2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

doi 10.26089/NumMet.v24r433 УДК 519.68

Исследование производительности архитектурно-независимого фреймворка VGL для эффективной реализации графовых алгоритмов

Д. И. Личманов
Московский государственный университет имени М. В. Ломоносова,
Научно-исследовательский вычислительный центр, Москва, Российская Федерация
ORCID: 0000-0001-5401-9522, e-mail: dimlichmanov@gmail.com

И. В. Афанасьев
Московский государственный университет имени М. В. Ломоносова,
Научно-исследовательский вычислительный центр, Москва, Российская Федерация
ORCID: 0000-0002-0202-1548, e-mail: afanasiev_ilya@icloud.com

В. В. Воеводин
Московский государственный университет имени М. В. Ломоносова,
Научно-исследовательский вычислительный центр, Москва, Российская Федерация
ORCID: 0000-0003-1897-1828, e-mail: vadim@parallel.ru

Аннотация: В настоящее время графовые алгоритмы очень часто применяются для решения различных задач моделирования, поскольку многие реальные объекты хорошо моделируются графами (например, дорожная сеть или социальные связи). При этом эффективная реализация таких алгоритмов зачастую очень сложна, что связано, в частности, с нерегулярным доступом к памяти при работе с графами и огромным размером входных графов. Помочь с решением этой проблемы могут графовые фреймворки — программные среды для решения графовых задач. Ранее был разработан архитектурно-независимый фреймворк VGL (Vector Graph Library), позволяющий эффективно реализовывать графовые алгоритмы на различных аппаратных платформах (на многоядерных процессорах с векторными расширениями, графических ускорителях и векторных процессорах NEC). В данной работе было проведено изучение производительности VGL на разных платформах, выполнено сравнение производительности с существующими аналогами, а также предложен и апробирован подход для автоматического выбора формата входного графа на основе методов машинного обучения.

Ключевые слова: графовый фреймворк, графовые алгоритмы, высокопроизводительные вычисления, анализ производительности, векторная обработка, VGL.

Благодарности: Исследование выполнено при финансовой поддержке РФФИ и ЯОПН в рамках научного проекта № 21–57–50002.

Для цитирования: Личманов Д.И., Афанасьев И.В., Воеводин В.В. Исследование производительности архитектурно-независимого фреймворка VGL для эффективной реализации графовых алгоритмов // Вычислительные методы и программирование. 2023. **24**, № 4. 485–499. doi 10.26089/NumMet.v24r433.

© Д. И. Личманов, И. В. Афанасьев, В. В. Воеводин

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

Performance study of the architecture-independent VGL framework for efficient implementation of graph algorithms

Dmitry I. Lichmanov
Lomonosov Moscow State University, Research Computing Center, Moscow, Russia
ORCID: 0000-0001-5401-9522, e-mail: dimlichmanov@gmail.com

Ilya V. Afanasyev
Lomonosov Moscow State University, Research Computing Center, Moscow, Russia
ORCID: 0000-0002-0202-1548, e-mail: afanasiev_ilya@icloud.com

Vadim V. Voevodin
Lomonosov Moscow State University, Research Computing Center, Moscow, Russia
ORCID: 0000-0003-1897-1828, e-mail: vadim@parallel.ru

Abstract: Graph algorithms are currently often used to solve various modeling tasks, since many real-life objects are well modeled by graphs (for example, a road network or social connections). At the same time, the efficient implementation of such algorithms is often very complex, which is due, in particular, to irregular memory access when working with graphs and the huge size of the input graphs. Graph frameworks — software environments for implementing graph algorithms — can help solve this problem. Previously, an architecture-independent VGL (Vector Graph Library) framework was developed that allows for efficient implementation of graph algorithms on various hardware platforms (multi-core processors with vector extensions, graphics accelerators and NEC vector processors). In this work, the performance of VGL was studied on different platforms, its performance was also compared with existing analogues, and an approach for automatic selection of input graph format based on machine learning methods was proposed and evaluated.

Keywords: graph framework, graph algorithms, high-performance computing, performance analysis, vector processing, VGL.

Acknowledgements: The reported study was funded by RFBR and JSPS, project No. 21–57–50002.

For citation: D. I. Lichmanov, I. V. Afanasyev, V. V. Voevodin, “Performance study of the architecture-independent VGL framework for efficient implementation of graph algorithms,” Numerical Methods and Programming. **24** (4), 485–499 (2023). doi 10.26089/NumMet.v24r433.

---

1. Введение. В настоящее время графовые алгоритмы все чаще применяются для решения задач моделирования, поскольку постоянно растет число областей, в которых используются графовые модели. При этом из-за значительной сложности создания эффективных реализаций графовых алгоритмов для современных вычислительных систем возникает необходимость в разработке графовых фреймворков — удобных программных сред для решения графовых задач. Графовые фреймворки позволяют скрывать от пользователя сложные вопросы микроархитектурной оптимизации, выбора форматов, представления графов и вспомогательных данных.

При разработке высокопроизводительного графового фреймворка сложной и важной задачей является обеспечение его архитектурной независимости — возможности эффективного исполнения графовых алгоритмов на различных аппаратных платформах без сложной перенастройки, что существенно повышает его применимость и облегчает использование сторонними пользователями. В случае графовых алгоритмов одним из главных аспектов, который необходимо учитывать для обеспечения их эффективной реализации, является оптимизация работы с памятью. Растущий размер современных графовых моделей обязывает разработчиков прибегать к адаптации приложений под возможности современных вычислительных систем с быстрой памятью, таких как многоядерные системы, векторные и графические ускорители. При этом, несмотря на то что архитектурные особенности перечисленных платформ в значительной степени отличаются, многие аспекты создания эффективных версий графовых алгоритмов

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

являются общими и, следовательно, могут использовать схожие интерфейсы, обеспечивая переносимость разрабатываемого фреймворка.

На основе этой идеи ранее был разработан архитектурно-независимый фреймворк VGL [1, 2], который позволяет эффективно реализовывать графовые алгоритмы на многоядерных процессорах, графических ускорителях и векторных процессорах NEC [3]. Реализация данного фреймворка выложена в открытом доступе [1].

В данной работе проведено детальное сравнение производительности фреймворка VGL на многоядерных процессорах и графических ускорителях с существующими высокопроизводительными аналогами, которое показывает, что VGL во многих случаях не хуже или же превосходит их по скорости. Также разработан и описан подход с использованием методов машинного обучения, позволяющий автоматически выбирать наиболее подходящий формат хранения, исходя из характеристик графа, целевой архитектуры и графового алгоритма. Это позволяет улучшить работу фреймворка, поскольку выбор формата может кардинально влиять на производительность программной реализации графового алгоритма.

Далее статья устроена следующим образом. В разделе 2 рассмотрен разработанный ранее фреймворк VGL, его возможности и используемые алгоритмические абстракции. В разделе 3 приведены результаты сравнения производительности реализаций четырех графовых алгоритмов, выполненных с помощью VGL и трех других фреймворков — Gunrock [4, 5], Ligra [6, 7] и GAP Benchmark Suite (GAPBS) [8]. В разделе 4 описан предложенный автоматизированный метод выбора оптимального формата хранения данных для каждого конкретного входного графа по его базовым характеристикам, а также приведены результаты его работы на реальных больших графах.

2. Описание фреймворка VGL. Представленные далее выводы и результаты посвящены работе с фреймворком VGL, разработанным ранее авторами настоящей статьи. Данный фреймворк нацелен на следующие современные вычислительные системы: векторные процессоры NEC SX-Aurora TSUBASA, процессоры x86 с векторными расширениями, процессоры ARM, а также графические ускорители NVIDIA.

2.1. Архитектура и возможности VGL. Отправной точкой для создания фреймворка, позволяющего унифицировать подходы к разработке графовых алгоритмов, является идея выявления изоморфизма информационных графов различных алгоритмов. Несмотря на то что группы из элементарных операций могут значительно отличаться для различных групп графовых алгоритмов, на уровне макрографа можно выделить типовые алгоритмические структуры, общие для всех реализованных далее во фреймворке алгоритмов [9]. Предложенные в VGL алгоритмические абстракции и абстракции данных позволяют оптимально подойти к эффективному описанию широкого класса итеративных графовых алгоритмов. Итеративные графовые алгоритмы устроены следующим образом: на каждой итерации алгоритма производится обработка (выполнение некоторых вычислительных операций) некоторого, зачастую небольшого, подмножества вершин и ребер графа. Так, например, в алгоритме обхода графа в ширину на каждой итерации проводится обработка только тех ребер, которые являются инцидентными хотя бы одной вершине из текущего набора вершин.

Основная ценность фреймворка VGL заключается в том, что для оптимальной реализации графового алгоритма нет необходимости в построении уникальных структур данных и операций, присущих конкретному алгоритму. За счет реализации универсальных алгоритмических структур от пользователя скрывается большинство архитектурных и программных особенностей их реализации. При этом у пользователя есть возможность с помощью простых настроек выбрать, где это возможно, интересующий вариант реализации структур данных. Так, например, во фреймворке VGL реализовано несколько форматов хранения графа, переключение между которыми может быть сделано путем выставления нужных флагов. Вопрос об автоматизации выбора нужного формата представлен далее в разделе 4.

Графовые алгоритмы с итеративной схемой вычислений во фреймворке VGL реализуются через всего лишь четыре абстракции вычислений и две абстракции данных, речь о которых пойдет далее. Это значит, что быстродействие и эффективность каждого из реализованных в VGL алгоритмов существенно опирается на эффективность реализаций этих абстракций. Описанная особенность позволяет транзитивно применять архитектурные оптимизации реализаций абстракций ко всем алгоритмам, использующим какую-либо конкретную абстракцию.

Двумя центральными абстракциями данных фреймворка VGL являются подмножество вершин (для краткости называемое далее фронтом) и граф. Пользователь VGL формирует различные фронты графа на основании некоторых определяемых им критериев, после чего применяет к вершинам фронта, а также,

при необходимости, к их смежным ребрам различные вычислительные операции. Для работы с абстракциями данных во фреймворк VGL включены четыре основные вычислительные абстракции: advance, generate_new_frontier, compute и reduce.

- Advance. Вычислительная абстракция advance является основным способом обхода графа во фреймворке VGL. Реализация advance принимает на вход граф и заданный фронт, а также несколько определяемых пользователем функций-обработчиков. Для каждой из вершин фронта в начале выполняется предварительная операция (preprocess), затем для каждого исходящего из данной вершины ребра выполняется основная операция, причем все смежные ребра заданной вершины обрабатываются параллельно. После этого для каждой из вершин выполняется завершающая операция (postprocess). Обработка вершин векторными инструкциями производится двумя принципиально отличными способами: вершины с малой степенью (числом инцидентных вершине ребер) обрабатываются векторными инструкциями коллективно, в то время как каждая из вершин с большой степенью обрабатывается индивидуально, на них приходится по одной или несколько векторных инструкций.

  Также из-за особенностей работы с направленными графами во фреймворке VGL предусмотрены функции-обертки для абстракции advance: scatter вызывается при обработке ребер графа, исходящих из вершины, в то время как входящие в вершину ребра обрабатываются с помощью функции gather.
- Generate_new_frontier. Эта вычислительная абстракция осуществляет генерацию фронта вершин в три этапа. На первом этапе для каждой из вершин заполняется массив флагов принадлежности вершин к фронту на основании некоторого условия, причем условие проверяется для всех вершин графа. На втором этапе производится оценка числа вершин в создаваемом фронте, после чего на третьем этапе принимается решение о принадлежности фронта к различным типам на основании критериев, заданных во фреймворке или пользователем.

  Во избежание сильной разреженности итоговых фронтов в фреймворке VGL реализована альтернативная абстракция advance; она также может генерировать новый фронт, в который могут попасть только вершины, являющиеся смежными для вершин из фронта, подаваемых на вход альтернативной операции advance. Это значительно уменьшает объем необходимых вычислений для сильно разреженных случаев.
- Compute. Абстракция compute применяет заданную пользователем операцию к каждой из вершин, принадлежащих заданному фронту. Реализация абстракции compute выполнена прямолинейным образом, поскольку применяемые для каждой вершины алгоритмические шаблоны не имеют информационных зависимостей друг от друга, следовательно, шаблоны можно обрабатывать, используя полноценную векторную и параллельную обработку данных. Абстракция compute для большинства алгоритмов не выполняет косвенных обращений к памяти.
- Reduce. Реализация абстракции reduce аналогична реализации абстракции compute, а выделение reduce в отдельную абстракцию обусловлено повсеместно используемой в алгоритмах операции редуцирования вычисленных значений после применения алгоритмических шаблонов. Тип редукции при этом разнообразен и включает возможности реализации операций сложения, умножения, поиска минимума, максимума, подсчета суммы и др.

На основе представленных абстракций в VGL на настоящий момент реализовано 19 широко распространенных графовых алгоритмов [10].

2.2. Создание единого фреймворка для различных платформ. Для трех рассматриваемых архитектур (векторные архитектуры, многоядерные центральные процессоры x86 и ARM, а также графические ускорители) реализованы отдельные классы, полностью соответствующие вышеприведенному набору абстракций вычислений и данных. Эти классы являются производными от базовых классов, в которых определен единый для всех архитектур интерфейс для абстракций вычислений и данных, описанных выше. Общая схема разработанного программного решения приведена на рис. 1.

В производных классах во фреймворке VGL реализованы соответствующие единому архитектурно-независимому интерфейсу оптимизированные реализации абстракций и структур данных, отличающиеся для различных архитектур. Так, в случае реализации методов класса для выполнения на NVIDIA GPU используется модель CUDA [11], позволяющая при аккуратной работе создавать вычислительные ядра с более гибкими настройками, нежели при использовании, например, библиотеки Thrust [12] или средств для работы с GPU, внедренных в новые стандарты OpenMP. Реализации методов класса при целевой

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

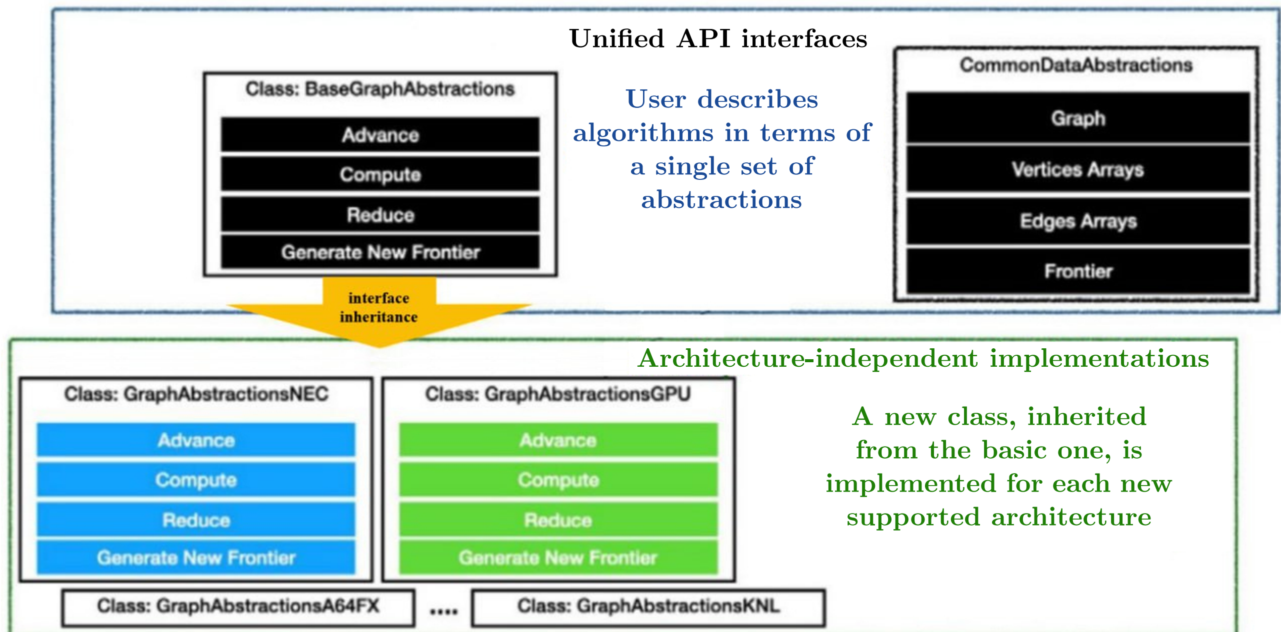

Рис. 1. Иерархия классов во фреймворке VGL

Fig. 1. Class hierarchy in VGL framework

архитектуре NEC SX-Aurora TSUBASA компилируются со специальными директивами, схожими с директивами OpenMP, использующимися при реализации методов классов для многоядерных архитектур.

С пользовательской точки зрения сценарий работы с разработанным фреймворком VGL на различных архитектурах очень прост. Для запуска расчета задачи на выбранной архитектуре достаточно лишь указать соответствующий флаг и код выбранной архитектуры — для GPU это *cu/gpu*, для NEC SX-Aurora TSUBASA это *sx/aurora*, для многоядерных систем это *mc/multicore*. По умолчанию запуск задачи проводится на многоядерной платформе ввиду ее всеобщей доступности. Для работы на кластерных системах с гетерогенным набором узлов, таких как Ломоносов-2, также реализована обертка, позволяющая подтягивать окружение, необходимое для запуска задания посредством менеджера ресурсов Slurm.

В случае отсутствия соответствующего выбранной архитектуре исполняемого файла автоматически запускается сборка реализации соответствующего графового алгоритма. Также у пользователя есть возможность указать желаемый формат хранения входных графов.

3. Производительность VGL на различных платформах и сравнение с существующими аналогами. Использование классических методов оценки производительности на основе метрики Floating Point Operations Per Second (FLOPS) для графовых алгоритмов не всегда информативно, так как графовые алгоритмы слабо задействуют вещественную арифметику. Показатель эффективной пропускной способности памяти, ввиду четкой классификации графовых алгоритмов как memory-bound задач, дает больше информации о производительности конкретного графового алгоритма. Однако наиболее нацеленной на графовые приложения является специальная метрика Traversed Edges Per Second (TEPS), определяемая по формуле $edges_count/time$, где $edges_count$ — общее количество ребер в обрабатываемом графе, а $time$ — полное время работы исследуемого графового алгоритма. Эта метрика используется при формировании общемирового рейтинга Graph500 [13], в котором представлены суперкомпьютерные платформы, показавшие наибольшую производительность при выполнении алгоритма поиска в ширину и алгоритма нахождения кратчайших путей от вершины. При помощи метрики TEPS удобно сравнивать производительность различных реализаций графовых задач, в том числе между различными вычислительными архитектурами. В данной статье при сравнении работы графовых алгоритмов на различных архитектурах и на различных платформах авторами будет использоваться именно метрика TEPS. Стоит также отметить, что все приведенные показатели будут представлены в масштабе миллиона обработанных ребер в секунду и иметь обозначение MTEPS.

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

Нами были выбраны задачи, алгоритмы решения которых широко распространены и используют различные сочетания выделенных выше алгоритмических абстракций. В результате в настоящей работе производительность фреймворка VGL как на разных платформах, так и в сравнении с другими фреймворками демонстрируется на следующих фундаментальных графовых задачах:

- поиск в ширину (Breadth First Search — BFS), алгоритм top-down;
- поиск всех пар кратчайших путей в графе от заданной вершины (Single Source Shortest Paths — SSSP), параллельная модификация алгоритма Дейкстры [14];
- ранжирование вершин в графе, алгоритм Page Rank (PR) [14];
- поиск связных компонент в графе (Connected Components — CC), алгоритм Шилоаха–Вишкина [15].

Так как выше пояснено, какие алгоритмы взяты в качестве опорных для решения той или иной задачи, будем далее взаимозаменять понятия задачи и алгоритма.

Несмотря на то что во фреймворке VGL реализованы генераторы различных синтетических графов, такие как равномерно-случайный граф (Random Uniform) и RMAT (Recursive Matrix) [16], наибольший интерес представляет рассмотрение производительности фреймворка на входных графах реального мира. Коллекция подобных графов доступна в рамках проекта Konect [17] через web-сайт. Графы реального мира на web-сайте Konect разложены по категориям, таким как графы цитирования, графы соавторства, инфраструктурные графы, графы, полученные из систем рейтингов.

Во фреймворке VGL реализованы удобные пользовательские сценарии работы с графами из коллекции Konect. Во-первых, с помощью реализованных скриптов можно быстро получить названия всех графов из коллекции, удовлетворяющих заданному условию. Во-вторых, по заданным названиям графов можно автоматически произвести их загрузку с web-сайта и конвертацию в используемый для алгоритмов VGL формат входного файла. Более того, уже сформировано много тестовых множеств, включающих в себя сгруппированные как по размеру, так и по категориям наборы графов.

В табл. 1 указаны графы, выбранные для проведения сравнительных экспериментов. По аналогии с представлением групп графов в настройках VGL, графы сгруппированы по категориям и размерам.

3.1. Сравнение производительности VGL на различных платформах. Кросс-платформенность разработанного фреймворка VGL позволяет с легкостью проводить эксперименты с одним набором тестовых графов на различных архитектурах. В данном разделе приведено сравнение производительности реализаций графовых алгоритмов для многоядерных центральных процессоров различных архитектур, графических ускорителей NVIDIA двух архитектур (V100 и P100) и архитектуры с векторными ускорителями NEC SX-Aurora TSUBASA.

Для проведения экспериментов преимущественно использовался кластер Ломоносов-2. В частности, для проведения расчетов на графических ускорителях NVIDIA использовались узлы из разделов volta1 (для запусков на NVIDIA V100 и на Intel Xeon Gold 6126), pascal (NVIDIA P100), nec (NEC SX-Aurora TSUBASA). Для проведения вычислений на процессорах с архитектурой ARM использовался отдельный сервер с 64-ядерными процессорами HiSilicon Kunpeng 920.

Таблица 1. Информация о графах, на которых проводились эксперименты по сравнению производительности. Для каждого графа в скобках приводится два числа (в миллионах): первое показывает количество вершин, второе — количество ребер

Table 1. Characteristics of graphs which participate in performance comparison evaluations. The first number means the number of vertices in the graph, the second — the number of edges (in the scale of millions)

	Социальные Social	Инфраструктурные Infrastructure	Сетевые Web	Рейтинговые Rating
Малые Small	Youtube friendships (1.13 M, 3 M)	Texas (1.38 M, 1.9 M)	Stanford (0.28 M, 2.3 M)	Amazon ratings (3.4 M, 5.8 M)
Средние Medium	LiveJournal Links (5.2 M, 49 M)	Western USA (6.2 M, 15 M)	Zhishi (7.83 M, 66 M)	Epinions (0.87 M, 13.6 M)
Большие Big	—	Central USA (14 M, 34 M)	—	—
Огромные Large	—	Full USA (24 M, 57.7 M)	UK domain (18.5 M, 262 M)	Yahoo songs (1.62 M, 257 M)

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

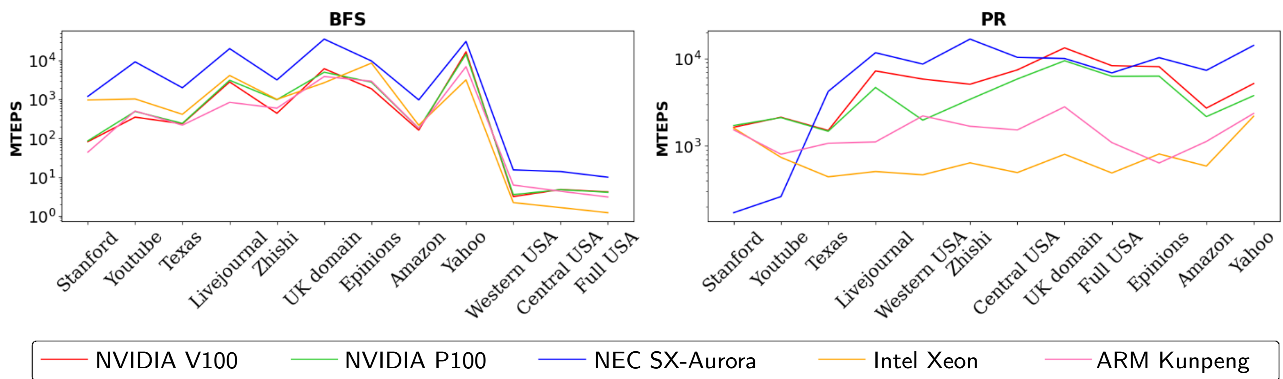

Рис. 2. Сравнительная производительность алгоритмов Breadth-First-Search и PageRank на целевых вычислительных платформах

Fig. 2. Performance comparison of Breadth-First-Search and PageRank algorithms evaluated on target architectures

Для сборки приложений для многоядерных архитектур (и для сборки хостового кода при реализации приложений для графических ускорителей) на Ломоносов-2 использовался компилятор g++ (GCC) 9.1.0, для Kunpeng 920 — g++ (GCC) 12.0.0. Для сборки приложений для графических ускорителей NVIDIA использовалась модель параллельных вычислений CUDA версии 11.1, с соответствующими средствами компиляции. Компилятор nc++ (NCC) 3.3.0 использовался в случае реализации приложений для векторной архитектуры NEC SX-Aurora TSUBASA.

На рис. 2 представлена производительность в MTEPS (показатели метрики здесь и далее даны в логарифмической шкале) реализаций графовых алгоритмов для задач BFS и PR на всех заявленных архитектурах. Сравнительная производительность на алгоритмах CC и SSSP схожа с BFS. На примере приведенных графиков стоит обратить внимание: реализации на NEC SX-Aurora TSUBASA показывают до 15 раз бóльшую производительность по сравнению с реализациями для других архитектур. Стоит также отметить, что в процессе увеличения размера графов из представленного в табл. 1 набора позиции многоядерных архитектур в рейтинге производительности падают.

Данные показатели производительности обусловлены сильно различными теоретическими значениями пропускной способности памяти для целевых платформ: 110 ГБ/с для Intel Xeon, 190 ГБ/с для Kunpeng 920 против 1.2 ТБ/с для NEC SX-Aurora TSUBASA, 900 ГБ/с для V100 GPU и 720 ГБ/с для P100 GPU.

Тот факт, что производительность реализаций приблизительно пропорциональна значениям теоретических пропускных способностей памяти, подтверждает высказанный ранее тезис о значительном преимуществе использования векторных систем с быстрой памятью для ускорения решения графовых задач [9, 18]. Эта особенность также говорит о том, что фреймворк VGL, изначально реализованный для архитектур с векторными ускорителями, достаточно хорошо оптимизирован и для работы на графических ускорителях и многоядерных процессорах, поскольку показывает схожие с векторными ускорителями показатели эффективности.

3.2. Сравнение производительности VGL с существующими аналогами. Ввиду наличия достаточного числа существующих известных графовых фреймворков, необходимо провести сравнительные эксперименты производительности. С этой целью были выбраны наиболее известные из фреймворков, как для многоядерных архитектур, так и для графических ускорителей. Поскольку VGL является единственным фреймворком, в котором есть реализации для векторных архитектур NEC SX-Aurora, для этой архитектуры проведение сравнений с аналогами не представляется возможным.

Отметим, что помимо рассматриваемых далее существуют и другие фреймворки, такие как Galois [19], GraphIT [20] и реализации стандарта GraphBLAS [21]. Они еще не рассматривались, потому что на момент начала работ фреймворки, с которыми было проведено сравнение, были одними из наиболее производительных и популярных. В дальнейших работах будут проведены сравнения и с упомянутыми более новыми фреймворками.

§ 3.2.1. Сторонние фреймворки для многоядерных архитектур. Ligra — это легковесный фреймворк для обработки графов на системах с общей памятью. Авторы подчеркивают, что он особенно хорошо подходит для реализации алгоритмов параллельного обхода графа, где за одну итерацию обрабатывается только подмножество вершин, что является аналогом фронта в VGL. Авторы также обращают внимание, что фреймворк Ligra был разработан в качестве альтернативы фреймворкам, использующим распределенные вычисления для расчета графовых алгоритмов.

Аналогично VGL, Ligra поддерживает два типа данных, один из которых представляет граф, а другой — подмножества вершин, которые называются vertexSubsets. В интерфейсе также присутствуют удобные функции для обхода произвольного множества вершин или ребер графа. Авторы упоминают, что их реализации обхода подмножества вершин наиболее эффективны в случае небольшого размера vertexSubsets.

В Ligra реализован удобный процесс сборки через Makefile, однако запуски графовых алгоритмов для графов в формате MatrixMarket [22], принятом в Konect, во-первых, недоступны напрямую (без ручного использования программы-преобразователя), а во-вторых, файлы входных графов в производном от MatrixMarket формате Hypergraph принимаются лишь приложениями с префиксом hyper.

Аналогично Ligra, GAPBS предоставляет набор реализаций графовых алгоритмов для их апробации на многоядерных вычислительных платформах, однако авторы данной разработки позиционируют GAPBS не как фреймворк, а как набор монолитных бенчмарков, реализации которых содержат распараллеленные с помощью OpenMP участки кода. Число абстракций, подобно используемым в VGL и Ligra, в GAPBS сведено к минимуму.

GAPBS предоставляет возможность передавать в приложения графы непосредственно в MatrixMarket формате, предобрабатывая их в свой внутренний формат непосредственно перед выполнением алгоритма.

§ 3.2.2. Сторонние фреймворки для графических ускорителей. Gunrock — это фреймворк, использующий технологию CUDA для написания высокопроизводительных графовых примитивов. Фреймворк реализует высокоуровневые массово-синхронные абстракции, ориентированные на операции с фронтами вершин или ребер. Набор абстракций Gunrock наиболее близок к используемым в VGL, в его интерфейсе также присутствуют абстракции advance и compute, чей функционал аналогичен функционалу данных абстракций в VGL. Также реализована абстракция filter, являющаяся аналогом комбинации compute и generate_new_frontier фреймворка VGL. Для реализации многих операций в Gunrock используется библиотека Thrust, в которой реализованы, например, эффективные приемы для операции редукции на GPU.

Приложения фреймворка Gunrock, равно как и GAPBS, принимают на вход графы в формате MatrixMarket.

§ 3.2.3. Сравнительный анализ производительности VGL со сторонними фреймворками. На рис. 3 показано сравнение производительности VGL, Ligra и GAPBS на четырех указанных ранее графовых алгоритмах BFS, PR, SSSP и CC, реализованных для многоядерных процессоров. Сравнение производительности Gunrock и VGL на аналогичных алгоритмах, реализованных на графических ускорителях, представлено на рис. 4.

Как видно из рис. 3, 4, производительность реализаций фреймворка VGL показывает в среднем значительно лучшие значения, нежели реализации с помощью сторонних фреймворков. На всех рассматриваемых алгоритмах производительность фреймворка VGL на многоядерных архитектурах обгоняет производительность фреймворка Ligra. В случае другого конкурента на многоядерных архитектурах, GAPBS, значительное ускорение от использования VGL наблюдается на трех алгоритмах из четырех — BFS, SSSP, PR. В случае графических ускорителей фреймворк VGL также показывает более высокую производительность, чем его аналог Gunrock. На алгоритмах PR и SSSP наблюдается значительное ускорение, в то время как на BFS и CC VGL лишь немного уступает конкуренту.

Одна особенность полученных результатов заставляет обратить внимание на некоторые проблемы в построении логики фреймворка VGL. Она состоит в том, что на дорожных графах (Western, Central и Full USA) реализации VGL стабильно уступают реализациям конкурентов. Именно этим обусловлен, например, проигрыш фреймворку Gunrock при реализации алгоритмов BFS и CC.

Причина заключается в следующем. В VGL по умолчанию используется специальный формат хранения графов VectCSR (именно он использовался для получения результатов на обоих графиках выше), и без знания информации о преимуществах каждого конкретного формата при их реализации на векторных и графических ускорителях пользователь не примет это во внимание. Ввиду отличной производительно-

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

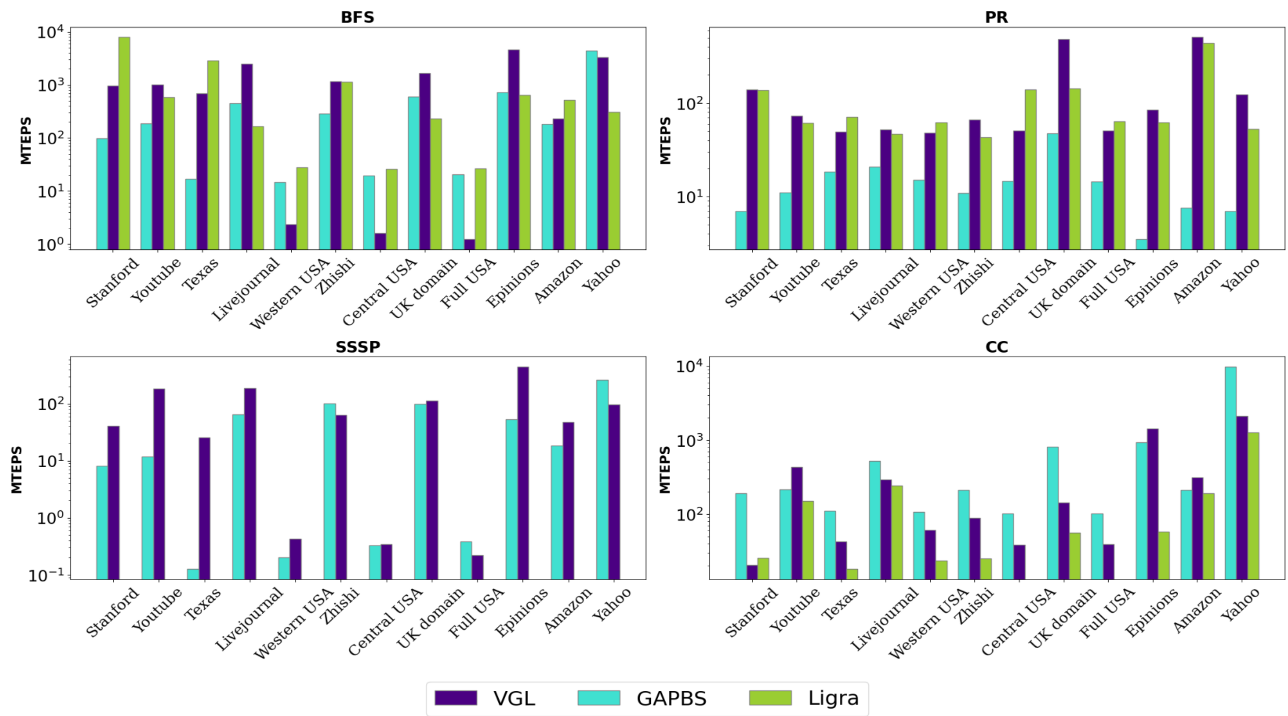

Рис. 3. Сравнение производительности четырех графовых алгоритмов на многоядерных процессорах при использовании фреймворков VGL, GAPBS и Ligra

Fig. 3. Performance comparison of VGL, GAPBS and Ligra CPU implementations evaluated on four graph algorithms

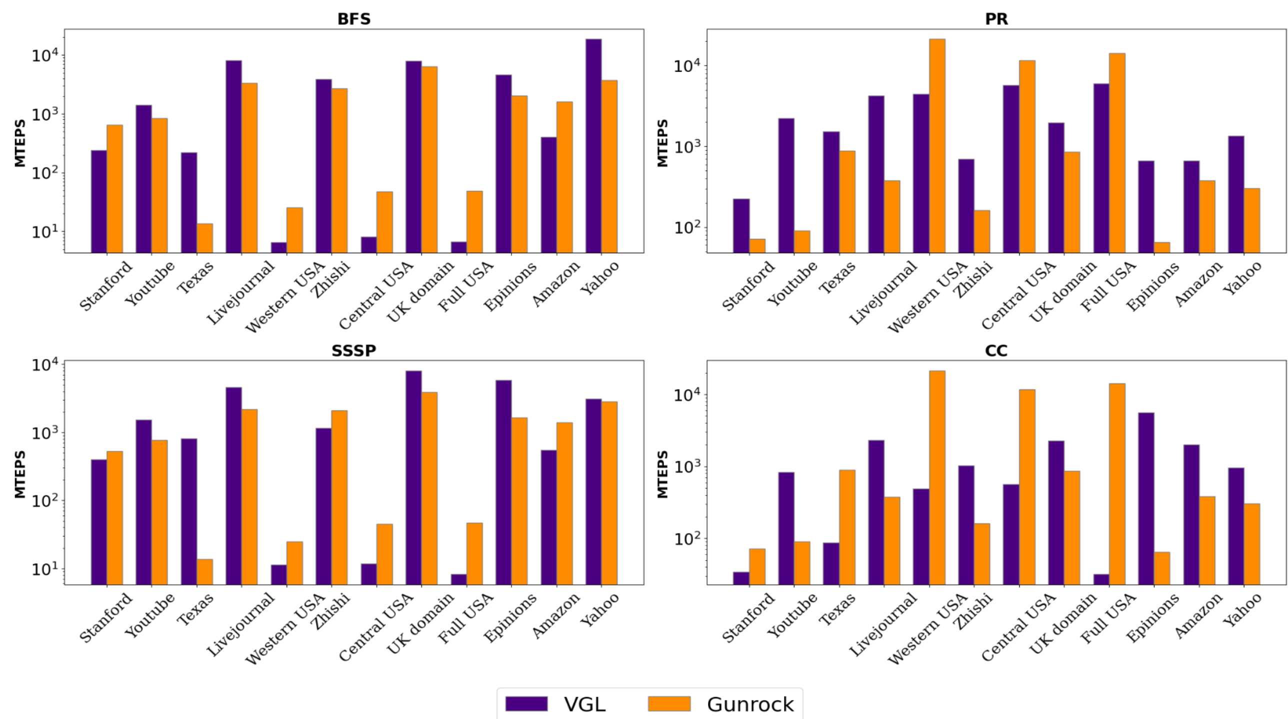

Рис. 4. Сравнение производительности четырех графовых алгоритмов на графических ускорителях при использовании фреймворков VGL и Gunrock

Fig. 4. Performance comparison of VGL and Gunrock GPU implementations evaluated on four graph algorithms

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

сти [10] реализаций алгоритмов, использующих данный формат как для векторных архитектур, так и для графических ускорителей и многоядерных архитектур, формат VectCSR было решено использовать как базовый формат фреймворка из-за его наилучшей эффективности для большинства входных графов.

Основная идея формата VectCSR заключается в кластеризации. При кластеризации в кэш-памяти хранятся только те вершины графа, к которым производятся наиболее частые обращения. Так как наиболее часто запрашиваемые вершины графа имеют самое большое количество смежных ребер (это верно для всех неориентированных и большинства ориентированных графов реального мира), то при сортировке вершин графа по убыванию степени наиболее часто запрашиваемые при косвенных адресациях данные будут находиться в смежных ячейках массива. Однако кластеризация имеет и недостаток: она становится значительно менее эффективной, если в графе отсутствуют вершины с высокой степенью. К такому классу графов и относятся графы, являющиеся моделями дорожных сетей. Соответственно, возникает задача разработать решение, которое помогло бы подобрать подходящий формат хранения для каждого конкретного графа.

4. Автоматизированный выбор форматов графа. Проблемы, поставленные в конце § 3.2.3, определили дальнейшее направление в работе над фреймворком VGL. Было решено провести исследование, которое позволит ответить на вопрос, могут ли модели машинного и/или глубокого обучения помочь по некоторому множеству характеристик входного графа автоматически определять наиболее подходящий для него формат? Далее в работе показываются подходы авторов к решению этой задачи и возможные дальнейшие направления исследований по этой теме. Стоит отметить, что предложенные в данном разделе методы было решено внедрить для реализации на графических ускорителях и многоядерных процессорах, так как именно они являются наиболее актуальными с точки зрения пользователя фреймворка VGL.

4.1. Описание предложенного метода. Сначала необходимо определить, что может являться входными данными для моделей. Такими данными могут служить некоторые характеристики входных графов, которые будут участвовать в обучении. В идеале данные характеристики должны быть вычислены непосредственно перед расчетом графового алгоритма, однако в текущей версии модуля для автоматического выбора формата графа данные характеристики берутся из графа с помощью специального скрипта с web-сайта Konect.

На основе многолетнего опыта работы авторов с графовыми алгоритмами в качестве базовых признаков для обучения классификатора были отобраны следующие характеристики:

- число вершин графа;
- число ребер графа;
- 90-перцентиль распределения диаметра (максимальное расстояние между любыми двумя его вершинами);
- среднее значение степени связности вершины;
- показатель экспоненты аппроксимации распределения вершин графа степенным законом.

Для проведения экспериментов были отобраны графы из коллекции, число ребер которых лежит в диапазоне [0.5 млн, 80 млн], так как именно для таких графов вопрос в выборе оптимального формата стоит максимально остро: разница во времени выполнения малых графов не так заметна в общей картине, а графы, число ребер которых превышает заданную верхнюю границу, оптимальнее запускать именно в формате VectCSR. Стоит отметить, что данная граница является приблизительной и нуждается в дальнейших уточнениях. Всего было отобрано чуть менее 400 графов, из которых в тестовый набор была случайным образом отобрана одна восьмая часть графов — порядка 50. Достаточно небольшое количество графов для проведения обучения было обусловлено числом доступных для использования графов на web-ресурсе Konect.

Для решения задачи автоматизированного выбора подходящего формата графов были рассмотрены различные модели машинного и глубокого обучения.

- K ближайших соседей (KNN). На валидации подбирался параметр K, наиболее оптимальным был K, равный 5.
- Наивный байесовский классификатор.

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

- Полносвязная нейронная сеть с двумя слоями и выходами по числу классов-форматов. Рассмотрение именно полносвязной сети среди доступных моделей глубокого обучения связано с повсеместным использованием именно данной модели для решения подобных задач классификации.
- XGBoost [23] классификатор. Число n_estimators было подобрано на валидации и равно 50.
- Алгоритм случайного леса (Random Forest) из библиотеки Scikit-Learn [24]. Число деревьев для оценки так же подбиралось на валидации, и в итоге выбрано 75.

Важно отметить, что в данном подходе реализация единой модели машинного обучения, общей для всех алгоритмов и для архитектур, не являлась приоритетной задачей. Одной из целей была разработка набора характеристик, не содержащего какие-либо категориальные характеристики алгоритмов или архитектур, а исчерпывающегося лишь характеристиками графов. При этом, как видно из рис. 3, 4, производительность графовых алгоритмов на разных входных графах меняется по разным законам, что также послужило поводом для рассмотрения независимых моделей.

Каждая из данных моделей была обучена с 7-кратной перекрестной проверкой (7-fold cross-validation). Качество предсказаний оценивалось согласно метрике точности (accuracy). Она определяет долю графов, для которых алгоритм машинного обучения в качестве результата выдал формат хранения, являющийся оптимальным для каждого конкретного графа (т.е. позволяет достичь максимальное значение MTEPS).

Принимая во внимание дальнейшие планы по созданию модели машинного обучения, которая являлась бы единой в рамках всего фреймворка и принимала бы целевую архитектуру и требуемый графовый алгоритм в качестве входных характеристик, наибольший интерес при реализации отдельных ML-моделей представляет определение общего для всех алгоритмов и для заданных целевых архитектур алгоритма машинного обучения. В качестве такого алгоритма был взят алгоритм случайного леса, поскольку его точность на подавляющем большинстве алгоритмов превосходила точности остальных алгоритмов. Так, например, для алгоритма BFS и его реализации на GPU точность обученного классификатора, использующего алгоритм случайного леса, составила 0.77 на валидации. Точность модели, использующей нейронные сети, составила 0.73; XGBoost — 0.74; KNN — 0.64; наивный байесовский алгоритм — 0.16. Стоит также отметить, что точность на алгоритме BFS является наименьшей среди тестовых алгоритмов, в то время как для алгоритмов PR и SSSP при использовании алгоритма случайного леса она составила около 0.9.

4.2. Оценка качества работы предложенного метода на реальных данных. Использование обученных классификаторов было внедрено в скрипты запуска VGL. Пользователь VGL с помощью специального флага при запуске может указать, нужно ли использовать обученную модель машинного обучения при предстоящем запуске графового алгоритма на выбранной архитектуре.

Всего было рассмотрено три формата графов — уже упомянутый VectCSR, а также CSR (Compressed-Sparse-Row) и EL (Edges List). VectCSR является вариантом по умолчанию, однако эксперименты показали, что два других формата в некоторых случаях превосходят VectCSR. Остальные форматы, реализованные в VGL, не были взяты в качестве опорных — ввиду слишком малого числа графов, на которых эти форматы показывают лучший результат. При этом их внедрение для проведения экспериментов значительно ухудшало качество работы получаемых моделей.

Таким образом, в экспериментах по применимости ML-модели на практике сравнивались значения метрики MTEPS, полученные от запусков алгоритмов на тестовом наборе графов для четырех вариантов: 1) для всех графов использовался только формат CSR; 2) только VectCSR; 3) только EL; 4) для каждого графа модель машинного обучения предсказывала, какой из этих трех форматов хранения больше всего подходит. В каждом случае нас интересовало ускорение варианта 4 относительно других вариантов, которое вычислялось как $mteps_4/mteps_x$, где $mteps_4$ — производительность в MTEPS варианта 4, а $mteps_x$ — производительность в MTEPS варианта 1, 2 или 3. В табл. 2, 3 приведено среднее геометрическое ускорение варианта 4 с ML-определением формата графа относительно первых трех вариантов с постоянным выбранным форматом.

Эксперименты проводились на суперкомпьютере Ломоносов-2. Результаты для многоядерных процессоров получены на Intel Xeon Gold 6126, для графических ускорителей — на NVIDIA V100.

Сразу стоит отметить, что полученные на GPU результаты явно показывают необходимость отойти от формата VectCSR, используемого по умолчанию для всех алгоритмов, в пользу EL. Поскольку использование ML-модели незначительно замедляет производительность только лишь на одной из восьми реализаций алгоритмов, принимая во внимание полную независимость данных моделей друг от друга, внедрение семи моделей машинного обучения (для алгоритмов BFS, CC, PR на двух архитектурах и

2023, **24** (4), 485–499. doi 10.26089/NumMet.v24r433

Таблица 2. Среднее геометрическое ускорение предложенного варианта на основе методов машинного обучения относительно трех других вариантов, многоядерные процессоры

Table 2. Geometric mean speedup of proposed ML-method in comparison to other three options, multicore

Алгоритм Algorithm	Вариант 1 (CSR) Option 1 (CSR)	Вариант 2 (VectCSR) Option 2 (VectCSR)	Вариант 3 (EL) Option 3 (EL)
BFS	1.05	1.07	6.78
CC	2.4	1.08	2.26
SSSP	2.37	0.95	1.50
PR	2.62	1.34	1.10

Таблица 3. Среднее геометрическое ускорение предложенного варианта на основе методов машинного обучения относительно трех других вариантов, графические ускорители

Table 3. Geometric mean speedup of proposed ML-method in comparison to other three options, GPU

Алгоритм Algorithm	Вариант 1 (CSR) Option 1 (CSR)	Вариант 2 (VectCSR) Option 2 (VectCSR)	Вариант 3 (EL) Option 3 (EL)
BFS	1.34	1.65	1.15
CC	3.00	4.08	1.04
SSSP	8.38	10.45	1.02
PR	9.13	8.18	1.10

SSSP на GPU) более чем оправдано. Данные модели были добавлены в новую версию разрабатываемого фреймворка. Стоит также напомнить, что полученные классификаторы должны быть и будут использованы только для графов, число ребер которых не превышает указанной выше верхней границы в 80 млн.

Эксперименты показали, что предложенный подход с использованием моделей машинного обучения позволяет улучшить среднюю производительность VGL на предложенном наборе алгоритмов. Пользовательская задача, использующая данный фреймворк, при указании всего лишь одного нужного флага может быть ускорена в среднем до 10% (по сравнению с оптимально выбранным статически заданным форматом). Схожесть информационных графов рассмотренных нами алгоритмов и остальных алгоритмов VGL, не представленных в работе, дает основания полагать, что данный ML-метод будет актуален для абсолютного большинства алгоритмов, реализованных в фреймворке VGL.

Дальнейшие работы, посвященные использованию ML-моделей, будут направлены на уточнение характеристик графа, используемых в обучении, и на расширение набора форматов графа, для которых проводятся предсказания. Также планируется сделать единое решение во избежание сложностей со стремительно растущим числом моделей при обучении на каждом алгоритме и каждой архитектуре отдельно. Наконец, планируется уточнить верхнюю границу размера графа, ниже которой считается целесообразным применение алгоритмов машинного обучения.

5. Заключение. В статье представлено сравнение производительности архитектурно-независимого фреймворка VGL с другими современными решениями на двух наиболее распространенных архитектурах — многоядерных процессорах и графических ускорителях. Результат сравнения показал, что VGL в большинстве случаев заметно выигрывает у рассмотренных конкурентов на реализациях четырех широко распространенных графовых задач: поиск в ширину, нахождение кратчайших путей от вершины, ранжирование страниц и нахождение связных компонент. Также в статье отдельно приведено сравнение производительности VGL на всех четырех поддерживаемых платформах (центральные процессоры x86 с векторными расширениями, графические ускорители NVIDIA, векторные процессоры NEC SX-Aurora TSUBASA и процессоры ARM), показывающее взаимосвязь производительности реализаций графовых алгоритмов и пропускной способности памяти.

С целью дальнейшего улучшения производительности фреймворка VGL был реализован метод для автоматического выбора формата графа. Данный метод, основанный на применении алгоритма Random Forest, практически во всех случаях ускоряет выполнение графовых алгоритмов относительно вариантов со статически выбранным форматом. Стоит отметить, что на данный момент ускорение не так велико, однако применение этого метода не требует от пользователя никаких дополнительных действий, поэто-

му его использование на практике представляется оправданным. В дальнейшем планируется развивать этот метод, в частности добавить другие форматы графа и реализовать единый классификатор для всех алгоритмов и архитектур.

Список литературы

1. VGL: Vector Graph Library. https://vgl.parallel.ru/. Cited December 13, 2023.
2. *Afanasyev I., Krymskii S.* VGL rating: a novel benchmarking suite for modern supercomputing architectures // Communications in Computer and Information Science. Vol. 1618. Cham: Springer, 2022. 3–16. doi 10.1007/978-3-031-11623-0_1.
3. *Yamada Y., Momose S.* Vector engine processor of NEC's brand-new supercomputer SX-Aurora TSUBASA // Proc. Int. Symp. on High Performance Chips (Hot Chips2018). Cupertino, USA, August 19–21, 2018. https://www.old.hotchips.org/hc30/2conf/2.14_NEC_vector_NEC_SXAurora_TSUBASA_HotChips30_finalb.pdf. Cited December 13, 2023.
4. *Wang Y., Pan Y., Davidson A., et al.* Gunrock: GPU graph analytics // ACM Transactions on Parallel Computing (TOPC). 2017. **4**, N 1. Article Number 3. doi 10.1145/3108140.
5. *Osama M., Porumbescu S.D., Owens J.D.* Essentials of parallel graph analytics // Proc. IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lyon, France. May 30–June 3, 2022. doi 10.1109/IPDPSW55747.2022.00061.
6. *Shun J., Blelloch G.E.* Ligra: a lightweight graph processing framework for shared memory // SIGPLAN Not. **48**, N 8. 135–146 (2013). doi 10.1145/2442516.2442530.
7. *Shun J.* Practical parallel hypergraph algorithms // Proc. 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, USA, February 22–26, 2020. doi 10.1145/3332466.3374527.
8. *Beamer S., Asanović K., Patterson D.* The GAP benchmark suite // arXiv preprint arXiv:1508.03619. 2015. doi 10.48550/arXiv.1508.03619.
9. *Afanasyev I.V., Voevodin Vad.V., Voevodin Vl.V., et al.* Analysis of relationship between SIMD-processing features used in NVIDIA GPUs and NEC SX-Aurora TSUBASA vector processors // Lecture Notes in Computer Science. Vol. 11657. Cham: Springer, 2019. 125–139. doi 10.1007/978-3-030-25636-4_10.
10. *Афанасьев И.В.* Исследование и разработка методов эффективной реализации графовых алгоритмов для современных векторных архитектур. Диссертация на соискание ученой степени кандидата физико-математических наук, специальность 05.13.11 "Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей". М.: МГУ, 2020.
11. NVidia CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit. (Дата обращения: 13 декабря 2023).
12. *Bell N., Hoberock J.* Thrust: a productivity-oriented library for CUDA // GPU Computing Gems Jade Edition. Amsterdam: Morgan Kaufmann, 2012. 359–371. doi 10.1016/B978-0-12-385963-1.00026-5.
13. Graph500 benchmark. https://graph500.org/. (Дата обращения: 13 декабря 2023).
14. *Afanasyev I.V., Voevodin Vad.V., Voevodin Vl.V., et al.* Developing efficient implementations of shortest paths and page rank algorithms for NEC SX-Aurora TSUBASA architecture // Lobachevskii Journal of Mathematics. 2019. **40**, N 11. 1753–1762. doi 10.1134/S1995080219110039.
15. *Afanasyev I.V., Voevodin Vl.V.* Developing efficient implementations of connected component algorithms for NEC SX-Aurora TSUBASA // Lobachevskii Journal of Mathematics. 2020. **41**, N 8. 1417–1426. doi 10.1134/s1995080220080028.
16. *Chakrabarti D., Zhan Y., Faloutsos C.* R-MAT: a recursive model for graph mining // Proc. 2004 SIAM International Conference on Data Mining, April, 2004. https://epubs.siam.org/doi/epdf/10.1137/1.9781611972740.43. Cited December 13, 2023.
17. *Kunegis J.* Konect: the Koblenz network collection // Proc. 22nd Int. Conf. on World Wide Web, Rio de Janeiro, Brazil, May 13–17, 2013. New York: ACM Press, 2013. 1343–1350. http://dl.acm.org/citation.cfm?id=2488173. Cited December 13, 2023.
18. *Afanasyev I., Komatsu K., Lichmanov D., et al.* High-performance GraphBLAS backend prototype for NEC SX-Aurora TSUBASA // IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lyon, France, May 30–June 3, 2022. Piscataway: IEEE Press, 2022. 221–229. doi 10.1109/ipdpsw55747.2022.00050.
19. *Kulkarni M., Pingali K., Walter B., et al.* Optimistic parallelism requires abstractions // Proc. 28th ACM SIGPLAN Conf. on Programming Language Design and Implementation, San Diego, USA, June 10–13, 2007. New York: ACM Press, 2017. 211–222. doi 10.1145/1250734.1250759.

20. *Zhang Y., Yang M., Baghdadi R., et al.* GraphIt: a high-performance DSL for graph analytics // arXiv preprint arXiv:1805.00923. 2018. doi 10.48550/arXiv.1805.00923.
21. *Davis T.A.* Algorithm 1000: SuiteSparse: GraphBLAS: Graph algorithms in the language of sparse linear algebra // ACM Transactions on Mathematical Software (TOMS). 2019. **45**, N 4. 1–25. doi 10.1145/3322125.
22. *Boisvert R.F., Pozo R., Remington K.A.* The matrix market exchange formats: initial design // US Department of Commerce, National Institute of Standards and Technology. Gaithersburg, 1996. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5bce84be62e12ffe4d8a63fd118e4cd42f512807. Cited December 13, 2023.
23. *Chen T., Guestrin C.* XGBoost: a scalable tree boosting system // Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining. San Francisco, USA, August 13–17, 2016. New York: ACM Press, 2016. 785–794. doi 10.1145/2939672.2939785.
24. *Pedregosa F., Varoquaux G., Gramfort A., et al.* Scikit-learn: machine learning in Python // Journal of Machine Learning Research. 2011. **12**. 2825–2830. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf. Cited December 13, 2023.

Поступила в редакцию
23 ноября 2023 г.

Принята к публикации
11 декабря 2023 г.

Информация об авторах

Дмитрий Игоревич Личманов — аспирант; Московский государственный университет имени М. В. Ломоносова, Научно-исследовательский вычислительный центр, Ленинские горы, 1, стр. 4, 119234, Москва, Российская Федерация.

Илья Викторович Афанасьев — к.ф.-м.н., специалист; Московский государственный университет имени М. В. Ломоносова, Научно-исследовательский вычислительный центр, Ленинские горы, 1, стр. 4, 119234, Москва, Российская Федерация.

Вадим Владимирович Воеводин — к.ф.-м.н., зав.лаб.; Московский государственный университет имени М. В. Ломоносова, Научно-исследовательский вычислительный центр, Ленинские горы, 1, стр. 4, 119234, Москва, Российская Федерация.

References

1. VGL: Vector Graph Library. https://vgl.parallel.ru/. Cited December 13, 2023.
2. I. Afanasyev and S. Krymskii, “VGL Rating: A Novel Benchmarking Suite for Modern Supercomputing Architectures,” in *Communications in Computer and Information Science* (Springer, Cham, 2022), Vol. 1618, pp. 3–16. doi 10.1007/978-3-031-11623-0_1.
3. Y. Yamada and S. Momose, “Vector Engine Processor of NEC’s Brand-New Supercomputer SX-Aurora TSUBASA,” in *Proc. Int. Symp. on High Performance Chips (Hot Chips2018), Cupertino, USA, August 19–21, 2018*, https://www.old.hotchips.org/hc30/2conf/2.14_NEC_vector_NEC_SXAurora_TSUBASA_HotChips30_finalb.pdf. Cited December 13, 2023.
4. Y. Wang, Y. Pan, A. Davidson, et al., “Gunrock: GPU Graph Analytics,” ACM Trans. Parallel Comput. **4** (1), Article Number 3 (2017). doi 10.1145/3108140.
5. M. Osama, S. D. Porumbescu, and J. D. Owens, “Essentials of Parallel Graph Analytics,” in *Proc. IEEE Int. Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lyon, France, May 30–June 3, 2022*, doi 10.1109/IPDPSW55747.2022.00061.
6. J. Shun and G. E. Blelloch, “Ligra: A Lightweight Graph Processing Framework for Shared Memory,” SIGPLAN Not. **48** (8), 135–146 (2013). doi 10.1145/2442516.2442530.
7. J. Shun, “Practical Parallel Hypergraph Algorithms,” in *Proc. of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, USA, February 22–26, 2020.* doi 10.1145/3332466.3374527.
8. S. Beamer, K. Asanović, and D. Patterson, “The GAP Benchmark Suite,” arXiv preprint arXiv:1508.03619. doi 10.48550/arXiv.1508.03619.
9. I. V. Afanasyev, Vad. V. Voevodin, Vl. V. Voevodin, et al., “Analysis of Relationship between SIMD-Processing Features Used in NVIDIA GPUs and NEC SX-Aurora TSUBASA Vector Processors,” in *Lecture Notes in Computer Science* (Springer, Cham, 2019), Vol. 11657, pp. 125–139. doi 10.1007/978-3-030-25636-4_10.

10. I. V. Afanasyev, *Research and Development of Effective Graph Algorithms Implementation on Modern Vector Architectures*, Candidate's Dissertation in Mathematics and Physics (Moscow State Univ., Moscow, 2020).
11. NVidia CUDA Toolkit. https://developer.nvidia.com/cuda-toolkit. Cited December 13, 2023.
12. N. Bell and J. Hoberock, "Thrust: A Productivity-Oriented Library for CUDA," in *GPU Computing Gems Jade Edition* (Morgan Kaufmann, Amsterdam, 2012), pp. 359–371. doi 10.1016/B978-0-12-385963-1.00026-5.
13. Graph500 benchmark. https://graph500.org/. Cited December 13, 2023.
14. I. V. Afanasyev, Vad. V. Voevodin, Vl. V. Voevodin, et al., "Developing Efficient Implementations of Shortest Paths and Page Rank Algorithms for NEC SX-Aurora TSUBASA Architecture," Lobachevskii J. Math. **40** (11), 1753–1762 (2019). doi 10.1134/S1995080219110039.
15. I. V. Afanasyev and Vl. V. Voevodin, "Developing Efficient Implementations of Connected Component Algorithms for NEC SX-Aurora TSUBASA," Lobachevskii J. Math. **41** (8), 1417–1426 (2020). doi 10.1134/s1995080220080028.
16. D. Chakrabarti, Y. Zhan, and C. Faloutsos, "R-MAT: A Recursive Model for Graph Mining," in *Proc. 2004 SIAM Int. Conf. on Data Mining, April, 2004*, pp. 442–446. https://epubs.siam.org/doi/epdf/10.1137/1.9781611972740.43. Cited December 13, 2023.
17. J. Kunegis, "Konect: the Koblenz Network Collection," in *Proc. of the 22nd Int. Conf. on World Wide Web, Rio de Janeiro, Brazil, May 13–17, 2013* (ACM Press, New York, 2013), pp. 1343–1350. http://dl.acm.org/citation.cfm?id=2488173. Cited December 13, 2023.
18. I. Afanasyev, K. Komatsu, D. Lichmanov, et al., "High-Performance GraphBLAS Backend Prototype for NEC SX-Aurora TSUBASA," in *IEEE Int. Parallel and Distributed Processing Symposium Workshops (IPDPSW), Lyon, France, May 30–June 3, 2022* (IEEE Press, Piscataway, 2022), pp. 221–229. doi 10.1109/ipdpsw55747.2022.00050.
19. M. Kulkarni, K. Pingali, B. Walter, et al., "Optimistic Parallelism Requires Abstractions," in *Proc. 28th ACM SIGPLAN Conf. on Programming Language Design and Implementation, San Diego, USA, June 10–13, 2007* (ACM Press, New York, 2017), pp. 211–222. doi 10.1145/1250734.1250759.
20. Y. Zhang, M. Yang, R. Baghdadi, et al., "GraphIt: A High-Performance DSL for Graph Analytics," arXiv preprint, arXiv:1805.00923. 2018. doi 10.48550/arXiv.1805.00923.
21. T. A. Davis, "Algorithm 1000: SuiteSparse: GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra," ACM Trans. Math. Softw. **45** (4), 1–25 (2019). doi 10.1145/3322125.
22. R. F. Boisvert, R. Pozo, and K. A. Remington, "The Matrix Market Exchange Formats: Initial Design," US Department of Commerce, National Institute of Standards and Technology. Gaithersburg, 1996. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5bce84be62e12ffe4d8a63fd118e4cd42f512807. Cited December 13, 2023.
23. T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proc. 22nd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, San Francisco, USA, August 13–17, 2016* (ACM Press, New York, 2016), pp. 785–794. doi 10.1145/2939672.2939785.
24. F. Pedregosa, G. Varoquaux, A. Gramfort, et al., "Scikit-Learn: Machine Learning in Python," J. Mach. Lear. Res. **12**, 2825–2830 (2011), https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf. Cited December 13, 2023.

Received
November 23, 2023

Accepted for publication
December 11, 2023

Information about the authors

Dmitry I. Lichmanov — PhD student; Lomonosov Moscow State University, Research Computing Center, Leninskie Gory, 1, building 4, 119234, Moscow, Russia.

Ilya V. Afanasyev — Ph.D., Specialist; Lomonosov Moscow State University, Research Computing Center, Leninskie Gory, 1, building 4, 119234, Moscow, Russia.

Vadim V. Voevodin — Ph.D., Head of Laboratory; Lomonosov Moscow State University, Research Computing Center, Leninskie Gory, 1, building 4, 119234, Moscow, Russia.