Исследование эффективности применения специализации для алгоритмов, основанных на методах линейной алгебры

Илья Вадимович Балашов

Научный руководитель: ст.преп., к.ф.-м.н. Д.А.Березун
Консультант: доц. каф. информатики, к.ф.-м.н. С.В.Григорьев
Рецензент: программист ООО “ИнтеллиДжей Лабс” М.Х.Ахин

Введение: специализация и библиотеки

- Крупные программы и библиотеки часто имеют базовые части, на которые опираются существенные части этих программ или библиотек
  - Умножение матриц для BLAS
  - Сопоставление шаблонов и автоматов в библиотеках операций над строками
- Оптимизируем базовые части — оптимизируем большие части библиотек или крупных программ
- Оптимальные алгоритмы вручную писать сложно
  - Нужны инструменты-помощники
- Специализация — метод автоматической оптимизации программ

Цели и постановка задач

Цель — исследование эффективности специализации для базовых матричных и строковых алгоритмов в матричной форме, широко используемых в некоторых областях.

1. Выполнить обзор предметной области: специализации и существующих инструментов для её проведения, а также алгоритмов и наборов данных из различных областей, к которым специализация применима
2. Спроектировать экспериментальное исследование специализации выбранных алгоритмов
3. Реализовать алгоритмы, выбранные для экспериментов
4. Выполнить эксперименты и проанализировать результаты

Обзор: специализация

- Метод агрессивной оптимизации программ
  - Осуществляется специализатором
- $[\![F]\!][a, b] = [\![F_b]\!][a]$
- Долгая специализация, но быстрое исполнение программы
  - Имеет смысл при многократном исполнении
- В качестве инструмента выбран AnyDSL
  - DSL Impala и Artic
  - Апробирован в разных задачах:
    - ★ Обработка изображений
    - ★ Биоинформатика
    - ★ Трассировка лучей
  - Конкуренты (LLPE, LLVM.mix) не подошли по применимости

Обзор: алгоритмы

- Примитивы GraphBLAS
  - Умножение матриц
  - Произведение Кронекера
- Алгоритмы на строках
  - Сопоставление шаблонов
  - Сопоставление с регулярным выражением с матричными данными
- Широко используются в науке и индустрии
  - GraphBLAS
  - КМП-тест
  - Утилиты семейства Grep

Обзор: наборы данных

- GraphBLAS и графовые алгоритмы в матричной форме
  - Набор матриц Harwell-Boeing
  - Набор SuiteSparse Matrix Collection
- Алгоритмы на строках
  - Сгенерированные строки
  - Образцы трафика
  - Регулярные выражения из каталога `regexlib.com`
- Выбраны наборы данных с разнообразной конфигурацией
  - Покрытие большого количества базовых сценариев
  - Вырожденные случаи
  - Претензия на полноту $\Longrightarrow$ повышение качества экспериментов

Структура экспериментов

Вопросы для исследования:

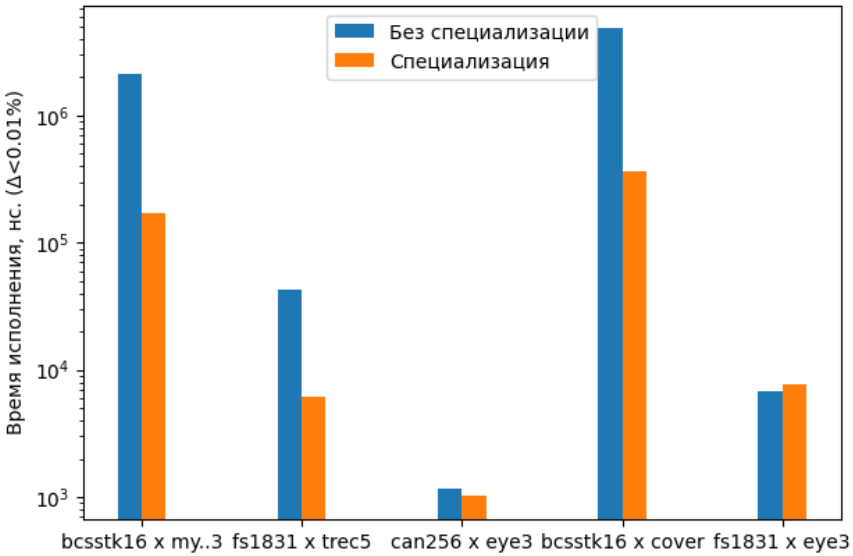
Q1 Как изменится время исполнения кода алгоритмов после их специализации AnyDSL?

Q2 Как время исполнения кода на AnyDSL соотносится со временем исполнения кода тех же алгоритмов в эталонных инструментах?

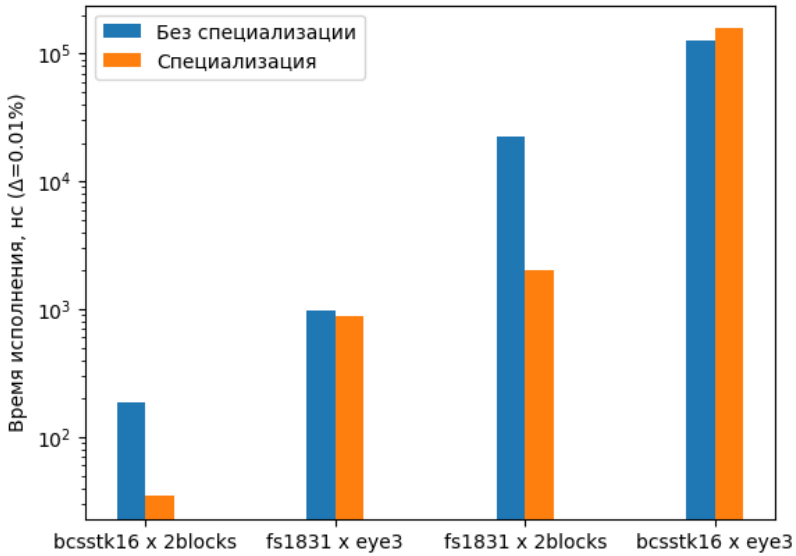
Тестовый стенд:

- Intel i5-7440HQ, 16GB RAM, Ubuntu 20.04
- Google Benchmark
- Будут показаны наиболее интересные случаи

Эксперименты: матрицы

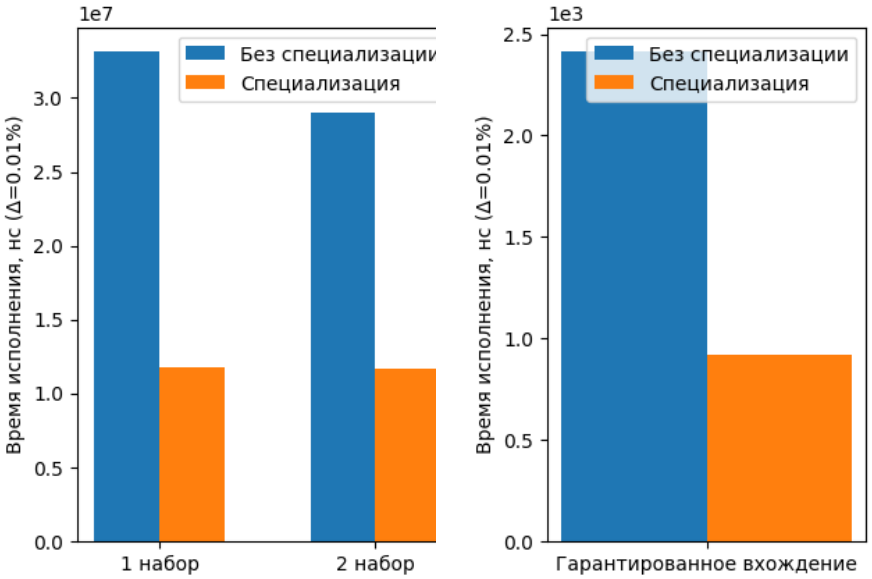


Умножение матриц

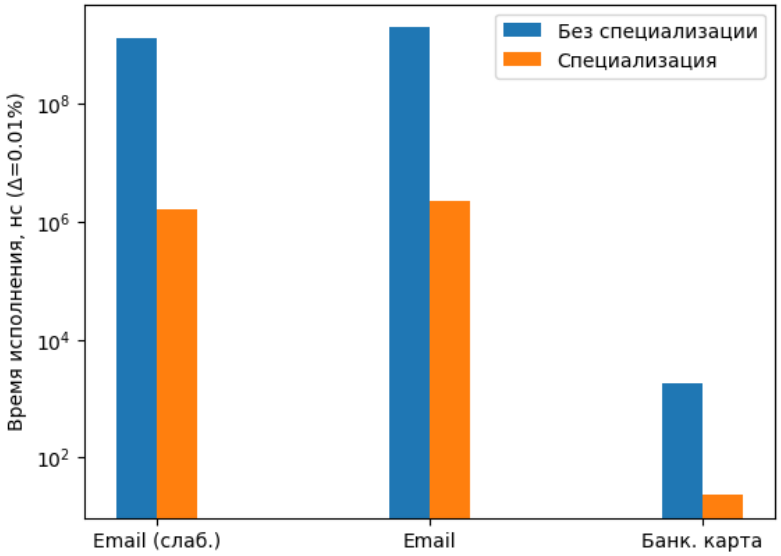


Тензорное произведение

Эксперименты: шаблоны

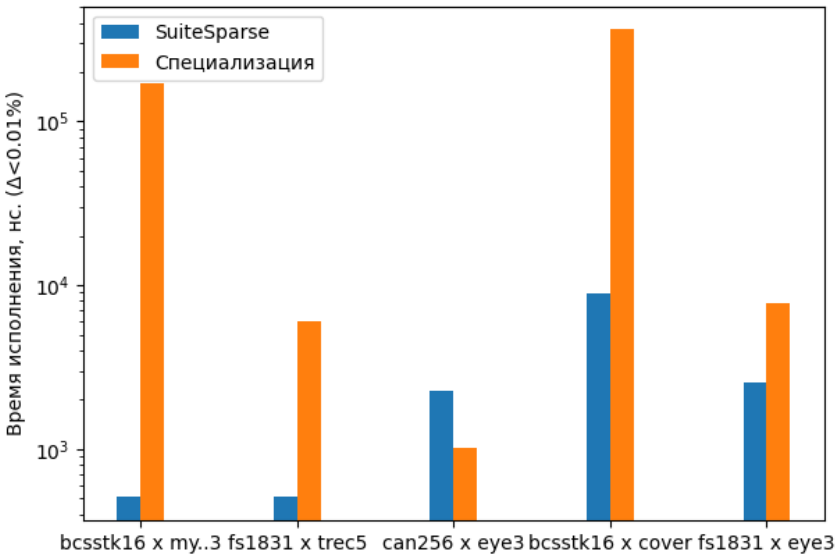


Поиск подстроки

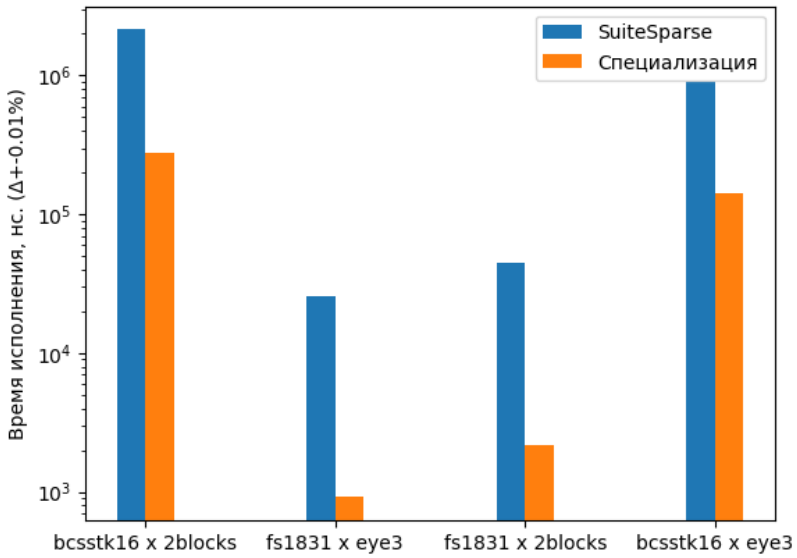


Регулярное выражение

Сравнение со SuiteSparse GraphBLAS



Умножение матриц

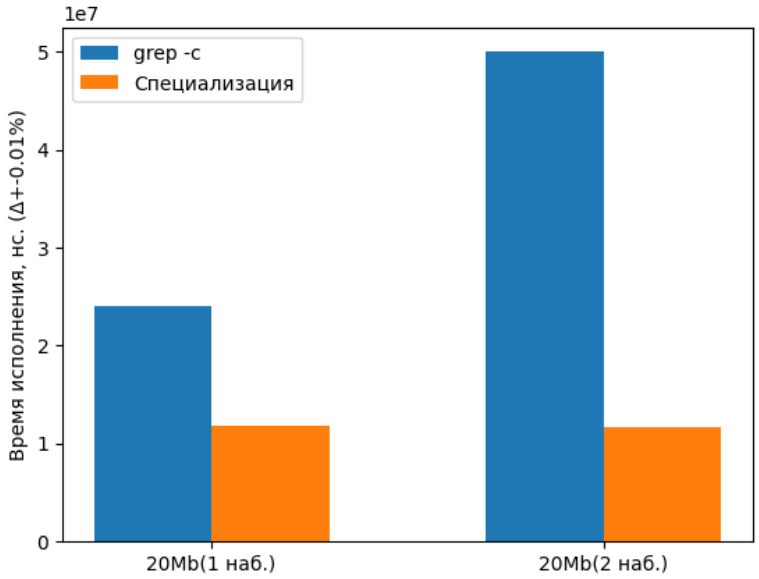


Тензорное произведение

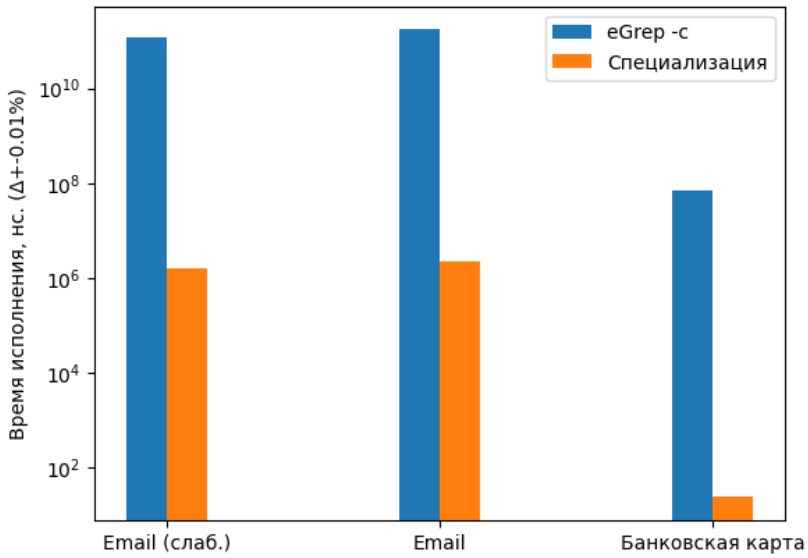
Есть проигрыш, но

- Полуавтоматический инструмент
- Выигрываем в 10+ раз у обычного алгоритма
- Оптимизация во время компиляции — нет нужды в библиотеке
- Можно использовать для различных модификаций алгоритма

Сравнение с (e)Grep



Поиск подстроки



Регулярное выражение

Результаты

- Выполнен обзор предметной области: специализации, инструментов, алгоритмов и наборов данных
- Спроектировано экспериментальное исследование по специализации на базе инструмента AnyDSL
- Выбраны и реализованы матричные и строковые алгоритмы для экспериментов
- Проанализированы результаты эксперимента в рамках круга поставленных вопросов $\pm 0.01\%$
  - Ускорение всех протестированных алгоритмов, от 10% до 100 раз.
  - Выигрыш от 2–5 до трёх порядков по времени исполнения на строковых алгоритмах по сравнению с (e)Grep
  - Тензорное произведение — выигрыш до 10 раз в сравнении со SuiteSparse
  - Проигрыш SuiteSparse на алгоритме умножения матриц около 5 раз — хороший результат

Дополнительно: что считали статическим

- Относительно небольшие данные
  - До нескольких сотен байтов
  - Разнообразная структура
  - Слишком большое ограничение $\Longrightarrow$ большие накладные расходы
- Для матриц
  - Размер матриц до 10
  - Ненулевых элементов до 20 штук
- Для строк
  - Строки длиной до 200 символов
  - Несложные регулярные выражения

Дополнительно: вопросы про JIT

- Под JIT подразумевается генерация, специализация и компиляция эксперимента в момент его запуска
  1. Кодогенерация — встраивания статических данных в код на Impala
  2. Специализация через AnyDSL
  3. JIT-компиляция через LLVM

- В тексте работы результаты даны *без* учёта времени JIT
  - Главное — многократное исполнение
  - На проведённых тестах время JIT пренебрежительно мало при многократном исполнении
  - Незаметно при от одного до 10^6 повторений

Дополнительно: сравнение по трудоёмкости разработки

- Простота использования — возожность относительно несложно и прямолинейно решать большинство *прикладных* задач после единократной подготовки программиста

- Нужен некоторый опыт (промышленного) программирования

- LLVM.mix < AnyDSL < LLPE << Экспериментальные Системы
  - LLVM.mix — только разметка кода C++ атрибутами
  - AnyDSL — Rust-like DSL, особые языковые конструкции, атрибуты
  - LLPE — программирование в конструкциях LLVM SDK
  - Э.С. — абстрактные ЯП, нужен особый опыт, теоретическая подготовка

Дополнительно: насколько глубокая специализация?

- Безусловно, оптимизация не идеальная
- Использовались все доступные возможности Impala DSL
- Возможностей в бэкенде AnyDSL больше, чем реализовано во фронтенде Impala
- Имеется Artic DSL
  - Поддержка новых возможностей и оптимизаций AnyDSL
  - Фронтенд ещё «сырой» — нужен существенный опыт
  - Возможное направление будущей работы

Дополнительно: список матриц с характеристиками

	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

Матрицы, использованные в экспериментах по специализации.

Дополнительно: произведение матриц, полные числа

Время, нс. Spec/NoSpec/SP $\Delta < 0.01\%$	$\times$ *eye3*	$\times$ *2blocks*	$\times$ *cover*	$\times$ *my...3*	$\times$ *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

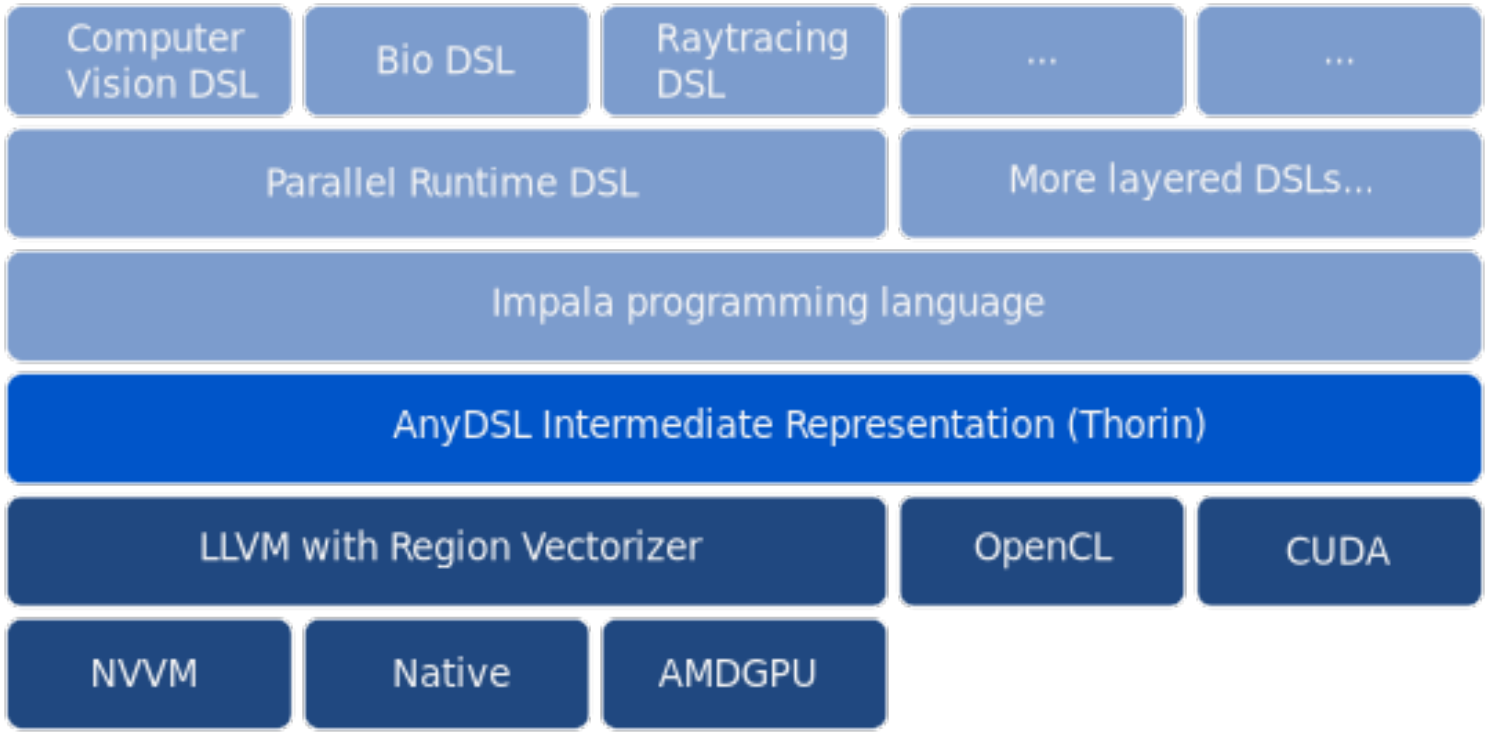
Время исполнения в эксперименте для произведения матриц

Дополнительно: тензорное произведение, полные числа

Время, нс. Spec/NoSpec/SP $\Delta < 0.01\%$	$\otimes$ *eye3*	$\otimes$ *2blocks*	$\otimes$ *cover*	$\otimes$ *my...3*	$\otimes$ *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

Время исполнения в эксперименте для тензорного произведения

Дополнительно: структура AnyDSL



Структура AnyDSL

Источник: https://anydsl.github.io/

Дополнительно: сравнение с чистым Си