Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Балашов Илья Вадимович

Исследование эффективности применения специализации при разработке алгоритмов, основанных на методах линейной алгебры

Бакалаврская работа

Научный руководитель:
ст.преп. факультета МКН, к.ф.-м.н. Д.А. Березун

Консультант:
доц. кафедры информатики, к.ф.-м.н. С.В. Григорьев

Рецензент:
ст.преп., «Санкт-Петербургский политехнический университет Петра Великого»
М.Х.Ахин

Санкт-Петербург
2021

SAINT PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Ilya Balashov

Effectiveness of partial evaluation of linear algebra-based algorithms

Bachelor's Thesis

Scientific supervisor:
Senior Lecturer Daniil Berezun, PhD.

Scientific consultant:
Assistant Professor Semyon Grigorev, PhD.

Reviewer:
Senior Lecturer, Peter the Great St. Petersburg Polytechnic University
Marat Akhin

Saint-Petersburg
2021

Оглавление

Введение

В связи с возрастанием объемов и сложности программ в последние годы в промышленной разработке появилась необходимость в дополнительных автоматических инструментах и техниках. В частности, особо важными оказываются техники автоматической трансформации кода, изменяющие некоторые свойства получаемых из этого кода программ. Одна из таких техник — специализация, также известная как частичные или смешанные вычисления [1]. Специализация позволяет использовать известную перед компиляцией часть входных данных программы для получения новой программы. Гарантируется, что новая программа будет вести себя аналогично исходной. Очень часто данная техника используется для генерации программ, имеющих ту же семантику, но оптимизированных по времени исполнения.

Специализация применима для сокращения времени исполнения программ из большого количества областей, например, трассировки лучей [2], биоинформатики [3], обработки изображений [4]. Кроме того, одной из классических областей применения специализации являются строковые алгоритмы.

Как известно, большое количество алгоритмов на графах можно реализовать методами линейной алгебры через определённый набор матричных операций [5]. К примеру, алгоритм поиска в ширину, традиционно реализуемый посредством поочередного добавления вершин в очередь, может быть выражен через многократное умножение матрицы смежности графа на некоторый вектор. Подобная форма графовых алгоритмов в последние годы получила широкое теоретическое и практическое применение в задачах анализа графовых данных разных видов. Одной из таких задач является выполнение контекстно-свободных запросов (CFPQ) к графовым базам данных, внутренняя структура которых на высоком уровне представляет собой разреженный граф большого размера [6].

В связи с распространением вычислений над графами в матричной форме были предприняты попытки стандартизировать базовые про-

граммные блоки, из которых может быть сконструирован произвольный алгоритм на графах. Одним из подобных стандартов является GraphBLAS [7]. В этом стандарте основными базовыми операциями являются умножение матриц и векторов, а также произведение Кронекера (тензорное), выполняемые над различными полукольцами. Для выполнения таких операций существует широкий класс алгоритмов с различной вычислительной сложностью. Более того, из года в год появляются новые алгоритмы умножения матриц со всё меньшей и меньшей сложностью, однако не гарантируется, что во всех реализациях GraphBLAS или других стандартов реализованы именно алгоритмы с наименьшей известной сложностью. Поэтому оптимизация данных базовых блоков по времени исполнения может позволить существенно сократить время исполнения кода произвольных графовых алгоритмов.

Также, существенное значение в разработке больших программ и библиотек имеют алгоритмы на строках [8]. К примеру, алгоритмы поиска подстроки по регулярному выражению и сопоставления шаблонов (поиска подстроки) могут применяться как в универсальных библиотеках обработки строк, так и в особых задачах, например, CFPQ [6]. Более того, многие строковые алгоритмы используются как тесты производительности для разных инструментов и методик, в частности, инструментов специализации. Поэтому оптимизация подобных базовых алгоритмов на строках может иметь как прикладное, так и теоретическое значение.

Как правило, наиболее оптимальные версии таких алгоритмов, как умножение матриц или сопоставление с регулярными выражениями, весьма сложны для реализации и понимания рядовыми программистами. Техника специализации теоретически позволяет приблизить время исполнения кода базовых версий алгоритмов ко времени исполнения их наиболее оптимальных “ручных” реализаций. Поэтому применение специализации в промышленной разработке может существенно упростить программистам создание качественного алгоритмически насыщенного кода. Тем не менее, перед внедрением этой техники важно выяснить, насколько оправданной на практике окажется специализация различ-

ных базовых алгоритмов.

1. Постановка задачи

Целью данной работы является исследование эффективности специализации для базовых матричных и строковых алгоритмов в матричной форме, широко используемые в некоторых областях.

Для достижения поставленной цели были сформулированы следующие задачи.

1. Выполнить обзор предметной области, рассмотрев специализацию и существующие инструменты для её проведения, а также алгоритмы и наборы данных из различных областей, к которым специализация применима.

2. Спроектировать экспериментальное исследование специализации матричных алгоритмов:

   - выбрать алгоритмы, пригодные для данного эксперимента, а также инструмент для специализации;
   - определить набор вопросов для исследования, на которые эксперимент должен дать ответ;
   - задать набор метрик, которые позволят оценить результат специализации.

3. Выделить графовые алгоритмы, предположительно хорошо поддающиеся специализации.

4. Реализовать алгоритмы, выбранные для экспериментов.

5. Выполнить эксперименты и проанализировать результаты.

2. Обзор предметной области

В данной главе приведён обзор идей стандарта GraphBLAS для конструирования графовых алгоритмов на языке линейной алгебры, а также понятий и инструментов, используемых для специализации.

2.1. Алгоритмы, реализованные методами линейной алгебры

Из теории графов известно, что граф может быть представлен в матричном виде: матрица смежности, матрица инцидентности и так далее [9]. Более того, многие графовые операции и алгоритмы могут быть выражены как линейные операции над матричным представлением графа [10].

Рассмотрим в качестве примера алгоритм поиска в ширину.

- В классическом определении, этот алгоритм заключается в занесении в хвост очереди всех непосещённых соседей данной вершины и повторение алгоритма от головы очереди.

- Через матрицы алгоритм может быть представлен указанным ниже образом.

  Пусть A — матрица смежности некоторого графа (размерность $N \times N$). v — столбец высоты N вида $\begin{pmatrix} 0 & 0 & .. & 1 & 0 & .. \end{pmatrix}^T$, где единица стоит на строке с номером, соответствующим начальной вершине обхода, как на формуле (1). Тогда шаг обхода в ширину (поиск соседей головы очереди при классической интерпретации) может быть представлен как $A^T v$, где единицы будут соответствовать вершинам на следующем шаге.

Как можно видеть на примере формулы (1) и рисунка 1, результаты шага алгоритма в классическом и матричном представлении будут

совпадать [10].

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}^T \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \tag{1}$$

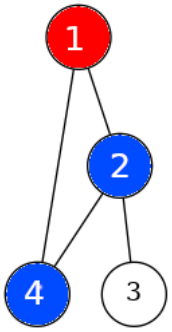


Рис. 1: Визуализация первого шага обхода в ширину.

Одной из наиболее важных абстракций при работе с графовыми алгоритмами на языке линейной алгебры является полукольцо. Полукольцом называют множество S, обладающее перечисленными ниже свойствами [11].

- S — коммутативный моноид по сложению.
- S — полугруппа по умножению.
- Дистрибутивность умножения относительно сложения.
- Мультипликативность относительно нуля.

Нестандартные полукольца могут быть использованы для конструирования ещё более сложных графовых алгоритмов на языке линейной алгебры. К примеру, можно ввести следующее полукольцо: минимум в качестве сложения и вещественное сложение в качестве умножения. Тогда алгоритм нахождения кратчайшего пути в графе может быть выражен через сложение и умножение матриц смежности в этом полукольце [10].

Для реализации описанных идей сообществом был создан открытый стандарт GraphBLAS [7]. Он определяет основные блоки для конструирования графовых алгоритмов в терминах линейной алгебры и на данный момент является единственным известным стандартом такого рода. Основной его особенностью является прикладной характер, а также эффективность основных реализаций, таких как SuiteSparse [12]. Тем не менее, работа со стандартом на практике осложняется необходимостью хорошего знания линейной алгебры и теории графов для создания эффективных алгоритмов.

2.2. Специализация

Специализация представляет из себя метод агрессивной оптимизации программ. Содержанием данного метода, основные идеи которого изложены в [13], является автоматическая генерация на основе входной процедуры, а также некоторой заранее известной части её входных параметров (которые можно понимать как ограничения на программу) новой процедуры, на оставшейся части входных параметров (то есть на данных ограничениях) полностью эквивалентной изначальной с точки зрения результата. При этом благодаря применённой технике новая процедура может обладать существенно более высокой производительностью по времени исполнения, менее сложной структурой кода, а также меньшим размером, нежели изначальная. Инструмент, при помощи которого осуществляется автоматическая специализация, называют специализатором.

Классическим примером применения специализации является оптимизация алгоритма возведения числа в фиксированную заранее степень.

Пусть имеется вещественное число x, а также целое положительное число n. Тогда алгоритм возведения в степень, реализованный на псевдокоде, может выглядеть указанным на листинге 1 образом.

После специализации такого алгоритма на параметр $n = 5$ рекурсивные вызовы могут быть развёрнуты в линейную функцию. Возможный

```
1 def power(x, n):
2     if n == 0: 1
3     else if n mod 2 == 0: power(x, n / 2) ↑ 2
4     else: x * power(x, n − 1)
```

Листинг 1: Функция возведения в степень до специализации.

результат представлен на листинге 2.

```
1 def fifthPower(x):
2     ((x ↑ 2) ↑ 2) * x
```

Листинг 2: Специализированная функция возведения в степень 5.

Существует две основных стратегии специализации по времени её выполнения [14].

1. Офлайн-специализация — решения о специализации конкретных элементов программы принимается перед трансляцией. Для этого используется анализ времени связывания (binding-time analysis), который отделяет код, опирающийся на статически-известные данные, от всего остального.

2. Онлайн-специализация — решения о специализации принимаются непосредственно во время трансляции.

Офлайн-специализация в общем случае может быть применена для работы с более сложными языковыми конструкциями, в то время как онлайн-специализация в общем случае проще в реализации [14].

2.3. Инструменты для специализации

На момент выполнения работы были найдены следующие специализаторы, потенциально подходящие для оптимизации произвольного кода.

- **Библиотека LLPE**[1] является специализатором промежуточного кода инфраструктуры для построения компиляторов LLVM.

[1]Сайт проекта LLPE: `http://www.llpe.org/` (Дата обращения: 04.05.2021)

LLPE предлагает программисту хорошую документацию и удобство использования. Тем не менее, данная библиотека заявлена как нестабильная, что ограничивает возможности по использованию библиотеки для создания качественной пользовательской библиотеки.

- **Фреймворк AnyDSL** [15] обеспечивает поддержку офлайн–специализации объектов в пользовательских библиотеках. Он представляет из себя прослойку между LLVM и библиотекой, отличается стабильностью и поддержкой от разработчиков и сообщества. Кроме того, AnyDSL имеет поддержку GPU, что может быть полезным для дальнейших экспериментов. AnyDSL использует отдельные языки для написания кода для специализатора — Impala [2] и Artic [3], что, вместе с наличием дополнительных надстроек над LLVM, с одной стороны, существенно усложняет использование специализатора в прикладных задачах, но с другой стороны, делает более доступным массовое программирование экспериментов за счёт более выразительного языка в данном конкретном случае.

- **LLVM.mix** [16] является офлайн–специализатором для промежуточного кода LLVM IR. Главной особенностью системы является отсутствие необходимости в отдельном языке программирования, поскольку конструкции управления реализованы как атрибуты компилятора, обрабатываемые модифицированным LLVM-фронтендом Clang.mix[4]. Кроме того, сама система с точки зрения реализации представляет из себя проход для оптимизатора LLVM, что позволяет в общем случае обеспечить достаточно высокую производительность. Тем не менее, как показали эксперименты, проведённые в прошлом году [17], LLVM.mix не является стабильным и не позволяет создавать пользовательские библиотеки в связи с необходимостью для каждой задачи пересобирать

[2]Сайт AnyDSL и Impala: `https://anydsl.github.io/` (Дата обращения: 04.05.2021)
[3]Репозиторий Artic: `https://github.com/AnyDSL/artic` (Дата обращения: 04.05.2021)
[4]Репозиторий Clang.Mix: `https://github.com/eush77/clang.mix` (Дата обращения: 04.05.2021)

потенциальную библиотеку с помощью специального фронтенда Clang.mix.

2.4. Алгоритмы, пригодные для специализации

В данном разделе будет проведен обзор алгоритмов, потенциально пригодных для проведения с ними дальнейших исследований по специализации.

В соответствии с поставленными задачами необходимо выбрать несколько алгоритмов, которые с точки зрения теории имеют хороший потенциал для оптимизации с использованием техники специализации и, с другой стороны, могут продемонстрировать качество оптимизаций, проводимых выбранным специализатором в целом, а также конкретно в задаче специализации алгоритмов, выразимых через GraphBLAS.

Таким образом, были рассмотрены перечисленные ниже алгоритмы.

- **Умножение разреженных матриц**. Алгоритм представляет собой умножение разреженной матрицы, хранящейся в представлении COO, на разреженную или плотную матрицу небольшой размерности (порядка не более нескольких десятков), объявляемую статической, в том же представлении. Такой алгоритм может быть специализирован в связи с наличием линейной структуры у его тривиальной реализации. Кроме того, задача умножения матриц имеет большое прикладное значение в связи с её основополагающим значением в большом количестве областей, например, в GraphBLAS — результаты специализации можно считать базовым тестом перед реализацией подмножества стандарта.

- **Тензорное произведение операторов с разреженной матрицей**. Алгоритм тензорного произведения в данном экспериментальном исследовании принимает в качестве исходных данных разреженную матрицу, а также вторую матрицу произвольного вида, причём последняя матрица имеет размерность порядка нескольких десятков, и для специализатора отмечается как статическая.

Данный алгоритм может быть специализирован в силу наличия достаточного количества статических элементов в его реализации в частности внутренний цикл по статической матрице. На практике тензорное произведение используется в области графовых баз данных [18]: динамическая разреженная матрица — матричное представление графа, плотная статическая — запрос, а их тензорное произведение представляет собой выполнение регулярного запроса к графовой базе данных. Данный тест выбран к рассмотрению, поскольку в рамках прошлогодней работы [17] были получены хорошие результаты по специализации тензорного произведения, и такой эксперимент наглядно продемонстрирует качество проводимых оптимизаций в сравнении со специализатором LLVM.mix.

- **Сопоставление шаблонов**. В тривиальной реализации алгоритм представляет собой поэлементное сопоставление искомого шаблона с каждым суффиксом строки поиска. В качестве примера практического приложения алгоритма сопоставления шаблонов можно назвать молекулярную биологию [19], а также биоинформатику. Данный алгоритм был выбран к рассмотрению как часть так называемого КМП-теста [20], который показывает, насколько специализатору удалось приблизить время выполнения тривиального алгоритма поиска подстроки до времени выполнения алгоритма Кнута-Морриса-Пратта (теоретически специализация может полностью свести тривиальный алгоритм к нему по производительности [21]), и является одним из базовых тестов для проверки качества специализатора общего назначения.

- **Сопоставление с регулярным выражением**. Алгоритм проверяет принадлежность строки ко множеству последовательностей символов из заданного алфавита, порождаемых регулярным выражением. Регулярное выражение представляется в виде матрицы смежности графа минимизированного детерминированного конечного автомата, ему соответствующего. Благодаря матрич-

ному представлению регулярного выражения алгоритм сопоставления шаблонов приобретает линейный вид, и поэтому теоретически может быть успешно подвергнут специализации. Об этом также свидетельствуют выводы Н.Джонса [1]. В качестве применений алгоритма на практике можно назвать глубокую инспекцию пакетов (DPI) [22] или обработку естественного языка [23]. Данный алгоритм был выбран к рассмотрению для экспериментов как усложнённая версия КМП-теста, которая может показать, насколько хорошо специализатор справляется с алгоритмами с увеличенным количеством условных переходов.

2.5. Данные для экспериментов

- В качестве **матричных данных** для экспериментов были выбраны разреженные матрицы из набора SuiteSparse Matrix Collection [24], а в частности, основная часть была взята из его подмножества Hardwell-Boeing [5] Более конкретно, были задействованы матрицы, представленные на Таблице 1. Первые три матрицы из таблицы были использованы в левой части операторов матричного произведения и произведения Кронекера в связи с их относительно большим размеров, остальные были применены в правой части данных операторов. Как видно из Таблицы 1, использованные для экспериментов матрицы имеют различные показатели количества ненулевых элементов, симметрии (процент пар одинаковых ненулевых элементов, расположенных симметрично относительно главной диагонали), а также типа значений. Кроме того, *bcsstk16* и *eye3* вырожденные — ненулевые элементы сконцентрированы у главной диагонали матрицы. Поэтому тестирование применения матричных алгоритмов ко всем парам указанных данных позволяет довольно полно и точно показать результаты специализации как в общих, так и в различных особенных случаях.

[5]Сайт набора Hardwell-Boeing: `https://math.nist.gov/MatrixMarket/collections/hb.html` (Дата обращения: 28.04.2020).

	Размер	Ненулевые	Симметрия, %	Тип значений
bcsstk16	4884	147631	100	Вещественные
fs_183_1	183	1069	41.8	Вещественные
can_256	256	2916	100	Двоичные
eye3	3	3	100	Двоичные
2blocks	4	8	100	Двоичные
cover	8	12	16.67	Двоичные
mycielskian3	6	5	0	Двоичные
trec5	8	12	0	Вещественные

Таблица 1: Матрицы, использованные в экспериментах по специализации.

Для представления матриц использовали формат COO (COOrdinate list), в котором матрица представляется в виде троек: номера строки и столбца, а также значения на данной позиции в матрице, — в связи с предполагаемо более линейной структурой специализированного кода при использовании этого формата.

- Использованные **строковые данные** перечислены ниже.
  - В качестве строковых шаблонов были выбраны большие псевдослучайно сгенерированные стоки длиной 20 мегабайт, в которые в зависимости от эксперимента помещали (набор «Гарантированное вхождение») или не помещали (1 и 2 наборы) вхождения искомых шаблонов.
  - Статическими данными для специализации выбрали автоматически генерированные статические строки небольшой длины, а также отдельные осмысленные шаблоны: гиперссылки и расширения файлов.
- В качестве статических **регулярных выражений** были выбраны выражения, распознающие адреса электронной почты (одно распознаёт более слабо, другое более строго и допускает меньше некорректных адресов), а также номера банковских карт.

  Основной особенностью является представление регулярного вы-

ражения в виде разреженного графа, что, по предположению, может привести алгоритм к более линейной структуре, лучше поддающейся специализации. Регулярные выражения были преобразованы в соответствующие им минимизированные детерминированные конечные автоматы в виде матрицы, представленной в формате COO.

Выбранный набор псевдослучайных регулярных выражений (как и в случае со строковыми данными) вместе с широко используемыми строками и регулярными выражениями позволяет достаточно полно и с оглядкой на практическое применение провести эксперименты по специализации строковых алгоритмов.

3. Структура экспериментального исследования

В рамках данной работы определён следующий круг вопросов, ответы на которые должны быть получены в результате экспериментов.

Q1: Как изменятся время исполнения кода алгоритма, выраженного на языке линейной алгебры, после его специализации с выбранным инструментом (с учётом погрешности)?

Q2: Как время исполнения алгоритма после специализации кода выбранным инструментом соотносятся со временем исполнения этих алгоритмов на других используемых в индустрии и науке инструментах (с учётом погрешности)?

3.1. Инструмент для специализации

Для проведения дальнейших исследований специализации алгоритмов необходимо выбрать специализатор, обладающий перечисленными ниже свойствами.

- **Быстродействие специализированной программы**. Основным назначением специализации в рамках данной работы выступает оптимизация программы по времени исполнения, поэтому необходимо по возможности минимизировать накладные расходы в коде итоговой программы, производимые специализатором.

- **Простота использования**. Для реализации поставленных задач требуется провести большое количество экспериментов. Поэтому желательно сделать управление специализацией как можно более простым для программиста.

- **Возможность создания пользовательской библиотеки**. Одним из основных направлений дальнейшей работы является разработка библиотеки специализированной линейной алгебры, а так-

же строковых алгоритмов. Поэтому важно обеспечить совместимость инструмента с данным сценарием.

Были рассмотрены следующие инструменты для специализации:

- LLPE ;
- AnyDSL [15];
- LLVM.mix [25].

По результатам анализа альтернативных вариантов, для решения поставленных задач была выбрана система AnyDSL как оптимальная с точки зрения простоты использования, быстродействия специализированного и возможностей для создания пользовательской библиотеки.

3.2. Алгоритмы для экспериментов

К рассмотрению в качестве кандидатов для проведения экспериментов были выдвинуты алгоритмы:

- Умножение матриц;
- Тензорное произведение (произведение Кронекера);
- Сопоставление строковых шаблонов;
- Сопоставление с регулярным выражением в графовой форме.

По результатам обзора потенциальной пригодности для специализации, а также прикладного применения и значения для дальнейших исследований, все они были выбраны для экспериментального исследования.

3.3. Метрики

Для целей данной работы задан перечисленный ниже набор метрик, позволяющих оценить результат специализации.

- В качестве метрики времени исполнения выбрано время в наносекундах, полученное инструментом Google Benchmark [6].

- В качестве метрики погрешности измерений времени исполнения выбрана погрешность в процентах, измеренная Google Benchmark во время проведения эксперимента.

В формулировках метрик был использован именно инструмент Google Benchmark, поскольку он позволяет многократно проводить измерения для повышения точности результатов, а также получать статистические данные об эксперименте. Все описанные далее эксперименты проводились в режиме Just-In-Time, то есть генерация кода программы со статическими данными проводилась во время исполнения эксперимента. Такое решение позволило включать статические данные непосредственно в специализируемый код, что существенно повысило качество работы специализатора.

Эксперименты проводились на стенде со следующими характеристиками: Intel Core i5-7440HQ (4x 3.8GHz), 16GB RAM, Ubuntu 20.04.

[6]Репозиторий Google Benchmark: `https://github.com/google/benchmark` (Дата обращения 27.04.2021)

4. Реализация алгоритмов

Реализация алгоритмов, исследуемых в рамках экспериментов представлена в репозитории:

`https://github.com/ibalashov24/spec_experiments`
(Дата обращения: 17.05.2021).

Все алгоритмы были реализованы в указанных ниже трёх исполнениях.

1. На языке Impala [15], который является одним из языков (DSL) управления специализацией в фреймворке AnyDSL. Программа на Impala транслирована AnyDSL в промежуточный код LLVM IR, а далее скомпилирована и скомпонована со специально написанным на языке C++ кодом инициализации специализации.

2. Аналогично на языке Impala, однако все конструкции управления специализацией из кода удалены (специализатор не задействуется).

3. С использованием сторонних инструментов: библиотеки SuiteSparse и инструментов e(Grep) в соответствии с их документацией.

Таким образом, стало возможным сравнить время исполнения специализированного и неспециализированного кода, а также времен работы специализированного кода и широко распространённых в науке и индустрии инструментов.

В качестве статических данных (ограничений) при специализации использованы те из выбранных данных, которые обладают небольшим размером (до нескольких десятков килобайтов). Такое ограничение, с одной стороны, позволяет специализатору завершать свою работу за обозримое время, а с другой, – не увеличивать сложность и объем итогового кода до такой степени, что накладные расходы при его исполнении превысят преимущества от специализации.

Отдельно стоит сравнить сложность разработки программ с использованием специализации на AnyDSL со сложностью разработки аналогичных программ на языках общего пользования. В рамках работы

отдельно от основных экспериментов были реализованы все протестированные алгоритмы на языке Си без использования каких-либо дополнительных инструментов. При разработке эффективного кода для специализатора необходимо было максимально часто использовать статические переменные, по возможности стараясь составлять операторы программы исключительно с их использованием. Часто с точки зрения специализации эффективнее оказалось применять вместо конструкций императивного программирования конструкции функционального, например, вместо циклов использовать рекурсию (обычно хвостовую). Также, необходимо было вручную размечать код конструкциями управления специализатора и, используя профессиональный опыт, пытаться предугадывать итоговое направление специализации — в специализаторах часто используются жадные алгоритмы и эвристики. Таким образом, программирование со специализатором AnyDSL отлично от программирования на языках общего назначения (на примере Си) необходимостью специальной подготовки программиста в области специализации, а также наличия некоторого опыта работы с функциональными языками программирования и некоторой профессиональной интуиции в целом.

Время, нс. (Spec No Spec SuiteSparse) $\Delta < 0.01\%$	$\times$ *eye3*	$\times$ *2blocks*	$\times$ *cover*	$\times$ *mycielskian3*	$\times$ *trec5*
bcsstk16 $\times$	93608 121855 2270	133434 157850 7064	364772 4842889 8559	171085 2129094 511	308535 5226893 505
fs_183_1 $\times$	7796 6752 2553	20187 42353 12310	6928 38250 9796	1358 15194 506	6078 42493 507
can_256 $\times$	1016 1177 2259	5106 38221 6549	20339 66987 9409	2561 23105 503	9548 62668 506

Таблица 2: Сравнение времени исполнения специализированного, не специализированного и кода, реализованного с библиотекой SuiteSparse, в задаче *умножения матриц.*

5. Анализ результатов экспериментов

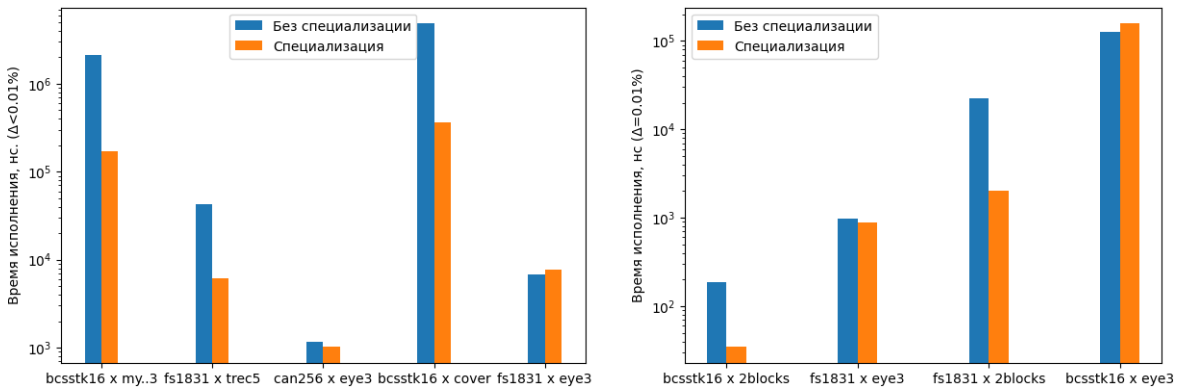
В данном разделе будет представлен анализ полученных в результате выполнения экспериментов результатов к рамках круга установленных исследовательских вопросов.

5.1. Время исполнения после специализации

Далее будут даны ответы на первый поставленный вопрос исследования: сравнение времени исполнения специализированного и неспециализированного кода.

Время, нс. (Spec No Spec SuiteSparse) $\Delta < 0.01\%$	$\otimes$ *eye3*	$\otimes$ *2blocks*	$\otimes$ *cover*	$\otimes$ *mycielskian3*	$\otimes$ *trec5*
bcsstk16 $\otimes$	140628 140744 901878	276222 3032308 2145104	433397 4307538 4420688	276433 1967189 2958016	481805 4571625 1440326
fs_183_1 $\otimes$	916 934 25833	2186 21272 45159	3046 31732 88847	1838 14533 35109	3146 34356 47912
can_256 $\otimes$	1159 1069 35162	2772 30841 60600	4512 45731 130084	2736 22079 43479	4576 49512 61500

Таблица 3: Сравнение времени исполнения специализированного, не специализированного и кода, реализованного с библиотекой SuiteSparse, в задаче произведения Кронекера (тензорного).



Произведение матриц Тезнорное произведение

Рис. 2: Сравнение времени исполнения обычных и специализированных версий алгоритмов тензорного и попарного произведения разреженных матриц.

5.1.1. Умножение матриц

В таблице 2 показаны результаты измерений времени исполнения умножения разреженных матриц без специализации и с ней (на Графике 2 сравнены наиболее интересные, на взгляд автора работы, случаи). Как видно, специализация даёт кратный выигрыш во времени исполнения (от 1.5) раз в большинстве проведенных тестов. Отдельно стоит рассмотреть кратные случаи: в случае *bcsstk16* x *2blocks* обе матрицы имеют сконцентрированные у главной диагонали ненулевые элементы, что приводит к низкому эффекту специализации. В случаях *fs1831* x *eye3* и *can_ 256* x *eye3* низкие результаты специализации получены за счёт большого показателя симметрии у обоих матриц. В обоих вырожденных случаях внутренний цикл кода алгоритма, отвечающий за сложение попарных произведений элементов строки и столбца умножаемых матриц и использующий статические данные, вырождается либо в единственное исполнение тела цикла, либо в единоразовое исполнение для симметрично-расположенных элементов, что не позволяет получить значимое ускорение в данных тестах.

Можно также отметить, что данные результаты оказались существенно лучше, чем в результате проведения схожего эксперимента в

рамках предыдущей работы [17], что свидетельствует о более высоком качестве специализации у AnyDSL, нежели у LLVM.mix.

5.1.2. Тензорное произведение

В таблице 3 показаны результаты измерений времени исполнения произведения Кронекера разреженных матриц без специализации и с ней. Можно заметить, что специализация даёт выигрыш во времени исполнения в среднем в 10 раз в большинстве проведенных тестов. Вырожденные случаи в данном эксперименте оказались аналогичными эксперименту по умножению матриц: *bcsstk16* x *2blocks*, *fs1831* x *eye3* и *fs1831* x *eye3*. В силу схожести структуры алгоритмов произведения Кронекера и обычного умножения матриц, на отсутствие эффекта от специализации снова повлияли сконцентрированность ненулевых элементов матриц и диагонали, а также высокие показатели симметрии у обоих сомножителей.

5.1.3. Сопоставление шаблонов

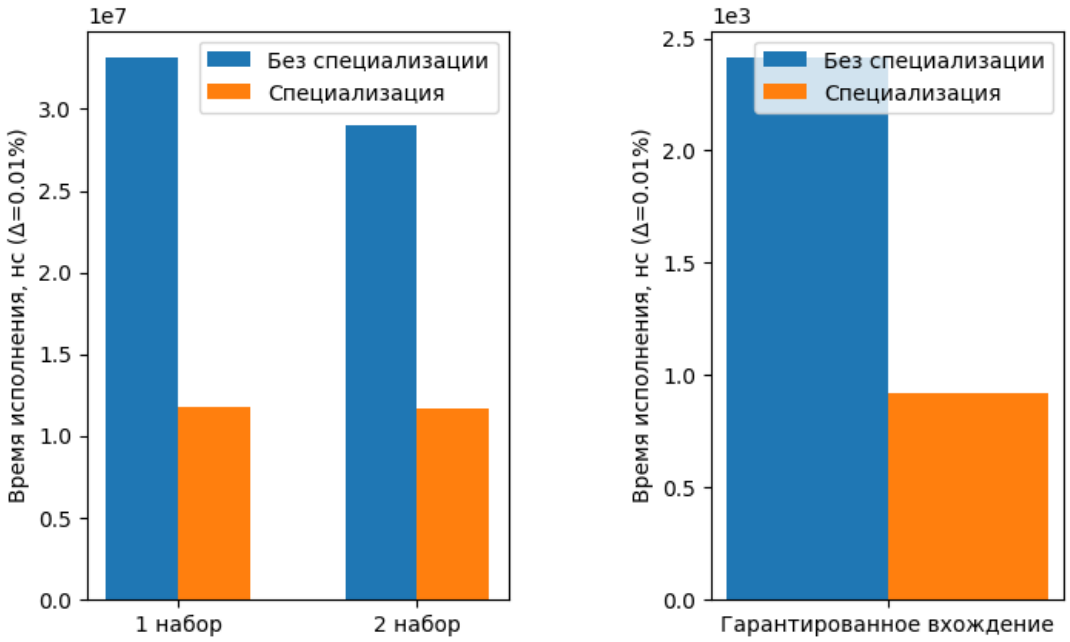


Рис. 3: Измерения времени сопоставления шаблонов на Impala.

На рисунке 3 показаны результаты измерений времени исполнения данного эксперимента. Наблюдается прирост производительности около трёх раз на исследованных шаблонах, что свидетельствует о высокой эффективности специализации в данном тесте. Примерно одинаковый множитель прироста производительности в тесте объясняется применяемой специализатором техникой: разворачивание циклов, для которой в данном алгоритме не важно наличие вхождения в строке.

5.1.4. Сопоставление с регулярным выражением в матричной форме

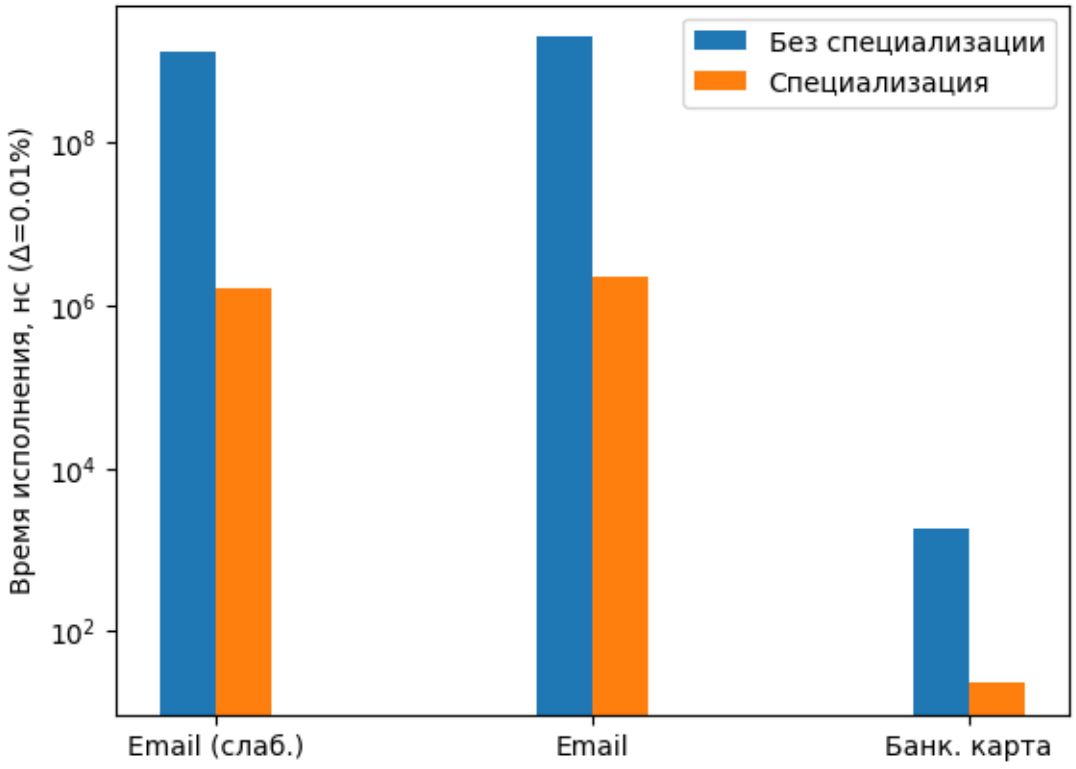


Рис. 4: Измерения времени сопоставления с регулярным выражением в матричной форме.

На рисунке 4 показаны результаты измерений производительности. Как можно видеть из графиков, применение техники специализации даёт уменьшение времени исполнения около трёх порядков (1000 раз)

на всех проведённых тестах. По результатам анализа промежуточного кода LLVM IR, сгенерированного специализатором, был сделан вывод, что такой результат был достигнут посредством применения AnyDSL перечисленных ниже оптимизаций.

- Предвычисление выражений со статическими данными.
- Разбиение графовой структуры COO на набор констант с их последующей подстановкой.
- Агрессивное разворачивание циклов и условных выражений с использованием статических данных.

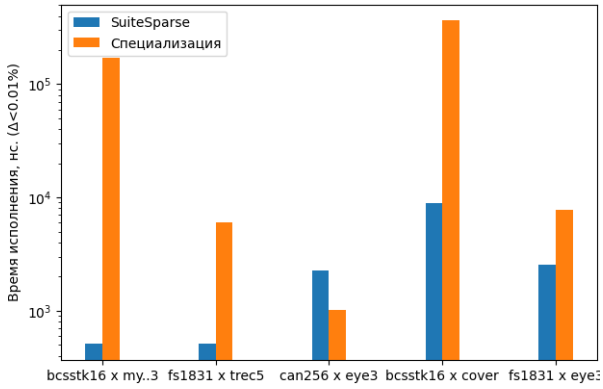
Побочным эффектом применения специализатора стало кратное разрастание объёма исполняемого кода. К примеру, при использовании регулярного выражения, слабо распознающего электронную почту, минимизированный детерминированный автомат которого имеет шесть состояний и около 350 переходов, объём промежуточного кода LLVM IR после обработки специализатором вырос более чем в 20 раз.

5.2. Сравнение по времени исполнения с аналогами

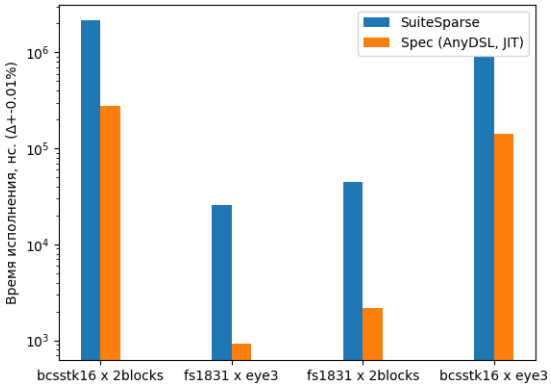
В данной главе в рамках второго исследовательского вопроса, поставленного в работе, будут проанализированы результаты сравнения времени исполнения специализированных версий алгоритмов со временем исполнения тех же алгоритмов, реализованных на широко используемых в индустрии или науке инструментах.

5.2.1. Умножение матриц

Как можно видеть из Таблицы 2, как неспециализированная, так и специализированная версия кода проигрывают по времени исполнения коду на распространённой в научной среде библиотеке SuiteSparse [12] в среднем в 10 раз (сравнение наиболее интересных, на взгляд автора работы, случаев представлен на Графике 5). Такой результат был получен



Произведение матриц



Тезнорное произведение

Рис. 5: Сравнение времени исполнения кода алгоритмов тензорного и попарного произведения матриц со временем исполнения аналогичных алгоритмов на библиотеке SuiteSparse.

за счёт использования SuiteSparse одной наиболее быстрых из известных на момент написания работы версии алгоритма умножения. В свою очередь, специализатору AnyDSL не хватает выразительной мощности для приближения времени вычисления кода к канонической. Тем не менее, такой результат можно назвать хорошим для полуавтоматического инструмента (SuiteSparse написана и оптимизирована полностью вручную). Потенциально он может получить применение в индустрии, учитывая существенные трудозатраты программистов на ручное создание сложных оптимизированных библиотек, таких как SuiteSparse.

5.2.2. Тензорное произведение

В противовес алгоритму умножения матриц, специализация позволила получить выигрыш в среднем на порядок в сравнении со SuiteSparse для алгоритма произведения Кронекера (тензорного), что можно видеть из Таблицы 3. Этот результат можно объяснить относительной простотой структуры алгоритма, а также тем, что SuiteSparse вычисляет тензорное произведение на абстрактных полукольцах и типах данных. Соответственно, накладные расходы на абстракцию существенно замедляют алгоритм, в то время как AnyDSL все абстракции успешно

подставляет в код.

5.2.3. Сопоставление шаблонов

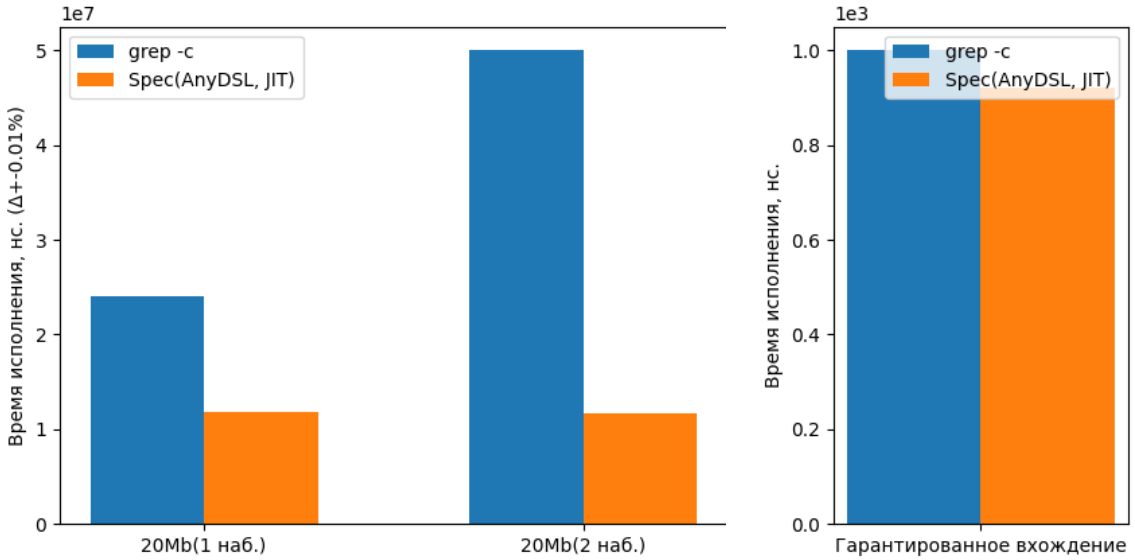


Рис. 6: Сравнение времени исполнения специализированного алгоритма сопоставления шаблонов и Grep.

Как видно из Графика 6, специализированный алгоритм сопоставления шаблонов кратно выигрывает у широко используемого в индустрии инструмента Grep. Ускорение достигнуто во многом благодаря широкому применению специализатором оптимизаций разворачивания циклов и подстановки констант (литералов). Подобные оптимизации позволяют специализированному алгоритмы пропускать излишние итерации цикла для небольших, гарантированно невходящих в шаблон подстрок, чем на момент написания работы не может похвастаться Grep.

5.2.4. Сопоставление с регулярным выражением

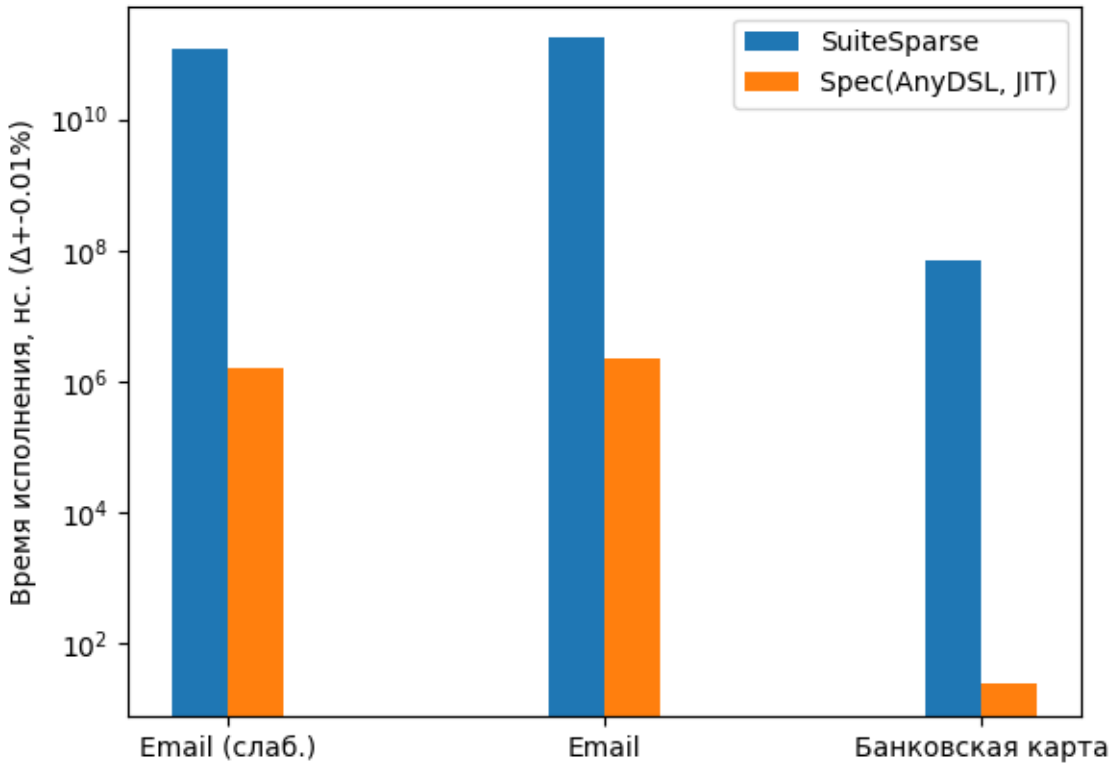


Рис. 7: Сравнение времени исполнения специализированного алгоритма сопоставления с регулярным выражением (в графовой форме) и eGrep.

Из Графика 7 можно видеть, что алгоритм сопоставления с регулярным выражением с данными в графовой форме, как и в случае с сопоставлением шаблонов, выигрывает у широко используемого в индустрии eGrep в несколько раз. Причины аналогичны случаю сопоставления шаблонов — более агрессивное использование специализатором оптимизаций, в то время как eGrep выполняет процедуру сопоставления практически без выполнения каких-либо оптимизаций.

6. Заключение

Таким образом, была исследована применимость техники специализации к графовым алгоритмам, выразимым на языке линейной алгебры, а также строковым алгоритмам.

В рамках данной работы были достигнуты следующие результаты.

1. Выполнен обзор техник и инструментов специализации AnyDSL, LLVM.mix, LLPE. Рассмотрены алгоритмы и наборы данных для экспериментов Harwell-Boeing, SuiteSparse Matrix Collection. Сделан обзор стандарта GraphBLAS, его специализации, а также теоретических основ специализации строковых алгоритмов.

2. Спроектировано экспериментальное исследование по специализации алгоритмов. Целью эксперимента задано исследование времени исполнения кода алгоритмов и его сравнения с результатами для эталонных реализаций SuiteSparse и (e)Grep.

3. Выбраны и реализованы следующие матричные алгоритмы: произведение матриц, произведение Кронекера, сопоставление шаблонов, сопоставление с автоматом, поиск в ширину, поиск кратчайшего пути в графе. В качестве инструмента специализации выбран инструмент AnyDSL.

4. Проведено экспериментальное исследование на наборе матриц Suite Sparse Matrix Collection и выбранном наборе строк и регулярных выражений. Установлено следующее.

    - Получено ускорение по времени исполнения у всех протестированных алгоритмов, минимально на 10%, максимально на 300% в зависимости от входных данных. Погрешность измерений $\pm 0.01\%$.
    - Получен кратный выигрыш по времени исполнения на строковых алгоритмах по сравнению с (e)Grep. На графовых алгоритмах специализатор AnyDSL проигрывает фреймворку

SuiteSparse в среднем в пять раз, что, однако, можно считать хорошим результатом для полуавтоматического инструмента. Погрешность измерений $\pm 0.01\%$.

Список литературы

[1] Jones Neil D., Gomard Carsten K., Sestoft Peter. Partial Evaluation and Automatic Program Generation. — Prentice Hall International, 1993. — ISBN: 0-13-020249-5.

[2] RaTrace: simple and efficient abstractions for BVH ray traversal algorithms / Arsène Pérard-Gayot, Martin Weier, Richard Membarth et al. // Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences. — 2017. — P. 157–168.

[3] AnySeq: a high performance sequence alignment library based on partial evaluation / André Müller, Bertil Schmidt, Andreas Hildebrandt et al. // 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS) / IEEE. — 2020. — P. 1030–1040.

[4] Rodent: generating renderers without writing a generator / Arsène Pérard-Gayot, Richard Membarth, Roland Leißa et al. // ACM Transactions on Graphics (TOG). — 2019. — Vol. 38, no. 4. — P. 1–12.

[5] Kepner Jeremy, Gilbert John. Graph algorithms in the language of linear algebra. — SIAM, 2011.

[6] Azimov Rustam, Grigorev Semyon. Context-free path querying by matrix multiplication // Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). — 2018. — P. 1–10.

[7] Aydın Bulu¸c Timothy Mattson Scott McMillan Jos´e Moreira Carl Yang. The GraphBLAS C API Specification. Version 1.3.0. — 2019. — Access mode: `http://people.eecs.berkeley.edu/~aydin/GraphBLAS_API_C_v13.pdf` (online; accessed: 14.12.2020).

[8] Stephen Graham A. String searching algorithms. — World Scientific, 1994.

[9] Introduction to algorithms / Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein. — MIT press, 2009.

[10] Kepner Jeremy, Gilbert John. Graph Algorithms in the Language of Linear Algebra / Ed. by Jeremy Kepner, John Gilbert. — Society for Industrial and Applied Mathematics, 2011.

[11] Lothaire M. Applied Combinatorics on Words. Encyclopedia of Mathematics and its Applications. — Cambridge University Press, 2005.

[12] Davis Timothy A. Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra // ACM Trans. Math. Softw. — 2019. — Dec. — Vol. 45, no. 4.

[13] Ershov A.P. Mixed computation: potential applications and problems for study // Theoretical Computer Science. — 1982. — Vol. 18, no. 1. — P. 41 – 67.

[14] Christensen Niels H., Glück Robert. Offline Partial Evaluation Can Be as Accurate as Online Partial Evaluation // ACM Trans. Program. Lang. Syst. — 2004. — Jan. — Vol. 26, no. 1. — P. 191–220. — Access mode: https://doi.org/10.1145/963778.963784.

[15] AnyDSL: A Partial Evaluation Framework for Programming High-performance Libraries / Roland Lei, Klaas Boesche, Sebastian Hack et al. // Proc. ACM Program. Lang. — 2018. — Oct. — Vol. 2, no. OOPSLA. — P. 119:1–119:30.

[16] Eugene Sharygin. Multi-stage compiler-assisted specializer generator built on LLVM, FOSDEM 2019. — Access mode: https://archive.fosdem.org/2019/schedule/event/llvm_mix/attachments/slides/3172/export/events/attachments/llvm_mix/slides/3172/llvm_mix.pdf (online; accessed: 13.12.2020).

[17] Балашов И.В. Комплексное исследование эффективности методов программной специализации для графических процессоров // Курсовая работа. — 2020. — Access mode: https://oops.math.spbu.ru/SE/YearlyProjects/vesna-2020/YearlyProjects/vesna-2020/mo-3rd-course/Balashov-report.pdf (online; accessed: 21.12.2020).

[18] Azimov Rustam, Grigorev Semyon. Context-free Path Querying by Matrix Multiplication // Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). — GRADES-NDA '18. — New York, NY, USA : ACM, 2018. — P. 5:1–5:10.

[19] Pattern Matching Techniques and Their Applications to Computational Molecular Biology-A Review : Rep. / Citeseer ; Executor: Eric C Rouchka : 1999.

[20] Sørensen Morten Heine, Glück Robert, Jones Neil D. Towards unifying partial evaluation, deforestation, supercompilation, and GPC // Programming Languages and Systems — ESOP '94 / Ed. by Donald Sannella. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1994. — P. 485–500.

[21] Consel Charles, Danvy Olivier. Partial evaluation of pattern matching in strings // Information Processing Letters. — 1989. — Vol. 30, no. 2. — P. 79–86.

[22] Algorithms to accelerate multiple regular expressions matching for deep packet inspection / Sailesh Kumar, Sarang Dharmapurikar, Fang Yu et al. // ACM SIGCOMM Computer Communication Review. — 2006. — Vol. 36, no. 4. — P. 339–350.

[23] Bird Steven, Klein Ewan. Regular expressions for natural language processing // University of Pennsylvania. — 2006.

[24] Davis Timothy A, Hu Yifan. The University of Florida sparse matrix collection // ACM Transactions on Mathematical Software (TOMS). — 2011. — Vol. 38, no. 1. — P. 1–25.

[25] E.Yu.Sharygin, R.A.Butchatskiy R.A.Zhuykov, A.R.Sher. Query compilation in PostgreSQL by specialization of the DBMS source code // Programming and Computer Software. — 2017. — P. 353–365.