) способом у вигляді вершин, які мають атрибутами, станом (активне і неактивне) і методом compute, що описує програму вершини. На рис. 1 показано процес роботи компілятора Green-Marl мови.

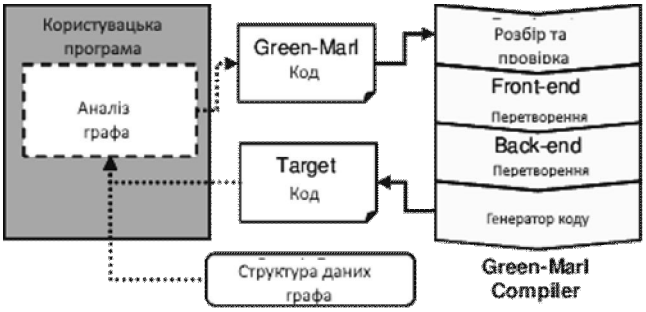

Рис. 1. Процес роботи Green-Marl компілятора

OptiGraph

Мова OptiGraph - DSL для аналізу статичних графів, оснований на специфікації Green-Marl. Мова OptiGraph, також як і Green-Marl, розроблена в лабораторії PPL в Стенфордському університеті. Концептуальних нововведень з точки зору мови в OptiGraph немає. На відміну від Green-Marl, OptiGraph - вбудований DSL, в якості базової мови використовується функціональна мова Scala. OptiGraph розроблений з використанням систем Delite [20] і Forge [21], призначених для розробки проблемно-орієнтованих мов, також розробляються в Стендфордському університеті. Система розробки високоефективних проблемно-орієнтованих мов Delite включає генератори коду для багатоядерних систем із загальною памятю (OpenMP), графічних прискорювачів (OpenCL, CUDA). У процесі розробки знаходяться генератори коду для обчислювальних кластерів, а також для ПЛІС (Verilog).

Elixir

Іншим прикладом проблемно-орієнтованої мови, призначеної для розробки і реалізації паралельних алгоритмів аналізу статичних графів, є Elixir [22]. Мова Elixir розроблена в Техаському університеті в Остіні. Elixir використовує як декларативні, так і імперативні конструкції для визначення обчислень над графом. У Elixir не підтримується структурна трансформація графа (т. Додавання і видалення вершин і дуг). З точки зору реалізації мова Elixir є зовнішньою DSL, тобто для Elixir розроблена своя атрибутна граматика і транслятор, що перетворює програму на Elixir в паралельний код на C++ з викликами системи підтримки паралельного виконання програм Galoios [23]. Основною особливістю Elixir є поділ операцій над графом (в термінах програмної моделі Elixir - операторів) і планування виконання операцій, тобто визначення порядку виконання операторів (в термінах Elixir - планувальників). Для завдання способу застосування операторів до графу використовуються такі вирази: foreach op, for i = low..high op, iterate op, де op - це оператор. Вираз foreach op одноразово застосовує оператор до всіх співставлених підграфів, for i = low..high op застосовує оператор задану кількість разів, iterate op застосовує оператор до тих пір поки є хоча б одне коректне зіставлення. Для визначення порядку виконання операцій вибірки підграфів, використовуються планувальники. У Elixir підтримуються статичні і динамічні політики планування.

Таким чином, Elixir є декларативною мовою програмування (з елементами імперативного програмування, наприклад, цикли for, foreach), заснованою на визначенні операторів над графом і специфікації порядку застосування операторів.

Falcon

Ще одним прикладом проблемно-орієнтованої мови для обробки статичних графів є Falcon. Falcon розробляється в Inidian Institute of Science. Falcon (також як Green-Marl і Elixir) відноситься до класу зовнішніх DSL, але є розширенням C (стандарт C99). В Falcon реалізована підтримка як багатоядерних систем зі спільною пам'яттю, так і графічних прискорювачів. Крім того, підтримуються конфігурації з декількох графічних процесорів, а також гібридна схема обчислень, тобто з одночасним використанням і CPU, і GPU. Falcon є мовою зі строгою типізацією. В доповнення до стандартних типів C Falcon підтримуються такі типи: Graph - граф, Point - вершина, Edge - ребро, Set - множина вершин (визначається статично), Collection - множина вершин (визначається динамічно). Для визначення паралельних ділянок коду, а також для синхронізації обчислень в Falcon використовуються спеціальні оператори.

Висновки

В статті розглянуті 4 проблемно-орієнтованих мови програмування, призначених для розробки і реалізації алгоритмів аналізу статичних графів. Особливості розглянутих DSL представлені в табл. 1.

Таблиця 1 – **Порівняння DSL для аналізу статичних графів**

	Green-Marl	**OptiGraph**	**Elixir**	**Falcon**
Тип DSL	Зовнішній	Вбудований (в Scala)	Зовнішній	Зовнішній
Тип мови	Імперативний, процедурний	Імперативний, процедурний	Декларативний з елементами імпер.	Імперативний, процедурний
Підтримка структурних трансф. графа	-	-	-	+
Підтримка багатоядерних систем з загальною пам'яттю	+ C++/OpenMP	+	+ Galoios	+ C++/OpenMP, Galoios
Підтримка графічних прискорювачів	-	+	-	+
Підтримка обчисл. кластерів (HPC)	-	-	-	-
Підтримка обчислювальних кластерів(BigData)	+	-	-	-
Відкритий вихідний код	+	+	-	-

Всі розглянуті мови є академічними розробками, спрямованими на дослідження підходів з створення проблемно-орієнтованих мов для паралельного аналізу графів. Концептуально, Green-Marl, OptiGraph і Falcon представляють один підхід до побудови мови, заснований на паралельних конструкціях циклів foreach і системі спеціальних типів і ітераторів, що відображають специфіку графових алгоритмів. OptiGraph відрізняється від Green-Marl і Falcon методом реалізації, будучи вбудованим DSL на базі функціональної мови Scala. Крім того, OptiGraph є представником сімейства генерованих DSL для різних областей додатків за допомогою інструментів Forge і Delite. У Elixir використовується зовсім інший підхід до побудови паралельних програм, а саме декларативний з виділенням операторів і специфікації їх застосування. Єдиним з розглянутих мов, які підтримують масово-паралельні обчислювальні комплекси є Green-Marl. Однак ця підтримка реалізована на базі програмних платформ для роботи з великими даними (Giraph і GPS використовують платформу Hadoop для завантаження і вивантаження графів, розроблені на Java), що спочатку є рішенням, орієнтованим на відмовостійкість і масштабування додатків.

Як показав аналіз, підтримки високо-продуктивних кластерів (суперкомп'ютерів) в проблемно-орієнтованих мовах програмування для обробки великих графів на даний момент немає. Беручи до уваги тенденцію на розширення спектра прикладних задач, які виконуються на суперкомп'ютерах, а саме

використання суперкомп'ютерів для рішення аналітичних задач (розуміння даних, машинне навчання, штучний інтелект), стає очевидною потреба в реалізації подібних мовних засобів з нуля або на базі існуючих з орієнтацією на обчислювальні комплекси класу HPC.

СПИСОК ЛІТЕРАТУРИ

1. Reed DA, Dongarra J. Exascale computing and big data // Communi-cations of the ACM. - 2015. - Т. 58. - №. 7. - С. 56-68.
2. Malewicz G. et al. Pregel: a system for large-scale graph processing // Proceedings of the 2010 ACM SIGMOD International Conference on Man-agement of data. - ACM, 2010. - С. 135-146.
3. Low Y. et al. Distributed GraphLab: a framework for machine learning and data mining in the cloud // Proceedings of the VLDB Endowment. - 2012. - Т. 5. - №. 8. - С. 716-727.
4. Gonzalez JE et al. Graphx: Graph processing in a distributed dataflow framework // 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14). - 2014. - С. 599-613.
5. Shvachko K. et al. The hadoop distributed file system // 2010 IEEE 26th symposium on mass storage systems and technologies (MSST). - IEEE 2010. - С. 1-10.
6. Avery C. Giraph: Large-scale graph processing infrastructure on hadoop Proceedings of the Hadoop Summit. Santa Clara. - 2011. - Т. 11.
7. Zaharia M. et al. Spark: cluster computing with working sets // Hot-Cloud. - 2010. - Т. 10. - С. 10-10.
8. Quilitz B., Leser U. Querying distributed RDF data sources with SPARQL European Semantic Web Conference. - Springer Berlin Heidelberg, 2008. - С. 524-538.
9. Webber J. A programmatic introduction to Neo4j // Proceedings of the 3rd annual conference on Systems, programming, and applications: soft-ware for humanity. - ACM, 2012. - С. 217-218.
10. Rodriguez MA The Gremlin graph traversal machine and language (invited talk) // Proceedings of the 15th Symposium on Database Program-ming Languages. - ACM, 2015. - С. 1-10.
11. Green-Marl Specification 0.7.1 https://docs.oracle.com/cd /E56133_01/1.2.0/ Green_Marl_Language_Specification.pdf
12. Hong S. et al. Green-Marl: a DSL for easy and efficient graph analysis // ACM SIGARCH Computer Architecture News. - ACM, 2012. - Т. 40. - №. 1. - 349-362.

Рецензент: д-р техн. наук, проф. В. В. Вишневський,
Державний університет телекомунікацій, Київ
Received (Надійшла) 22.10.2018
Accepted for publication (Прийнята до друку) 05.12.2018

Обзор проблемно-ориентированных языков программирования для параллельного анализа статических графов

П. М. Шутка, А. М. Филоненко

Предметом изучения в статье являются проблемно-ориентированные языки программирования для параллельного анализа статических графов. **Целью** данной статьи является обзор подходов к реализации проблемно-ориентированных языков программирования на примере Green-Marl, OptiGraph, Elixir и Falcon, предназначенных для анализа статических графов. **Задача**: показать эффективность использования предметно-ориентированных языков программирования в анализе статических графов, которыми могут оперировать не только специалисты в области программирования, но и специалисты разработки математических моделей и алгоритмов анализа данных, в частности с применением теории графов; рассмотреть существующие DSL для анализа статических графов с применением параллельных и распределенных вычислений; отметить существующие предметно-ориентированные языки для построения алгоритмов обходов графа; сравнить DSL с точки зрения выразительности параллелизма и применимости для генерации высокоэффективных параллельных программ для суперкомпьютеров и кластерных систем в виде сводной таблицы с основными свойствами как и их компиляторов. Используемым **методом** является: проведение сравнительного анализа предметно-ориентированных языков программирования. Получены следующие **результаты**: выявлены уровень эффективности использования предметно-ориентированных языков программирования в анализе статических графов; рассмотрены существующие DSL; проведен сравнительный анализ DSL. **Выводы**. В статье были рассмотрены четыре проблемно-ориентированных языка программирования, предназначенных для разработки и реализации алгоритмов анализа статических графов.

Ключевые слова: DSL, предметно-ориентированные, анализ статических графов, параллелизм, генерация, таблица, математические модели, обнаружения.

Review of problem-oriented programming languages for partial analysis of static graphics

P. Shutka, A. Filonenko

The subject of study in the article are problem-oriented programming languages for parallel analysis of static graphs. **The purpose** of this article is to review the approaches to the implementation of problem-oriented programming languages on the example of Green-Marl, OptiGraph, Elixir and Falcon, intended for the analysis of static graphs. **Task:** to show the effectiveness of using domain-specific programming languages in the analysis of static graphs that can be operated not only by specialists in the field of programming, but also by specialists in the development of mathematical models and data analysis algorithms, in particular, using graph theory; consider existing DSL for analyzing static graphs using parallel and distributed computing; compare existing domain-specific languages for building graph traversal algorithms; compare DSL in terms of expressiveness of parallelism and applicability for generating high-performance parallel programs for supercomputers and cluster systems in the form of a pivot table with basic properties as their compilers. **The method used is**: conducting a comparative analysis of domain-specific programming languages. **The following results** were obtained: the level of efficiency of using domain-specific programming languages in the analysis of static graphs was revealed; reviewed existing DSL; A comparative analysis of DSL. **Findings**. The article reviewed four problem-oriented programming languages designed to develop and implement algorithms for analyzing static graphs.

Keywords: DSL, domain-specific, analysis of static graphs, parallelism, generation, table, mathematical models, detection.