МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

На правах рукописи

Афанасьев Илья Викторович

Исследование и разработка методов эффективной реализации графовых алгоритмов для современных векторных архитектур

Специальность 05.13.11 —
«Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей»

Автореферат
диссертации на соискание учёной степени
кандидата физико-математических наук

Москва — 2020

Работа выполнена на кафедре Суперкомпьютеров и квантовой информатики факультета Вычислительной математики и кибернетики МГУ имени М.В. Ломоносова.

Научный руководитель: доктор физико-математических наук, профессор
Воеводин Владимир Валентинович

Официальные оппоненты:

Топорков Виктор Васильевич,
доктор технических наук, профессор,
Национальный исследовательский университет «МЭИ», кафедра Вычислительных технологий, заведующий кафедрой

Ильин Вячеслав Анатольевич,
доктор физико-математических наук, старший научный сотрудник,
Научно-исследовательский центр «Курчатовский институт», Курчатовский комплекс НБИКС-природоподобных технологий, главный научный сотрудник

Петровский Михаил Игоревич,
кандидат физико-математических наук, доцент,
факультет Вычислительной математики и кибернетики МГУ имени М.В. Ломоносова, кафедра интеллектуальных информационных технологий, доцент

Защита состоится 24 декабря 2020 г. в 14 часов на заседании диссертационного совета МГУ.01.19 Московского государственного университета имени М. В. Ломоносова. по адресу: 119991, Москва, ГСП-1, Ленинские горы, МГУ, д. 1 строение 52, факультет ВМК, комната 685.

E-mail: ilgova@cs.msu.su

С диссертацией можно ознакомиться в отделе диссертаций научной библиотеки МГУ имени М.В. Ломоносова (Ломоносовский просп., д. 27). Со сведениями о регистрации участия в защите в удаленном интерактивном режиме и с диссертацией в электронном виде также можно ознакомиться на сайте ИАС «ИСТИНА»: https://istina.msu.ru/dissertations/332702119/.

Автореферат разослан «. . . » . . . . . 2020 г..

Ученый секретарь
диссертационного совета
МГУ.01.19,
кандидат физико-математических
наук

Антонов Александр Сергеевич

Общая характеристика работы

Актуальность темы. Разработка эффективных реализаций графовых алгоритмов является важной и актуальной задачей, поскольку графы исключительно удачно моделируют многие объекты реального мира из различных прикладных областей. Так, обработка графов используется при анализе социальных сетей и веб–графов, решении инфраструктурах задач, социально–экономическом моделировании, решении биологических задач и многих других. Во всех перечисленных областях моделируемые объекты представляются графами, состоящими из миллионов или даже миллиардов вершин и дуг, что делает оправданным использование суперкомпьютеров как для ускорения вычислений, так и для размещения в памяти графовых объектов столь большого размера.

Вместе с этим, вопрос, какие из современных суперкомпьютерных архитектур позволяют более быстро и эффективно решать графовые задачи остается актуальным. При решении графовых задач традиционно используются высокопроизводительные вычислительные системы как с общей, так и с распределенной памятью. Неоспоримым преимуществом систем с распределенной памятью является возможность обработки графов существенно больших размеров, что, однако часто достигается ценой значительного снижения производительности [1]. В то же время многие актуальные графовые задачи применимы и к графам меньшего размера. Так, например, граф, моделирующий связи между друзьями в социальной сети Facebook, занимает лишь 1,5 ТБ в несжатом виде [2], что позволяет разместить данный граф в памяти многих многоядерных систем с общей памятью. На практике системы с общей памятью, как правило, значительно более эффективны и в расчете производительности на ядро и энергоэффективности [3]. Современные системы с общей памятью часто оборудованы сопроцессорами различных типов (например, графическими ускорителями), обеспечивающими еще более высокие значения производительности и энергоэффективности. Учитывая все эти соображения, основной акцент данной работы сделан на создании реализаций графовых алгоритмов именно для систем с общей памятью.

Большинство графовых задач относится к классу data–intensive, то есть требующих для их решения загрузки из памяти большого объема данных, и, как следствие, сильно нагружающих подсистему памяти целевых архитектур. По этой причине решение графовых задач может быть значительно ускорено за счет использования систем с быстрой памятью (High Bandwidth Memory, HBM), которая имеет существенно большую пропускную способность по сравнению с DDR памятью, устанавливаемой вместе с большинством современных центральных процессоров. На сегодняшний день память стандарта HBM устанавливается преимущественно

в векторные архитектуры (например, NEC SX–Aurora TSUBASA [4]) либо в многоядерные центральные процессоры с векторными расширениями (например, Fujitsu A64FX), поскольку векторная обработка данных позволяет эффективно задействовать широкую шину памяти за счет обращений к подсистеме памяти от векторных устройств. Другим важным классом архитектур, использующих память стандарта HBM, являются графические ускорители NVIDIA, которые, однако, так же используют векторную обработку данных: GPU–нити группируются в так называемые варпы, в каждый момент времени выполняющие одну и ту же инструкцию над различными данными, что является основным принципом векторных вычислений.

Степень разработанности темы. Подходы к реализации графовых алгоритмов для векторных систем с быстрой памятью на сегодняшний день исследованы слабо. Основная причина – это необходимость использования регулярной по своей природе векторной обработки данных для обработки нерегулярных структур данных, какими являются графы, а также значительная новизна векторных систем, использующих стандарт HBM памяти (большинство рассматриваемых в данной работе архитектур выпущены после 2018 года). Вследствие этого, для векторных систем на сегодняшней день не предложено подходов к эффективной реализации графовых алгоритмов, а также не существует специализированных библиотек для решения графовых задач. Существующие библиотечные реализации для многоядерных центральных процессоров на векторных системах демонстрируют крайне низкую производительность. Если рассматривать класс графических ускорителей, то реальная производительность существующих реализаций на графовых задачах очень и очень низка. Все эти аргументы в совокупности говорят о том, что разработка подходов к созданию эффективных реализаций графовых алгоритмов для современных векторных систем с быстрой памятью, являющаяся предметом данного диссертационного исследования, крайне актуальна.

Целью данной работы является разработка методов, позволяющих создавать эффективные реализации графовых алгоритмов для современных векторных архитектур с быстрой памятью. Под словами «эффективные реализации» будем понимать такие реализации, которые позволяют решать графовые задачи в разы быстрее современных аналогов.

Объектом исследования диссертационной работы является методика организации эффективного решения графовых задач. Под эффективным решением понимается как выбор архитектур вычислительных систем, позволяющих получить значительную реальную производительность при решении графовых задач, так и создание высокоэффективных реализаций для выбранных архитектур.

Предметом исследования диссертационной работы является процесс создания эффективных реализаций графовых задач для современных векторных архитектур с быстрой памятью. В работе рассмотрены 9 часто возникающих на практике графовых задач и 17 наиболее распространенных алгоритмов их решения.

Для достижения поставленной цели необходимо было решить следующие **задачи**:

1. исследовать существующие подходы к реализации графовых алгоритмов на различных современных вычислительных архитектурах;
2. исследовать взаимосвязь свойств, характеристик, а также принципов разработки и оптимизации программ для различных классов современных архитектур с быстрой памятью – векторных процессоров и графических ускорителей NVIDIA GPU;
3. разработать и обосновать подход к созданию эффективных реализаций графовых алгоритмов для векторных систем с быстрой памятью, определяющий вопросы выбора алгоритмов, структур данных и целевых оптимизаций;
4. разработать и апробировать эффективные и высокопроизводительные реализации набора фундаментальных графовых алгоритмов для различных векторных архитектур с быстрой памятью;
5. провести исследование производительности, эффективности и энергоэффективности разработанных реализаций, а также провести сравнительный анализ производительности с существующими аналогами графовых библиотек и фреймворков для многоядерных центральных процессоров и графических ускорителей NVIDIA.

Научная новизна:

1. Впервые была детально изучена и описана взаимосвязь современных векторных архитектур и графических ускорителей NVIDIA: их аппаратных свойств и характеристик, а также подходов к разработке и оптимизации графовых программ для обоих классов архитектур;
2. Был проведен анализ свойств и характеристик графовых алгоритмов на предмет их соответствия современным векторным архитектурам с быстрой памятью;
3. Для широкого класса графовых алгоритмов был выделен компактный набор типовых алгоритмических структур, которые возможно эффективно реализовать на векторных архитектурах с быстрой памятью; были детально описаны подходы к реализации и оптимизации данных типовых алгоритмических структур для векторных систем с быстрой памятью;

4. Был предложен метод, позволяющий создавать эффективные реализации графовых алгоритмов для современных векторных архитектур с быстрой памятью;
5. На основе предложенного метода впервые были созданы эффективные реализации фундаментальных графовых алгоритмов для современных векторных систем с быстрой памятью, имеющие существенно более высокую производительность и эффективность по сравнению с существующими библиотечными аналогами для многоядерных центральных процессоров и графических ускорителей NVIDIA;
6. Был реализован первый в мире архитектурно–независимый графовый фреймворк, позволяющий создавать эффективные реализации широкого класса графовых алгоритмов для современных векторных систем с быстрой памятью.

Практическая значимость диссертационной работы состоит в разработке и реализации программного комплекса для решения графовых задач на современных векторных системах с быстрой памятью. Разработанный программный комплекс обладает тремя чрезвычайно важными характеристиками:

- эффективность: реализации графовых алгоритмов на основе предложенного программного комплекса обладают значительно более высокой производительностью по сравнению с существующим аналогами графовых библиотек;
- продуктивность: процесс разработки новых реализаций графовых алгоритмов заметно упрощен за счет предоставления программной среды высокого уровня, позволяющей решать за пользователя многие сложные вопросы, возникающие в процессе реализации графовых алгоритмов;
- переносимость: за счет выбора типовых алгоритмических структур, получаемые реализации можно легко переносить между рассмотренными в работе вычислительными архитектурами с быстрой памятью с сохранением высокой производительности.

Методология и методы исследования. При получении основных результатов диссертационной работы использовались элементы теории графов, методы параллельного программирования, методы анализа информационной структуры параллельных алгоритмов, а также формальные модели оценки эффективности параллельных программ и степени локальности данных. При разработке реализаций графовых алгоритмов и создании программного комплекса использовались методы объектно–ориентированного анализа и проектирования, а также программно–аппаратная архитектура параллельных вычислений CUDA, открытый стандарт для распараллеливания программ OpenMP, специализированные компиляторы с поддержкой векторизации, а также

средства анализа эффективности и производительности – CUDAToolkit, Intel Parallel Studio и инструменты компании NEC.

Основные положения, выносимые на защиту.

1. Метод создания реализаций множества распространенных графовых алгоритмов, использующий набор из четырех алгоритмических абстракций и двух абстракций данных. Показано, что с помощью этого набора можно выразить все рассмотренные графовые алгоритмы, и обоснована возможность их эффективной реализации на современных векторных архитектурах.
2. Принципы проектирования и реализации архитектурно-независимого программного комплекса VGL (графовый фреймворк) для векторных систем с быстрой памятью, позволяющего объединить сразу три важные характеристики в разработке программного обеспечения: эффективность, продуктивность и переносимость.
3. Эффективные реализации набора фундаментальных графовых алгоритмов для современных векторных архитектур с быстрой памятью – NEC SX–Aurora TSUBASA, NVIDIA GPU, Intel KNL, созданные на основе разработанного фреймворка и демонстрирующие существенно большую производительность (как правило, в разы) и энергоэффективность по сравнению с существующими библиотечными аналогами для многоядерных центральных процессоров и графических ускорителей NVIDIA.

Степень достоверности и апробация результатов. Представленные в работе результаты докладывались на следующих научных международных и российских конференциях и семинарах::

1. «15th International Conference on Parallel Computing Technologies (PaCT–2019)», Алматы, Казахстан, 19–23 августа 2019;
2. «10th JHPCN Symposium», Токио, Япония, 12–13 июля 2018;
3. «SC17: International Conference for High Performance Computing, Networking, Storage and Analysis», Denver, CO, США, 12–17 ноября 2017;
4. «Суперкомпьютерные дни в России», Москва, Россия, 23–24 сентября 2019;
5. «Параллельные вычислительные технологии (ПаВТ)», Москва, Россия, 27–29 мая 2020;
6. «Время, хаос и математические проблемы», Москва, Россия, 19 августа 2020;
7. «Ural-PDC 2018», Екатеринбург, Россия, 15 ноября 2018;
8. «Суперкомпьютерные дни в России», Москва, Россия, 24–25 сентября 2018;
9. «3rd Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists (Ural-PDC 2017)», Екатеринбург, Россия, 19 октября 2017;

10. «Суперкомпьютерные дни в России», Москва, Россия, 25–26 сентября 2017;
11. «Суперкомпьютерные дни в России», Москва, Россия, 26–27 сентября 2016;
12. Семинар ЛИТ ОИЯИ, 16 октября 2020;
13. Научный семинар кафедры ИИТ ВМК;
14. Научные семинары кафедры СКИ ВМК и НИВЦ МГУ.

Личный вклад. Личный вклад автора заключается в выполнении всего объема теоретических и экспериментальных исследований, а также в разработке архитектуры и реализации программного комплекса, предназначенного для эффективного решения графовых задач на современных векторных архитектурах с быстрой памятью. Полученные результаты диссертационной работы были оформлены автором в виде научных публикаций, а также представлены на научных конференциях. В совместных публикациях [P10–P16] автором диссертации производился всесторонний анализ графовых алгоритмов и их модификация для векторных систем с быстрой памятью, реализация и оптимизация алгоритмов на данных системах, а также сравнительный анализ производительности и эффективности с существующими аналогами. В публикации [P17] автором был сформулирован и описан метод анализа эффективности суперкомпьютерных приложений для графических ускорителей NVIDIA, основанный на анализе динамических характеристик, который был применен к задаче расчета структуры жидкокристаллических капель. Данный метод (и его модификации) в рамках данного диссертационного исследования применялся для анализа эффективности разработанных реализаций графовых алгоритмов для различных целевых архитектур.

Публикации.

Основные положения и выводы диссертационного исследования в полной мере изложены в 11 научных работах [P10–P18;T19;T20], опубликованными в том числе в 9 публикациях в рецензируемых научных изданиях [P10–P18], определенных п. 2.3 Положения о присуждении ученых степеней в Московском государственном университете имени М.В.Ломоносова.

Объем и структура работы. Диссертация состоит из введения, пяти глав и заключения. Полный объём диссертации составляет 135 страниц, включая 30 рисунков и 13 таблиц. Список литературы содержит 83 наименования.

Содержание работы

Во **введении** описывается область исследования, обосновывается актуальность работы, раскрываются ее цели, обосновываются научная новизна, практическая и теоретическая значимость работы.

Первая глава посвящена выбору целевых архитектур, позволяющих существенно ускорить решение исследуемого в работе класса графовых задач, а также исследованию существующих подходов к реализации графовых алгоритмов на выбранных архитектурах. В **разделе 1.1** выделены три основных свойства графовых алгоритмов: (1) принадлежность к классу data-intensive (производящих большое количество обменов с оперативной памятью при малом числе арифметических операций), (2) нерегулярность, а также (3) заложенный в них значительный ресурс параллелизма. Вследствие принадлежности графовых алгоритмов к классу data-intensive была выдвинута гипотеза о перспективности использования современных вычислительных систем с быстрой памятью для ускорения решения различных графовых задач. В **разделе 1.2** приведены основные современные архитектуры с быстрой памятью. На основе анализа вычислительных свойств и аппаратных характеристик перечисленных архитектур показано, что все данные системы обладают тремя характерными свойствами: (1) высокой степенью ресурса внутреннего параллелизма, (2) наличием быстрой памяти, а также (3) присутствием различного рода элементов векторной обработки данных. Данный класс систем далее будет обозначаться как «векторные архитектуры с быстрой памятью», и именно эти архитектуры используются для ускорения решения графовых задач.

В **разделе 1.4** приведен обзор существующих подходов к реализации и оптимизации графовых алгоритмов для современных многоядерных центральных процессоров, графических ускорителей NVIDIA GPU и векторных систем. Приведенный обзор демонстрирует как актуальность разработки новых подходов к реализации и оптимизации графовых алгоритмов для класса векторных систем с быстрой памятью, так и необходимость создания эффективных библиотечных реализаций графовых алгоритмов для данных систем. В **разделе 1.5** рассмотрены рассмотрены методы, позволяющие оценить производительность, локальность и эффективность разрабатываемых реализаций графовых алгоритмов – roofline–модель, а также модели пространственной и временно́й локальности. Наконец, в **разделе 1.6** приведены основные характеристики графов (как синтетических, так и реального мира), которые в ходе диссертационного исследования используются для оценки производительности, эффективности и энергоэффективности разрабатываемых реализаций.

Вторая глава посвящена исследованию взаимосвязи между различными представителями выделенного класса векторных систем с быстрой памятью, а также исследованию возможности эффективной реализации на них графовых алгоритмов. В **разделах 2.1, 2.2, 2.3** приведены основные аппаратные характеристики и архитектурные особенности используемых в данной работе систем с быстрой памятью: векторных процессоров NEC SX-Aurora TSUBASA, графических ускорителей NVIDIA GPU, а также процессоров Intel KNL.

В **разделе 2.4** для векторных архитектур и графических ускорителей NVIDIA выделено большое число схожих (а зачастую идентичных) особенностей организации потока управления, вычислений, а также работы с подсистемой памяти, большая часть которых обусловлена использованием векторного подхода к обработке данных в обоих типах систем. Наличие большого числа общих вычислительных особенностей позволяет создавать реализации графовых алгоритмов для векторных архитектур и графических ускорителей с использованием схожих подходов к реализации и оптимизации, Это, в свою очередь, позволяет разработанным в ходе данного исследования реализациям как иметь высокую степень переносимости между различными целевыми архитектурами, так и значительно опережать существующие библиотечные аналоги за счёт выбора подходов к реализации и оптимизации, согласованных с подробно описанными в данном разделе свойствами целевых архитектур (принцип суперкомпьютерного кодизайна).

В **разделе 2.5** были выделены различные классы приложений, имеющие как сравнимую, так и принципиально различную эффективность для векторных архитектур и графических ускорителей NVIDIA: различные бенчмарки, операции над плотными и разреженными матрицами, алгоритмы сортировки, и многие другие. Понятие эффективности в данном разделе определяется как процент использования пиковых аппаратных характеристик каждой из архитектур – производительности или пропускной способности памяти в зависимости от свойств приложения. Благодаря тому, что имеющие схожую степень эффективности классы приложений крайне широки, можно ожидать, что графовые алгоритмы также будут иметь потенциально схожую степень эффективности при реализации на векторных систем и графических ускорителях.

Изложенные во второй главе результаты опубликованы в работах [P13; T19].

Третья глава посвящена разработке метода создания эффективных реализаций графовых алгоритмов для векторных систем с быстрой памятью на основе выделения типовых алгоритмических абстракций.

В **разделе 3.1** обозначена важность исследования структуры графовых алгоритмов на основе анализа информационных графов. Данный анализ позволяет выделить типовые алгоритмические структуры графовых алгоритмов, которые впоследствии могут быть эффективно реализованы на векторных системах с быстрой памятью, что является важным шагом на пути к формулировке требуемого метода создания эффективных реализаций графовых алгоритмов.

В **разделе 3.2** перечислен список из 9 исследованных в работе фундаментальных графовых задач и 17 алгоритмов их решения. Примерами данных задач являются поиск в ширину, поиск кратчайших путей, поиск

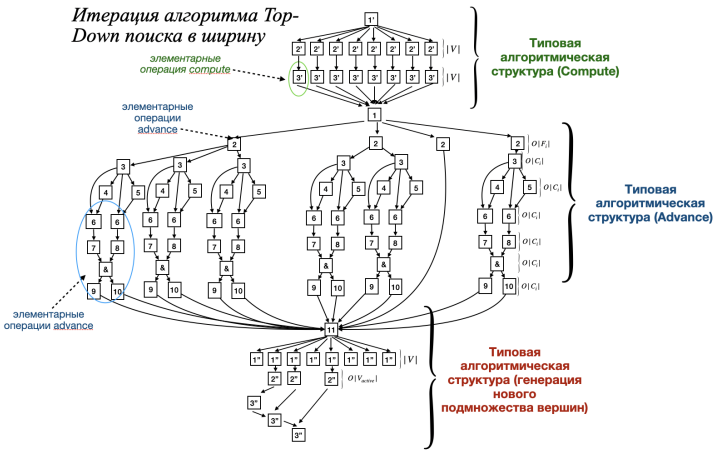

Рис. 1: Информационный граф алгоритма Top-Down поиска в ширину и выделенные в нём типовые алгоритмические структуры

связанных компонент, ранжирование вершин в графе, для которых в данной работе приведен детальный анализ информационной структуры, на основе которого в исследуемых алгоритмах выделены типовые алгоритмические структуры. «Типовая алгоритмическая структура» – подграф в информационном макрографе алгоритма, изоморфный для всех исследуемых алгоритмов при условии их применения к одним и тем же входным данным. Информационный граф алгоритма top-down поиска в ширину, в котором выделены три типовые алгоритмические структуры, приведен на Рис. 1.

Вершины информационного графа, соответствующего типовой алгоритмической структуре, отвечают за выполнение групп из элементарных операций (сложения, сравнения, присваивания, загрузка данных из памяти), отвечающих за обработку вершин и ребер входного графа на различных этапах работы алгоритма. Применяемые для обработки вершин и ребер графа операции могут значительно отличаться для различных графовых алгоритмов, вследствие чего в работе вводится термин «алгоритмический шаблон». «Алгоритмический шаблон» – группа из элементарных операций в информационном графе, применяемых в рамках выделенных типовых алгоритмических структур для обработки различных вершин и ребер входных графов. Примеры алгоритмических шаблонов для различных графовых алгоритмов приведены на Рис. 2.

Было показано, что алгоритмы решения всех рассматриваемых в работе графовых задач обладают типовыми фрагментами (алгоритмическими структурами), причем для рассмотренных 9 задач и 17 алгоритмов их решения удалось выделить всего лишь 4 типовые алгоритмические структуры, входящих в их состав. Более того, вся содержательная часть

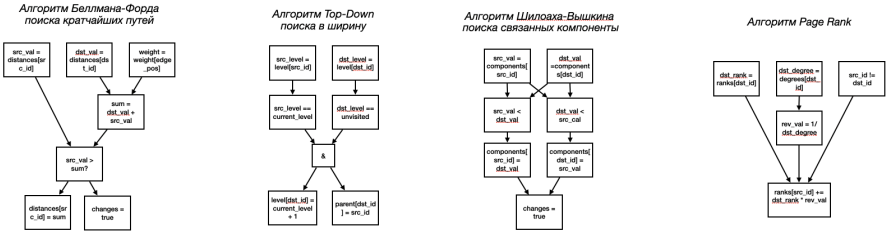

Рис. 2: Алгоритмические шаблоны нескольких исследуемых в данной работе графовых алгоритмов, отвечающие за обработку ребер графа

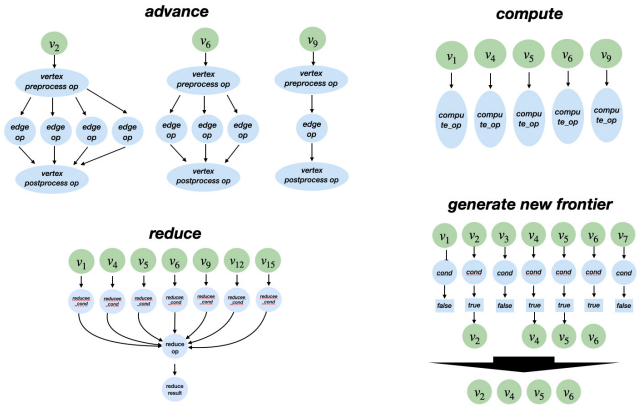

Рис. 3: Макрографы выделенных типовых алгоритмических структур: advance, compute, reduce и создания подмножества вершин

рассмотренных алгоритмов может быть выражена в терминах этого набора типовых алгоритмических структур. Для обозначения совокупности выделенных информационных структур и соответствующим им подходов к реализации для векторных архитектур с быстрой памятью в работе вводится термин «алгоритмическая абстракция», или просто «абстракция». Макрографы всех четырех выделенных алгоритмических абстракций приведены на Рис. 3.

Первой выделенной алгоритмической абстракцией является применение заданного алгоритмического шаблона к каждому из ребер графа, смежных к некоторому подмножеству вершин. Данная алгоритмическая абстракция далее в работе обозначена как «advance».

Второй выделенной алгоритмической абстракцией является применение алгоритмического шаблона к каждой из вершин заданного подмножества вершин графа. Данная алгоритмическая абстракция далее в работе обозначена как «compute».

Третьей выделенной алгоритмической абстракцией является процедура создания заданного подмножества вершин графа. Данная алгоритмическая абстракция далее в работе обозначена как «generate new frontier»[1].

Четвертой выделенной алгоритмической абстракцией является процедура применения алгоритмического шаблона к каждой из вершин заданного подмножества вершин графа, с последующей аккумуляцией некоторых значений, вычисленных для каждой из вершин. Данная алгоритмическая абстракция далее в работе обозначена как «reduce».

В **разделе 3.3** вводится понятие типовой абстракции данных. Описанные алгоритмические абстракции оперируют с тремя абстракциями данных: (1) граф, (2) подмножество вершин графа и (3) подмножество ребер графа. Поскольку все исследуемые в работе графовые алгоритмы обрабатывают **все** ребра графа, смежные к заданному подмножеству вершин, то отсутствует необходимость в явной реализации абстракции данных, отвечающей подмножеству ребер. Необходимые для работы абстракции advance подмножества ребер могут быть легко получены из подмножества вершин и непосредственно самого графа. В то же время реализация двух других абстракций данных – графа и подмножества его вершин, для векторных систем крайне актуальна, и должна хорошо соответствовать как выделенным ранее алгоритмическим абстракциям, так и аппаратным особенностям целевых векторных архитектур.

В **разделе 3.4** описываются подходы к эффективной реализации выделенных абстракций данных для векторных систем с быстрой памятью. Для этого в работе предложен векторно-ориентированный формат хранения графа VectCSR, проиллюстрированный на Рис. 4.

Во-первых, формат VectCSR использует предварительную сортировку вершин графа (оптимизацию кластеризации), тем самым позволяя эффективно использовать кэш-память при выполнении косвенных обращений к памяти. Данная оптимизация соответствует требованию локальности обработки данных. Во-вторых, разбиение вершин на группы по степени позволяет производить эффективную балансировку параллельной нагрузки, выделяя на обработку вершин различных групп разные вычислительные мощности (число ядер и различный подход к векторной обработке данных), что соответствует требованию использования массивного параллелизма целевых архитектур. В-третьих, предложенный формат VectCSR использует векторное расширение, что позволяет эффективно задействовать векторные инструкции максимальной длины для обработки вершины (или группы вершин) с любой степенью связанности, что соответствует требованию о необходимости использовать векторную обработку данных. Таким образом, предложенный в данном разделе формат VectCSR хорошо подходит для всех исследуемых векторных архитектур с быстрой памятью.

[1] «frontier» – подмножество вершин графа, образовано от понятия «фронт» вершин в алгоритме поиска в ширину.

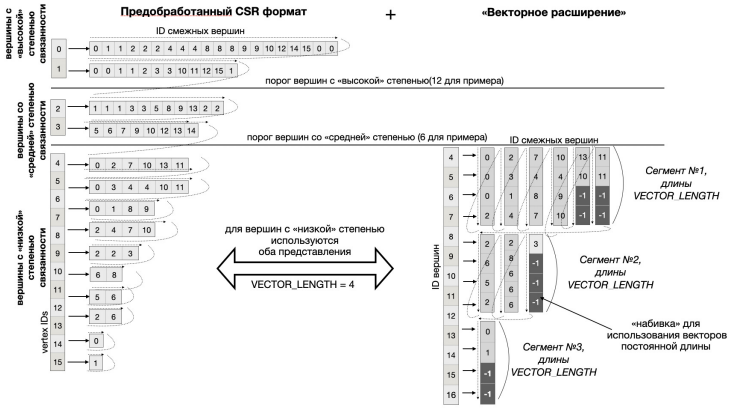

Рис. 4: Формат хранения графа VectCSR: граф в предобработанном формате CSR (слева) и его векторное расширение (справа)

Так же в данном разделе описана векторно-ориентированная реализация формата представления подмножества вершин графа. Подномжество может иметь один из трех типов «all-active», «dense» или «mixed». «All-active» – подмножества, в которые входят все вершины графа. «Dense» – подмножества, в которые входят большая часть вершин графа, реализованы в виде массива флагов принадлежности каждой из вершин к данному подмножеству. В случае, если подмножество включает в себя лишь небольшое число вершин графа, оно имеет смешанный тип («mixed»). Вершины подмножества смешанного типа разделены на группы по степени, идентичные используемым группам в формате представления VectCSR (с точно такими же порогами). Каждая из групп может иметь плотный либо разреженный тип. Плотные группы вершин представлены в виде массива флагов, в то время как разреженные – в виде списка номеров принадлежащих к группе вершин.

В **разделе 3.5** описываются подходы к эффективной реализации выделенных алгоритмических абстракций для векторных систем с быстрой памятью. Так, в **подразделе 3.5.1** для абстракции «generate new frontier» детально описан подход, позволяющей создавать подмножества вершин графа в описанном ранее векторно-ориентированном представлении на основе векторной реализации алгоритма параллельного условного копирования. В **подразделе 3.5.2** для абстракция advance описаны основные подходы к её оптимизации, такие как:

- балансировка параллельной нагрузки при обработке вершин с различной степенью,
- использование различных параллельных программных моделей,
- использование различного числа нитей и режимов балансировки,
- использование векторных инструкций максимальной длины,

Таблица 1: Сравнительная эффективность абстракции advance для архитектур NEC SX-Aurora TSUBASA и NVIDIA GPU. Все вершины и ребра графа участвуют в вычислениях

граф	SX-Aurora, EBW (Гб/с)	SX-Aurora, E (%)	P100 GPU, EBW (Гб/с)	P100 GPU, E (%)
RMAT(число вершин 2^20)	536 Гб/с	44%	417 Гб/с	57%
RMAT(число вершин 2^24)	589 Гб/с	49%	405 Гб/с	56%
liveJournal	458 Гб/с	38%	187 Гб/с	25%
twitter	340 Гб/с	28%	186 Гб/с	25%
wiki-ru	437 Гб/с	36%	341 Гб/с	47%

– улучшение шаблона доступа к памяти при доступе к ребрам графа,
– улучшение шаблона доступа к памяти при доступе к вершинам графа,
– упаковка 4-байтных косвенно запрашиваемых данных в 8-байтные,
– использование векторных регистров для хранения промежуточных результатов обработки графа,
– использование наиболее эффективного направления обхода ребер графа,
– реализация алгоритмических шаблонов без использования атомарных операций.

В **подразделах 3.5.3 и 3.5.4** описаны подходы к реализации абстракций compute и reduce. Данные абстракции могут быть реализованы для векторных архитектур крайне эффективно вследствие высокой степени регулярности данных абстракций.

В **разделе 3.6** приведены оценки эффективности реализованных абстракций для различных рассматриваемых векторных архитектур с быстрой памятью, полученные на основе roofline–модели. В **подразделе 3.6.1** показано,применительно к графовым алгоритмам roofline–модель позволяет оценить эффективность(E) реализованных абстракций согласно следующим формулам:

$EBW = \frac{bytes\_requested}{T}$ – эффективная(используемая) пропускная способность, где T – время работы исследуемой абстракции, а $bytes\_requested$ – объем запрошенных из подсистемы памяти данных;

$E = 100 * \frac{EBW}{TB}$ – эффективность, где TB – теоретическая пиковая пропускная способность памяти целевой архитектуры.

На основе данных метрик в **подразделах 3.6.1-3.6.4** приведены оценки эффективности для всех четырех реализованных абстракций. Пример оценки эффективности для абстракции advance для различных архитектур и типов входных графов приведен в Табл. 1.

В **подразделе 3.6.6** приведены динамические характеристики реализованных абстракций, отвечающие за эффективность выполнения транзакций к памяти, эффективность использования векторных инструкций

(нитей варпа), эффективность загрузки GPU и других. Замеренные значения данных характеристик подтверждают значительную эффективность всех реализованных абстракций для целевых векторных архитектур.

Наконец, в **разделе 3.7** сформулирован метод создания эффективных реализаций графовых алгоритмов для векторных систем с быстрой памятью. Согласно данному методу, для ускорения решения поставленной графовой задачи необходимо:

1. выбрать итеративный алгоритм решения данной задачи, который возможно представить в виде комбинации выделенного набора алгоритмических абстракций;
2. представить выбранный алгоритм в виде комбинации описанных алгоритмических абстракций;
3. использовать реализованные абстракции вычислений и данных для создания реализации выбранного алгоритма;
4. выбрать наиболее производительную из векторных архитектур с быстрой памятью для решения данной задачи на определенных входных данных.

Изложенные во третьей главе результаты опубликованы в работах [P10–P12; P14–P17; T20].

Четвертая глава посвящена описанию разработанного программного комплекса для создания эффективных архитектурно-независимых реализаций графовых алгоритмов. Предложенный программный комплекс является графовым фреймворком – программной средой для решения графовых задач. Графовые фреймворки включают в себя набор вычислительных абстракций и структур данных, используя которые пользователи могут реализовывать подклассы графовых алгоритмов, самостоятельно определяя лишь несложные последовательные фрагменты кода, описывающие основную вычислительную логику графовых алгоритмов.

Выделенные в предыдущей главе алгоритмические абстракции и абстракции данных могут быть использованы в качестве основы для графового фреймворка, поскольку данные абстракции позволяют описать достаточно широкий класс итеративных графовых алгоритмов. Тот факт, что данные абстракции могут быть в равной степени эффективно реализованы и для векторных процессоров, и для графических ускорителей NVIDIA, и для многоядерных центральных процессоров, позволяет реализовать архитектурно-независимый фреймворк, обеспечивающий переносимость разработанных на его основе реализаций между различными современными вычислительными системами.

В **разделе 4.1** описана актуальность разработки архитектурно-независимого фреймворка, обусловленная: (1) отсутствием фреймворков для определенных современных вычислительных архитектур (например векторных систем), (2) наличием значительного потенциала для оптимизации существующих фреймворков для многоядерных центральных процессоров

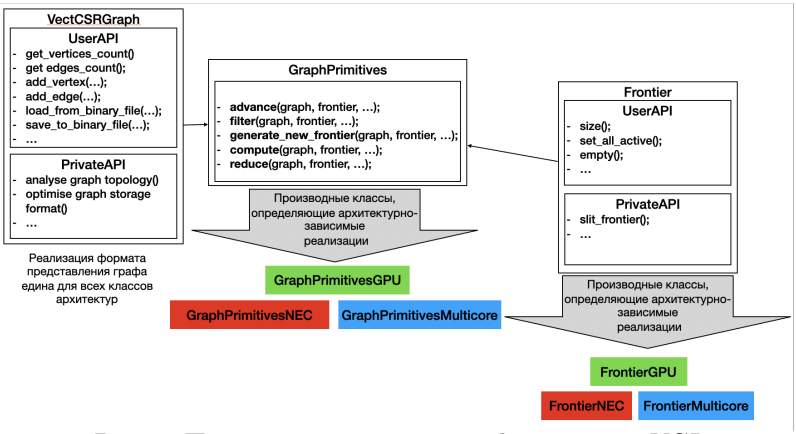

Рис. 5: Программная структура фреймворка VGL

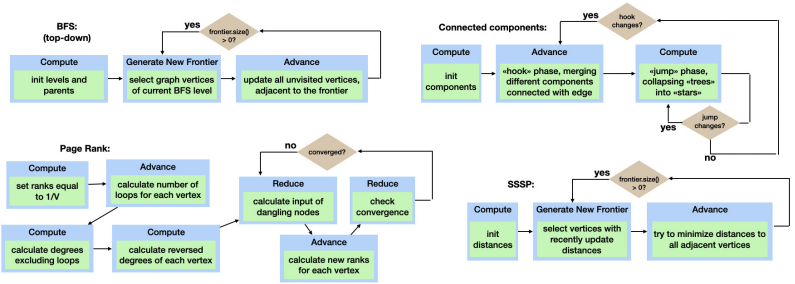

Рис. 6: Диаграмма состояний, используемая для реализации четырех графовых алгоритмов через абстракции фреймворка VGL

и графических ускорителей и (3) принципиально различным подходом к выбору абстракций в существующих фреймворках, что не позволяет эффективно переносить реализации на основе данных фреймворков между различными целевыми архитектурами. В то же время показано, что разработанный в ходе данной работе фреймворк VGL (Vector Graph Library) обладает тремя важными преимуществами по сравнению с существующим аналогами: (1) эффективностью, (2) продуктивностью и (3) переносимостью.

В **разделе 4.2** описаны программные интерфейсы пользователя для каждой из используемых во фреймворке VGL абстракций вычислений и данных: graph, frontier, advance, compute, reduce и generate_new_frontier. При использовании фреймворка VGL пользователь передаёт в реализации данных абстракций вычислений lambda-функции языка C++, определяющие вычислительную логику обработки вершин и ребер в графовых алгоритмах.

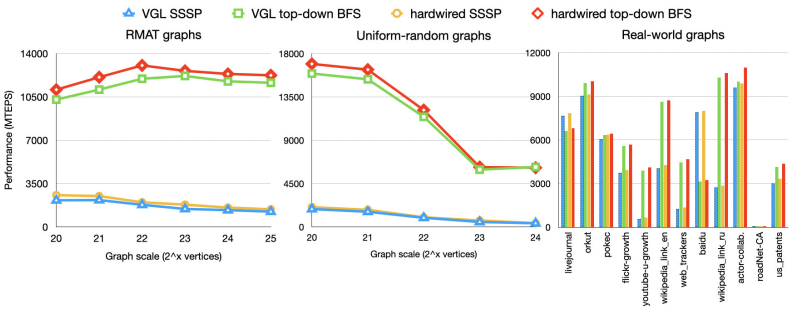

Рис. 7: Сравнение производительности оптимизированных «вручную» реализаций алгоритмов поиска в ширину и поиска кратчайших путей в графе с реализациями на основе фреймворка VGL

В **разделе 4.3** описана программная структура фреймворка VGL, приведенная на Рис. 5. Для трех классов вычислительных систем (векторные архитектуры, многоядерные центральные процессоры и графические ускорители) реализованы отдельные C++ классы, соответствующие набору абстракций вычислений, а также двум абстракциям данных для различных архитектур. Данные классы являются производными от базовых классов, в которых определен единый интерфейс пользователя для каждой из абстракций вычислений и данных. Таким образом, в производных классах реализованы оптимизированные реализации абстракций вычислений и структур данных, несколько отличающиеся для различных архитектур, однако имеющие единый интерфейс, определяемый в базовых классах.

В **разделе 4.4** описаны типовые подходы к реализации графовых алгоритмов на основе фреймворка VGL. Так, на Рис. 6 приведена схема реализаций четырех графовых алгоритмов решения задач PR (Page Rank), BFS (Breadth-First Search), CC (Connected Components) и SSSP (Single Source Shortest Paths). Данная схема представляет из себя диаграмму, состояния которой описывают возможные подходы к реализации описывающих основную вычислительную логику алгоритмов lambda-функций, а переходы между состояниями – последовательность вызова вычислительных абстракций фреймворка.

В **разделе 4.5** приведены примеры программного кода, реализующего алгоритмы top-down поиска в ширину и Беллмана-Форда поиска кратчайших путей в графе на основе фреймворка VGL. В разделе **разделе 4.6** приведено сравнение производительности оптимизированных «вручную» реализаций алгоритмов поиска кратчайших путей и поиска в ширину с реализациями, разработанными на основе предложенного фреймворка VGL, результаты которого приведены на Рис. 7. Изложенные в четвертой главе результаты опубликованы в [P18].

Пятая глава посвящена анализу производительности, эффективности и энергоэффективности разработанных реализаций на основе предложенного фреймворка, а также сравнению его с существующими аналогами. В **разделе 5.1** приведен сравнительный анализ производительности фреймворка VGL с такими существующими фреймворками и библиотеками для графических ускорителей как Gunrock [5], cuSHA [6], Enterprise [7], NVGRAPH и Lonestar GPU [8], а также Ligra [3], Galois [8], GAPBS [9] для многоядерных центральных процессоров. Результаты сравнительного анализа производительности (измеряемой при помощи метрики TEPS – Traversed Edges Per Second) приведены на рисунках 8 и 9 для задач ранжирования вершин в графе и поиска связанных компонент соответственно. Для наглядности реализации но основе предложенного фреймворка VGL обозначены линиями с треугольниками.

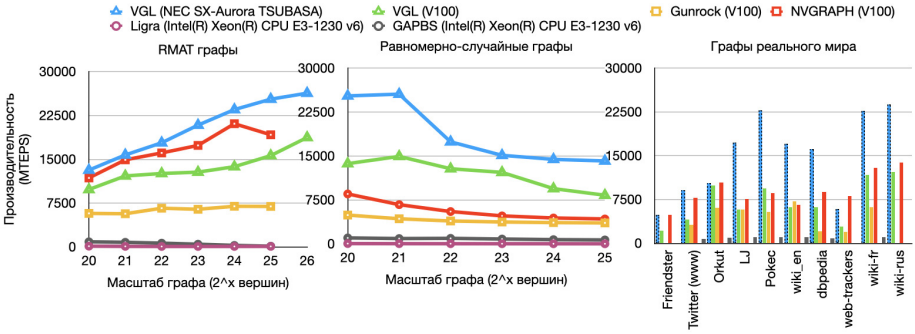

Рис. 8: Производительность реализации алгоритма Page Rank ранжирования вершин в графе по сравнению с наиболее быстрыми мировыми аналогами графовых библиотек и фреймворков

Таблица 2: Оценки эффективности (через используемую пропускную способность памяти) для реализаций алгоритмов решения задачи поиска кратчайших путей (SSSP) на основе фреймворка VGL

Граф	SX-Aurora, используемая ППС, Гб/с	SX-Aurora, % от пиковой ППС	P100 GPU, используемая ППС, Гб/с	P100 GPU, % от пиковой ППС
RMAT (масштаб 23)	556	46%	436	60%
RMAT (масштаб 25)	532	44%	513	71%
Uniform-Random (масштаб 21)	809	67%	133	18%
Twitter	307	26%	174	24%
Wiki_en	497	41%	215	29%
Pockec	647	53%	200	27%

Проведенное сравнение показывает, что реализации на основе фреймворка VGL для векторных систем с быстрой памятью обеспечивают

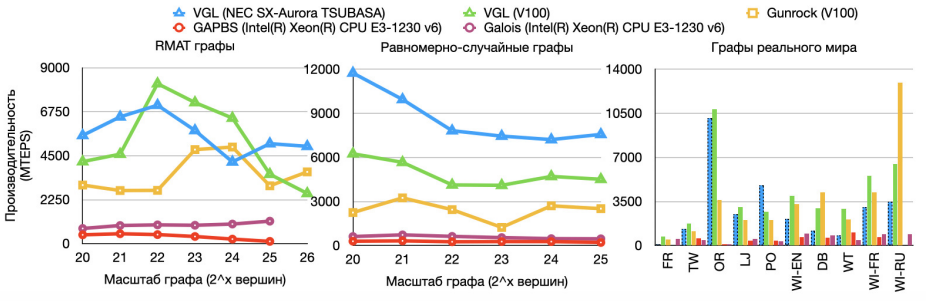

Рис. 9: Производительность реализации алгоритма Шилоаха-Вышкина поиска связанных компонент в графе по сравнению с наиболее быстрыми мировыми аналогами графовых библиотек и фреймворков

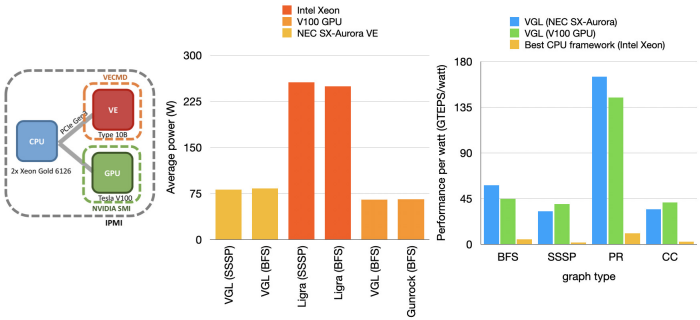

Рис. 10: Использованные средства для измерения потребляемой мощности (слева), средняя потребляемая мощность для различных рассматриваемых архитектур (посередине), производительность на единицу мощности для различных графовых задач(справа)

ускорение от 3 до 14 раз по сравнению с существующими библиотечными аналогами для многоядерных процессоров, а также 1.5 до 3 раз по сравнению с аналогами для графических ускорителей.

В **разделе 5.2** приведены оценки эффективности разработанных реализаций графовых алгоритмов с использованием введенных в главе 3 метрик эффективности на основе roofline–модели. В Табл. 2 данные оценки эффективности приведены для задачи поиска кратчайших путей.

В **Разделе 5.3** приведены оценки, демонстрирующие значительно более высокую энергоэффективность реализаций на основе фреймворка VGL по сравнению с аналогами для многоядерных центральных процессоров. На Рис. 10 приведено сравнение энергоэффективности на основе двух метрик: средней потребляемой мощности, а также производительности

на единицу потребляемой мощности. Таким образом, вклад в энергоэффективность VGL–реализаций вносится как за счет значительно большей производительности VGL–реализаций (до 14 раз по сравнению с аналогами для Intel Xeon), так и за счет значительно более низкой потребляемой мощностью архитектурами NEC SX-Aurora TSUBASA и NVIDIA GPU (до 5 по сравнению с Intel Xeon) – итого реализации на основе VGL от 40 до 70 раз более эффективны по сравнению с реализациями для многоядерных центральных процессоров (Рис. 10 справа).

В **заключении** приведены основные результаты работы, которые заключаются в следующем:

1. Опираясь на принципы суперкомпьютерного кодизайна и анализ информационной структуры типовых графовых алгоритмов, в работе предложен метод создания реализаций графовых алгоритмов, использующий набор из четырех алгоритмических абстракций и двух абстракций данных. Показано, что с помощью этого набора можно выразить все рассмотренные графовые алгоритмы, и обоснована возможность их эффективной реализации на современных векторных архитектурах.
2. Предложенный набор алгоритмических абстракций и абстракций данных составил основу для проектирования и реализации архитектурно-независимого программного комплекса VGL (графовый фреймворк) для векторных систем с быстрой памятью, позволяющего объединить сразу три важные характеристики в разработке программного обеспечения: эффективность, продуктивность и переносимость.
3. На основе разработанного фреймворка были созданы эффективные реализации набора фундаментальных графовых алгоритмов для современных векторных архитектур с быстрой памятью – NEC SX–Aurora TSUBASA, NVIDIA GPU, Intel KNL, демонстрирующие существенно большую производительность (как правило, в разы) и энергоэффективность по сравнению с существующими библиотечными аналогами для многоядерных центральных процессоров и графических ускорителей NVIDIA.

Список литературы

1. *McSherry Frank, Isard Michael, Murray Derek G.* Scalability! But at what {COST}? // 15th Workshop on Hot Topics in Operating Systems (HotOS {XV}). — 2015.

2. Four degrees of separation / Lars Backstrom, Paolo Boldi, Marco Rosa et al. // Proceedings of the 4th Annual ACM Web Science Conference. — 2012. — Pp. 33–42.

3. *Shun Julian, Blelloch Guy E.* Ligra: a lightweight graph processing framework for shared memory // ACM Sigplan Notices / ACM. — Vol. 48. — 2013. — Pp. 135–146.

4. *Yamada Yohei, Momose Shintaro.* Vector Engine Processor of NEC¡Çs Brand-New supercomputer SX-Aurora TSUBASA // Intenational symposium on High Performance Chips (Hot Chips2018). — 2018.

5. Gunrock: A high-performance graph processing library on the GPU / Yangzihao Wang, Andrew Davidson, Yuechao Pan et al. // Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. — 2016. — Pp. 1–12.

6. CuSha: vertex-centric graph processing on GPUs / Farzad Khorasani, Keval Vora, Rajiv Gupta, Laxmi N Bhuyan // Proceedings of the 23rd international symposium on High-performance parallel and distributed computing. — 2014. — Pp. 239–252.

7. *Liu Hang, Huang H Howie.* Enterprise: breadth-first graph traversal on GPUs // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. — 2015. — Pp. 1–12.

8. *Nguyen Donald, Lenharth Andrew, Pingali Keshav.* A lightweight infrastructure for graph analytics // Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. — 2013. — Pp. 456–471.

9. *Beamer Scott, Asanović Krste, Patterson David.* The GAP benchmark suite // *arXiv preprint arXiv:1508.03619.* — 2015.

Публикации автора по теме диссертации в изданиях, индексируемых в базах данных Web of Science, Scopus, RSCI и РИНЦ

P10. Developing Efficient Implementations of Bellman–Ford and Forward-Backward Graph Algorithms for NEC SX-ACE / Ilya V Afanasyev,

Alexander S Antonov, Dmitry A Nikitenko et al. // *Supercomputing Frontiers and Innovations*. — 2018. — Vol. 5, no. 3. — Pp. 65–69. — [Scopus, Impact Factor: 0.422].

P11. Developing Efficient Implementations of Shortest Paths and Page Rank Algorithms for NEC SX-Aurora TSUBASA Architecture / IV Afanasyev, Vad V Voevodin, Vl V Voevodin et al. // *Lobachevskii Journal of Mathematics*. — 2019. — Vol. 40, no. 11. — Pp. 1753–1762. — [Scopus, Impact Factor: 0.238].

P12. *Afanasyev IV, Voevodin Vl V*. Developing Efficient Implementations of Connected Component Algorithms for NEC SX-Aurora TSUBASA // *Lobachevskii Journal of Mathematics*. — 2020. — Vol. 41, no. 8. — Pp. 1417–1426. — [Scopus, Impact Factor: 0.422].

P13. Analysis of relationship between simd-processing features used in nvidia gpus and nec sx-aurora tsubasa vector processors / Ilya V Afanasyev, Vadim V Voevodin, Vladimir V Voevodin et al. // International Conference on Parallel Computing Technologies / Springer. — 2019. — Pp. 125–139.

P14. *Afanasyev Ilya, Voevodin Vladimir*. The comparison of large-scale graph processing algorithms implementation methods for Intel KNL and NVIDIA GPU // Russian Supercomputing Days / Springer. — 2017. — Pp. 80–94. — [Scopus, Impact Factor: 0.7].

P15. Techniques for Solving Large-Scale Graph Problems on Heterogeneous Platforms / Ilya Afanasyev, Alexander Daryin, Jack Dongarra et al. // Russian Supercomputing Days / Springer. — 2016. — Pp. 318–332. — [Scopus, Impact Factor: 0.7].

P16. Developing an Efficient Vector-Friendly Implementation of the Breadth-First Search Algorithm for NEC SX-Aurora TSUBASA / Ilya V Afanasyev, Vladimir V Voevodin, Kazuhiko Komatsu, Hiroaki Kobayashi // International Conference on Parallel Computational Technologies / Springer. — 2020. — Pp. 131–145. — [Scopus, Impact Factor: 0.7].

P17. Практика проведения анализа производительности суперкомпьютерных задач / Илья Викторович Афанасьев, Вадим Владимирович Воеводин, Владимир Юрьевич Рудяк, Александр Вячеславович Емельяненко // *вычислительные методы и программирование*. — 2019. — Vol. 20. — Pp. 346–355. — [RINC, Impact Factor: 0.429].

P18. *Афанасьев ИВ*. Разработка прототипа высокопроизводительного графового фреймворка для векторной архитектуры NEC SX–Aurora TSUBASA // *Вычислительные методы и программирование*. — 2020. — Vol. 21. — Pp. 290–305. — [RINC, Impact Factor: 0.429].

Иные публикации

T19. *Ilya Afanasyev*. The Comparative Performance Analysis of Data-intensive Applications for IBM Minsky and Newell Systems // Proceedings of the 4th Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists / CEUR. — 2018. — Pp. 40–49.

T20. *Afanasyev Ilya*. An Efficient Implementation of the Transitive Closure Problem on Intel KNL Architecture // Proceedings of the 3th Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists / CEUR. — 2017. — Pp. 10–19.