Санкт-Петербургский государственный университет

Тюляндин Иван Владимирович

Выпускная квалификационная работа

Исследование применимости специализации алгоритма Витерби скрытой марковской моделью

Уровень образования: магистратура

Направление *09.04.04 «Программная инженерия»*

Основная образовательная программа *ВМ.5666.2019 «Программная инженерия»*

Научный руководитель:
к.ф.-м.н., доцент кафедры информатики
С. В. Григорьев

Консультант:
к.ф.-м.н., старший преподаватель кафедры МКН СПбГУ
Д. А. Березун

Рецензент:
старший преподаватель, Санкт-Петербургский Политехнический Университет
М. Х. Ахин

Санкт-Петербург
2021

Saint Petersburg State University

Ivan Tyulyandin

Master's Thesis

Viterbi algorithm specialization with hidden Markov model

Education level: master

Speciality *09.04.04 «Software Engineering»*

Programme *BM.5666.2019 «Software Engineering»*

Scientific supervisor:
C.Sc., docent
S. V. Grigorev

Consultant:
senior lecturer at Department of Mathematics and Computer Science of SbPU, C.Sc.
D. A. Berezun

Reviewer:
senior lecturer at Peter the Great St.Petersburg Polytechnic University
M. H. Akhin

Saint Petersburg
2021

Оглавление

Введение

Количество доступной информации в современном мире стремительно растет. С целью повышения скорости обработки информации используются различные подходы, такие как вычислительные кластеры, новые алгоритмы и дополнительное оборудование, например, FPGA (ПЛИС) или GPGPU (графический процессор общего назначения). Однако на практике не всегда есть возможность увеличить имеющиеся вычислительные мощности, и в таком случае необходимо искать способы улучшения алгоритмов.

Одним из таких способов является описание и реализация отдельных шагов существующих алгоритмов с использованием иных понятий и формализмов. Для многих алгоритмов отдельные шаги могут быть описаны с использованием методов линейной алгебры, например, через операции над матрицами с переопределёнными операциями поэлементного сложения и умножения. Если описать алгоритм методами линейной алгебры и запрограммировать его с использованием промышленных библиотек, то он может значительно превзойти по скорости исходную реализацию за счет возможности эффективно распараллеливания матричных операций. Алгоритмы, которые выражены через операции и понятия из линейной алгебры, широко используются в различных областях, таких как машинное обучение [1], компьютерное зрение [19], статистика [8], анализ программ на логических языках программирования [13], теория графов [3] и многих других.

Другой способ улучшения алгоритмов основан на следующем наблюдении. Достаточно часто случается ситуация, когда часть параметров алгоритма не меняется от запуска к запуску, т.е. они зафиксированы в течение некоторого значительного промежутка времени. Зная фактические значения этих параметров, можно оптимизировать их использование в алгоритме. Таким образом, можно получить новый алгоритм, в котором вычисления, зависящие только от зафиксированных параметров, уже выполнены. Результат выполнения нового алгоритма с оставшимися параметрами должен быть семантически эквивалентен

результату выполнения исходного алгоритма на соответствующих данных. В сравнении с исходным алгоритмом, при повторных запусках новый алгоритм не выполняет те вычисления, которые зависят от зафиксированных параметров. Эти вычисления сделаны и сохранены на стадии генерации нового алгоритма. Такая техника преобразования алгоритмов известна как “специализация”, или “частичное вычисление”, а программа, генерирующая новый алгоритм, называется “специализатор” [11]. На текущий момент вопрос о границах применимости специализации к алгоритмам, выраженных с помощью методов линейной алгебры, до конца не исследован.

В данной работе будет рассмотрена специализация известного алгоритма Витерби [23], выраженного в терминах линейной алгебры [20]. Этот алгоритм используется в биоинформатике [10], при распознавании речи [17] и в финансовых расчетах [14]. У него имеется два входных параметра: скрытая марковская модель (далее — СММ) [6] и последовательность наблюдений. Задачей алгоритма Витерби является вычислить вероятность того, что последовательность наблюдений была сгенерирована именно с помощью данной СММ. Основная часть алгоритма Витерби существенно зависит от СММ, и в то же время на практике, как правило, используется какая-то одна марковская модель для анализа значительного количества последовательностей наблюдений. Следовательно, если специализировать алгоритм Витерби скрытой марковской моделью, то это может дать значительный прирост производительности.

1. Постановка задачи

Целью данной работы является исследовать применимость специализации к алгоритму Витерби, который описан методами линейной алгебры, с предположением, что скрытая марковская модель является зафиксированным параметром. Для достижения этой цели были поставлены следующие задачи:

- сделать обзор предметной области:
  - рассмотреть алгоритм Витерби и его существующие реализации;
  - описать технику специализации;
- разработать специализированный алгоритм Витерби, реализовать и протестировать корректность реализации;
- провести эксперименты по сравнению производительности специализированного алгоритма с неспециализированной версией и существующими реализациями.

2. Обзор

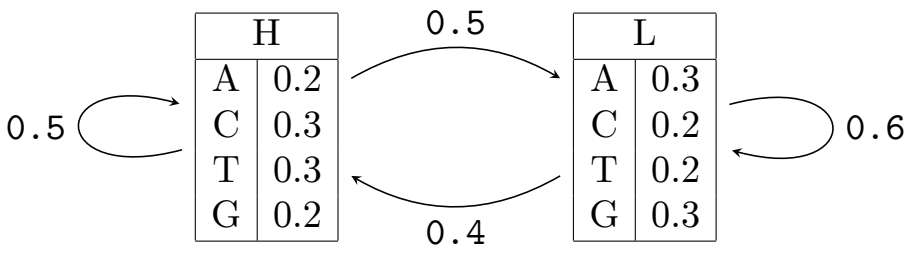
В главе выполнен обзор предметной области. Описаны скрытые марковские модели и алгоритм Витерби, а также приведены существующие решения, в которых этот алгоритм реализован для обработки большого объёма данных. В заключительном разделе обзора рассмотрена техника специализации.

2.1. Скрытые марковские модели

Скрытая марковская модель (СММ) [6] является дискретным вероятностным автоматом, каждое состояние которого может с определённой вероятностью создавать наблюдение. В определении СММ есть следующие параметры:

- $S_{1..N}$ — N состояний автомата;
- $O_{1..K}$ — K возможных наблюдений;
- $B_{1..N}$ — вероятности для каждого состояния из $S_{1..N}$ быть стартовым;
- $T_{1..N,1..N}$ — матрица переходов, $T_{i,j}$ является вероятностью перехода из состояния S_i в состояние S_j;
- $E_{1..N,1..K}$ — матрица наблюдений, где $E_{i,j}$ определяет вероятность создания наблюдения O_j в состоянии S_i.

Пример скрытой марковской модели представлен на рисунке 1. В ней два состояния H и L, и четыре возможных наблюдения A, C, T и G. Вероятности переходов между состояниями указаны на соответствующих стрелках, а вероятности создания определённого наблюдения описаны в состояниях.



Стартовые вероятности (B)

H	L
0.5	0.5

Рис. 1: Пример СММ

2.2. Алгоритм Витерби

Алгоритм Витерби [23] вычисляет для каждого состояния СММ (англ. HMM) максимальную вероятность нахождения в нём, при условии того, что последовательность событий *Obs* была сгенерирована этой СММ.

2.2.1. Описание методами динамического программирования

Алгоритм Витерби может быть описан методами динамического программирования. Псевдокод алгоритма представлен на листинге 2.2.1. Массивы индексируются с единицы до границы включительно. На строках 5-6 происходит обработка первого наблюдения из последовательности *Obs*. Так как это первое наблюдение, то вероятности перехода учитывать не нужно. Вычисления, связанные с оставшейся частью последовательности, выполняются на строках 8-11. Результатом является последняя строка матрицы *Dp*.

```
1  function Viterbi(HMM, Obs)
2    lo = length(Obs)
3    Dp[lo][HMM.N]
4
5    for j = 1..HMM.N
6      Dp[1][j] = HMM.E[j][Obs[1]] * HMM.B[j]
7
8    for i = 2..lo
9      for j = 1..N
10       Dp[i][j] =
```

```
11          max_{x=1..N}(HMM.T[x][j] * HMM.E[j][Obs[i]] * Dp[i−1][x])
12
13  return Dp[lo]
```

Листинг 1: Алгоритм Витерби

2.2.2. Описание методами линейной алгебры

Также алгоритм Витерби может быть выражен матричными операциями из линейной алгебры [20]. Рассмотрим подробнее, как это делается.

Ключевой идеей является использование специальной алгебраической структуры полукольцо *Min-plus*. Элементы полукольца будут описывать вероятности с помощью вещественных чисел. Операция сложения определяется как функция минимума из двух чисел, а операция умножения имеет семантику сложения чисел. Нейтральными элементами по сложению будет $+\infty$ и 0 по умножению соответственно. Ниже приведен пример умножения матрицы на столбец с использованием полукольца *Min-plus*.

$$\begin{pmatrix} 0 & 1 \\ +\infty & 2 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \begin{pmatrix} min(0+3, 1+4) \\ min(+\infty+3, 2+4) \end{pmatrix} = \begin{pmatrix} 3 \\ 6 \end{pmatrix}$$

Для того, чтобы можно было использовать полукольцо *Min-plus*, ко всем вероятностям p в СММ применяется преобразование 1. Это делается для сохранения точности расчетов. Далее такая вероятность будет называться *преобразованной*. Например, преобразованная вероятность 0.5 равна $-1 * log_2(0.5) = 1$.

$$t(p) = \begin{cases} p > 0: & -1 * log_2(p)) \\ p = 0: & +\infty \end{cases} \tag{1}$$

Для каждого события o из множества O определяем диагональную матрицу $P(o)$ размера $N \times N$.

$$P(o) = \begin{pmatrix} t(E[1,o]) & \dots & +\infty \\ \vdots & \ddots & \vdots \\ +\infty & \dots & t(E[N,o]) \end{pmatrix}$$

Начало алгоритма Витерби — это обработка первого наблюдения из последовательности Obs. В столбце B хранятся преобразованные вероятности состояний из СММ быть начальными. Символ $\times$ обозначает умножение матриц с использованием полукольца *Min-plus*:

$$Probs_1 = P(Obs[1]) \times B.$$

Далее вычисляются преобразованные вероятности для всех состояний СММ с учётом оставшихся событий из Obs. Матрица T хранит преобразованные вероятности переходов из состояния в состояние. В отличие от обработки первого наблюдения, далее необходимо учитывать и вероятности перехода, и вероятности создания наблюдения. Обработка происходит следующим образом:

$$Probs_t = P(Obs[t]) \times T^\top \times Probs_{t-1}.$$

После выполнения всех шагов алгоритма, в столбце $Probs_{lo}$, где lo — это длина последовательности Obs, будут находиться преобразованные вероятности быть в определённом состоянии СММ при условии наблюдения последовательности событий Obs. Псевдокод алгоритма приведен на листинге 2.

```
function Viterbi(HMM, Obs)
  lo = length(Obs)
  // Столбец для результатов
  Probs[1][HMM.N]

  // Все матричные умножения выполняются в полукольце Min-plus
  Probs = P(Obs[1]) × HMM.B

  for i = 2..lo
    Probs = P(Obs[i]) × (HMM.T)^⊤ × Probs

  return Probs
```

Листинг 2: Алгоритм Витерби, выраженный методами линейной алгебры

2.3. Существующие реализации алгоритма Витерби

В данном разделе рассмотрены существующие высокопроизводительные реализации алгоритма Витерби, которые используются на практике в бионформатике для решения задачи гомологичности, то есть для определения схожести протеинов. Группы схожих протеинов называются *семейством*, на основе которого можно построить СММ, описывающую общие части протеинов из семейства. Такая СММ называется *профилем*. Все протеины кодируются двадцатью аминокислотами, которые могут быть выражены буквами латинского алфавита. Последовательность, обрабатываемая с помощью профиля, также является закодированным протеином.

Рассмотренные далее реализации алгоритма Витерби выполнены с использованием метода динамического программирования. Этот метод описан в подразделе 2.2.1. Высокопроизводительных реализаций алгоритма Витерби иными методами для запуска на рабочей станции не было найдено.

2.3.1. HMMer

Решение HMMer [9] используется для поиска в базах данных последовательностей гомологов исследуемых протеинов, а также для создания профилей семейств протеинов. Этот проект является открытым (open source project), реализован на языке C с возможностью использовать SIMD-инструкции процессора. Успешно применяется во многих базах данных, таких как Pfam [15].

Авторами проекта были предложены вероятностные *фильтры*. Их применение позволяет ускорить обработку данных из-за уменьшения вычислений в алгоритме Витерби за счет уменьшения количества состояний и переходов в СММ. Один из таких фильтров — MSV (Multiple Segment Viterbi) [7].

2.3.2. CUDAMPF

В проекте CUDAMPF [10] реализованы вероятностные фильтры из существующего решения HMMer с использованием CUDA. Код предназначен для видеокарт NVIDIA с архитектурой Kepler или более новой. Проект рассчитан на определение гомологичности одновременно для множества протеинов.

Авторы предлагают четыре уровня параллелизма. Первые три основаны на логическом параллелизме по данным. Четвертый уровень использует SIMD-инструкции вычислителей видеокарты. Разделение данных по уровням позволило добиться ускорения в 23,1 раз при работе с фильтром MSV по сравнению с HMMer.

Несмотря на то, что авторами заявлена корректность реализации, в исходном коде есть гонка данных при вычислении одного из состояний при обработке MSV. В этом состоянии хранится максимум из определённого множества состояний. В коде CUDAMPF переменная, хранящая максимум, не защищена от одновременной записи двумя или более потоками.

2.4. Специализация

Техника *специализации* (или *частичных вычислений*) предназначена для преобразования программ, у которых часть входных параметров известна и зафиксирована [11] с целью оптимизации производительности. Типичным случаем для специализации является последовательное применение программы для обработки данных, часть из которых не меняется от запуска к запуску. Согласно определениям, которые приняты для специализации, зафиксированные параметры называются *статическими*, а все остальные параметры — *динамическими*. Цель применения специализации — уменьшить количество вычислений, которые зависят от статических параметров. Ожидается, что при многочисленных запусках специализированная программа на динамических параметрах будет более производительной чем изначальная версия программы, которая выполняется на статических и динамических параметрах. Рас-

пространённой проблемой на практике является замедление производительности из-за большого объема специализированного кода. Схема специализации приведена на рисунке 2.

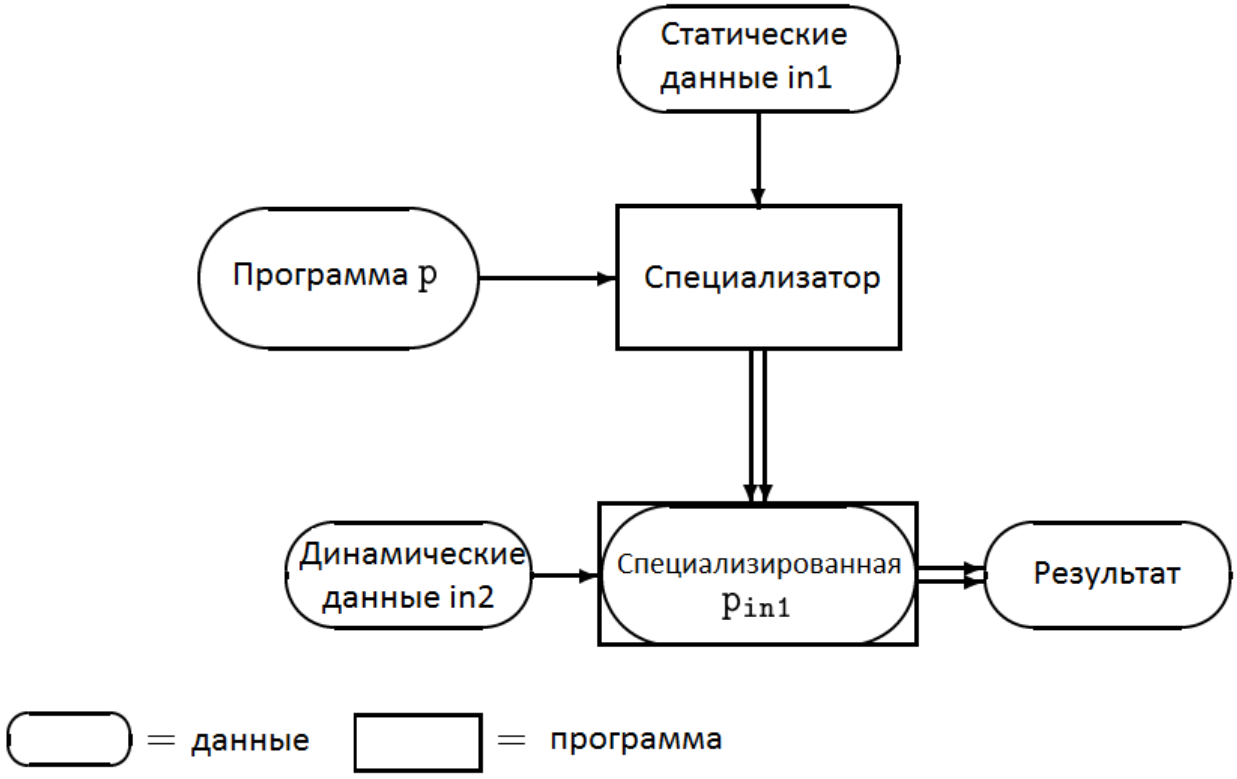


Рис. 2: Концептуальная схема специализации[1]

Специализация была успешно применена в обработке графики [2], обработке запросов к базам данных [16] и поиску подстроки в строке на GPGPU [22]. Более подробно о специализации можно узнать в книге Джонса, Гомарда и Сестофта [11].

Специализация может применяться как и к произвольным программам, так и к отдельным реализациям какого-то алгоритма. Независимо от того, что специализируется, создание оптимального специализатора является алгоритмически неразрешимым, это можно доказать через сведение к проблеме останова. В случае специализации какого-то заранее выбранного алгоритма, можно использовать его шаги и структуру для создания более эффективной специализированной версии. Далее

[1]Схема взята из книги [11], и содержание схемы переведено на русский язык.

будет рассмотрен алгоритм создания специализированной версии алгоритма Витерби.

3. Специализация алгоритма Витерби

В этой главе рассказывается о том, как можно выполнить специализацию алгоритма Витерби, выраженного методами линейной алгебры, где СММ является статическим, т.е. зафиксированным параметром. В разделе 3.1 представлен подход специализации алгоритма Витерби, который позволяет существенно сократить количество необходимых матричных операций за счет сохранения части предпосчитанных результатов в памяти. Предложенный подход специализации может быть использован в тех предметных областях, где СММ часто является зафиксированным параметром, и специфика области не вносит изменений в логику алгоритма Витерби.

3.1. Специализация с использованием матричных операций

Вариант представления алгоритма Витерби через матричные операции представлен в [20]. Он предназначен для работы со СММ из раздела 2.1 и рассмотрен в подразделе 2.2.2.

Рассмотрим операции алгоритма Витерби и выделим те части, которые можно специализировать, с учетом того, что СММ — статический параметр. Последовательность наблюдений Obs, которая является динамическим параметром, индексируется от 1 до lo включительно, где lo — это длина последовательности. Начальный шаг — это обработка первого наблюдения из последовательности событий Obs:

$$Probs_1 = P(Obs[1]) \times B.$$

В СММ записано множество возможных наблюдений O. Матрицы $P(o)$ с преобразованными вероятностями для каждого наблюдения o и столбец преобразованных вероятностей B состояний быть начальным могут быть получены из данных СММ. Следовательно, можно заранее вычис-

лить всевозможные варианты столбца $Probs_1$ как K матриц $PB(o)$:

$$PB(o) = P(o) \times B \quad \forall o \in O.$$

Далее в неспециализированной версии обрабатывается оставшаяся часть последовательности Obs:

$$Probs_t = P(Obs[t]) \times T^\top \times Probs_{t-1}.$$

Матрица переходов T также хранится в СММ. Это значит, что результат умножения матрицы $P(o)$ на $T^\top$ может быть получен для любого наблюдения o из множества O:

$$PT(o) = P(o) \times T^\top \quad \forall o \in O.$$

Все предпосчитанные матрицы сохраняются в памяти для дальнейшего переиспользования. Псевдокод специализированного алгоритма Витерби представлен на листинге 3.

```
HMM // Произвольная СММ
PB[HMM.K] // P(o) × B
PT[HMM.K] // P(o) × T^⊤

function spec_Viterbi()
  for i = 1..HMM.K
    PB[i] = P(HMM.O[i]) × HMM.B)
    PT[i] = P(HMM.O[i]) × (HMM.T)^⊤

function Viterbi(Obs)
  lo = length(Obs)
  Probs[1][HMM.K]

  Probs = PB(Obs[1])

  for i = 2..lo
    Probs = PT(Obs[i]) × Probs

  return Probs
```

Листинг 3: Алгоритм Витерби первого уровня специализации

Введем понятие *уровня специализации* — это количество наблюдений, которое обрабатывается за одно умножение матриц при вычисле-

ниях со второго и последующих наблюдений. Например, на листинге 3 уровень специализации равен одному, так как на строке 17 происходит обработка только одного наблюдения.

Далее можно воспользоваться тем фактом, что умножение матриц является ассоциативной операцией. Это позволяет увеличить уровень специализации и тем самым сократить количество матричных умножений. В формуле 2 показано, как обработать наблюдения o_t и o_{t-1} при условии, что $Probs_{t-2}$ известно:

$$\begin{aligned} Probs_t &= PT(o_t) \times Probs_{t-1} \\ &= PT(o_t) \times (PT(o_{t-1}) \times Probs_{t-2}) \\ &= (PT(o_t) \times PT(o_{t-1})) \times Probs_{t-2}. \end{aligned} \tag{2}$$

Результат умножения матриц $PT(o_t)$ и $PT(o_{t-1})$ можно получить, взяв данные из СММ. Такой подход дает основу для повышения уровня специализации, который ограничен лишь количеством имеющейся памяти для хранения предпосчитанных матриц. Так как произведение $PT(o_t) \times PT(o_{t-1}) \times PT(o_{t-2})$ может быть вычислено на стадии специализации, необходимо только одно умножение матриц. Та же самая идея может быть использована, чтобы получить четверый, пятый, и т.д. уровни специализации. На формуле 3 представлен способ для обработки трех наблюдений:

$$Probs_t = PT(o_t) \times PT(o_{t-1}) \times PT(o_{t-2}) \times Probs_{t-3}. \tag{3}$$

Псевдокод алгоритма Витерби с произвольным уровнем специализации представлен на листинге 4. Для того, чтобы получить нужный уровень специализации M, необходимо вычислить и сохранить произведение всевозможных комбинаций M матриц $PT(o)$.

```
1 HMM
2 PB[HMM.K]
3 PT[HMM.K]
4 level
5 // obs_lvl_handlers хранит всевозможные комбинации произведений level матриц из PT
6 obs_lvl_handlers[HMM.K^level]
7
```

```
function spec_Viterbi()
  for i = 1..HMM.K
    PB[i] = P(HMM.O[i]) × HMM.B)
    PT[i] = P(HMM.O[i]) × (HMM.T)^⊤
  calculate_combinations(obs_lvl_handlers, level, PT)

function Viterbi(Obs)
  // Обработка первого наблюдения
  Probs = PB[Obs[1]]

  lo = length(Obs)
  i = 2

  // Пока количество необработанных наблюдений больше или равно level
  while (lo − i) >= level)
    // Ищем матрицу для обработки следующих level наблюдений
    handler = obs_lvl_handlers.find(Obs[i:i+lvl])
    Probs = handler × Probs
    i = i + level

  // Количество необработанных наблюдений меньше, чем level
  for (; i < lo; i = i + 1)
    Probs = PT[Obs[i]] × Probs

  return Probs
```

Листинг 4: Алгоритм Витерби произвольного уровня специализации

В неспециализированной версии алгоритма Витерби необходимо выполнить $1 + 2 * (lo - 1)$ матричных умножений, где lo — это длина последовательности Obs. При специализации уровня M количество матричных умножений уменьшается, в таком случае нужно вычислить $(lo - 1)/M + (lo - 1)\ mod\ M$ произведений и выделить дополнительную память для хранения матриц PT и PB (это $2 * K$ матриц $N \times N$) и K^M матриц $N \times N$ для обработки последовательности наблюдений размером M.

Таким образом, при специализации алгоритма Витерби в терминах линейной алгебры возможно значительное сокращение количества матричных операций в сравнении с неспециализированной версией, но при этом растет количество требуемой памяти.

Однако следует учитывать, что числа с плавающей точкой могут быть представлены неточно в памяти компьютера. При вычислении

большого объема операций над числами с плавающей точкой возрастает погрешность, которая влияет на конечный результат. Смена порядка выполнения операций в свою очередь влияет на накопление этой погрешности, а из этого следует, что результат специализированной версии алгоритма Витерби может отличаться от неспециализированной версии на некоторую погрешность.

3.2. Особенности реализации

Целью специализации является повышение производительности специализируемого алгоритма. Для проверки на улучшение производительности необходимо реализовать два варианта алгоритма Витерби: неспециализированный и с произвольным уровнем специализации, которые описаны в подразделе 2.2.2 и разделе 3.1 соответственно.

3.2.1. Выбор технологий и тестирование корректности

При выборе библиотек для реализации нужно учитывать следующие факторы. Как следует из раздела 2.3, есть реализации как и на центральном процессоре (CPU), так и на графических процессорах общего назначения (GPGPU). Для работы с графами и их представлением в виде матриц сообществом был создан стандарт GraphBLAS [5]. Для проведения экспериментов по специализации алгоритма Витерби в терминах линейной алгебры была взята библиотека SuiteSparse:GraphBLAS [3], которая де-факто считается самой производительной и также является наиболее полной реализацией стандарта GraphBLAS. Алгоритмы этой библиотеки созданы с использованием OpenMP. К сожалению, код SuiteSparse:GraphBLAS предназначен только для выполнения на CPU, а стабильных и соответствующих стандарту GraphBLAS реализаций для запуска на GPGPU пока нет. Для проведения экспериментов на GPGPU была использована библиотека cuASR [4], которая реализована на основе библиотеки NVIDIA Cutlass [12]. Обе эти библиотеки предоставляют функции для работы с полукольцом *Min-plus*, что упрощает дальнейшую разработку.

Исходный код различных реализаций алгоритма Витерби доступен по ссылке [18]. Для взаимодействия с библиотеками был выбран язык C++, так как библиотека SuiteSparse:GraphBLAS разработана на языке C, а библиотека cuASR — на C++. В папке *Viterbi_impl* находятся реализации неспециализированного и специализированного алгоритма Витерби. Папка *tests* содержит следующие тесты для проверки отсутствия ошибок в программах:

- проверка правильности чтения данных из файлов;
- получение ожидаемого ответа на четырех конкретных СММ;
- сохранение семантики после специализации, то есть получение одинакового ответа для всех реализаций алгоритма Витерби при одинаковых входных данных.

Тест на сохранение семантики проводился на наборе данных из 3 последовательностей по 3500 наблюдений в каждой и 24 СММ, которые описаны далее в главе с экспериментами 4.1. При взаимодействии с низкоуровневыми средствами необходимо выявлять возможные ошибки, опечатки и утечки памяти, для этого используются инструменты Clang-tidy для статического анализа, Valgrind и cuda-memcheck для проверки на наличие утечек памяти и некорректных системных или библиотечных вызовов. Данные инструменты не выявили проблем, связанных с кодом реализаций алгоритма Витерби, но обнаружили некорректные вызовы при инициализации OpenMP. Это не влияет на прохождение тестов.

3.2.2. Описание формата входных параметров

Из-за того, что общепринятых форматов для описания скрытой марковской модели, соответствующей определению из раздела 2.1, и последовательности наблюдений не было найдено, для экспериментов был создан новый формат. В нем и состояния, и наблюдения кодируются с помощью натуральных чисел.

В файле с расширением *.chmm* находится СММ, которая закодирована следующим образом:

- количество состояний СММ N;
- количество состояний NZ, для которых вероятность быть начальным ненулевая;
- NZ строк вида:
  номер_состояния вероятность_быть_начальным;
- количество возможных наблюдений K;
- N строк, в каждой K вероятностей, каждая из которых содержит вероятность наблюдения события в состоянии, т.е. элементы матрицы E;
- количество переходов NT;
- NT строк вида:
  состояние_из состояние_куда вероятность_перехода.

Для описания набора последовательностей наблюдений предлагается следующий формат *.ess*:

- количество последовательностей в файле L;
- L пар строк вида, где lo — это длина последовательности:
  номер_последовательности lo
  наблюдение_1 наблюдение_2 ... наблюдение_lo.

Примеры СММ и последовательностей в таком формате могут быть найдены по ссылке [18] в папках *chmms_files* и *ess_files* соответственно.

Так как описанный ранее метод специализации может привести к снижению производительности алгоритма Витерби, далее необходимо поставить эксперименты по сравнению производительности реализаций неспециализированной и специализированной версий алгоритма.

4. Эксперименты

В данной главе описаны эксперименты по сравнению производительности неспециализированного алгоритма Витерби против специализированного алгоритма Витерби первого и второго уровня, а также по сравнению с реализацией из существующего решения CUDAMPF из подраздела 2.3.2. Существующее решение HMMer из подраздела 2.3.1 не измерялось, так как согласно статье [10] CUDAMPF превосходит HMMer более чем в 20 раз. Эти существующие решения взяты из биоинформатики. Так как предложенный подход специализации не связан с биоинформатикой, он может быть использован и в других областях, где CMM часто является зафиксированным параметром.

4.1. Описание набора данных и оборудования

Эксперименты выполнялись на рабочей станции с ОС Ubuntu 20.04, четырехядерном процессоре Intel Core i7-6700 с частотой 3.40 ГГц, 64 Гб оперативной памяти, видеокартой NVIDIA GeForce GTX 1070 с 8 Гб памяти и 1920 ядрами CUDA.

В качестве тестового набора были взяты CMM из репозитория проекта CUDAMPF. Это 24 так называемых *молчаливых CMM* [24], размером от 100 до 2405 состояний. Каждая из этих молчаливых CMM описывает вероятностный фильтр MSV, то есть CMM, специфичную для бионформатики. Эти CMM отличаются от CMM, описанных в разделе 2.1 тем, что состояния могут не создавать наблюдения. Из-за этого отличия алгоритм Витерби из раздела 2.2 не будет работать корректно. Реализация алгоритма Витерби в CUDAMPF адаптирована для работы с MSV. Данные молчаливые CMM из репозитория CUDAMPF можно использовать для моделирования вычислительной нагрузки. Чтобы ранее описанные варианты алгоритма Витерби могли работать с данными из молчаливой CMM, был разработан конвертер, который по молчаливой CMM создает CMM с теми же состояниями и, по возможности, сохраняет переходы между ними, либо добавляет новые переходы, если состояние не создавало наблюдений. Количество возможных наблю-

дений, т.е. K, равно 20. На рисунке 3 приведен график зависимости количества CMM, используемых в базе данных PFAM версии 34.0, от количества состояний в этих моделях. Минимальное количество состояний равно 7, а максимальное 2998.

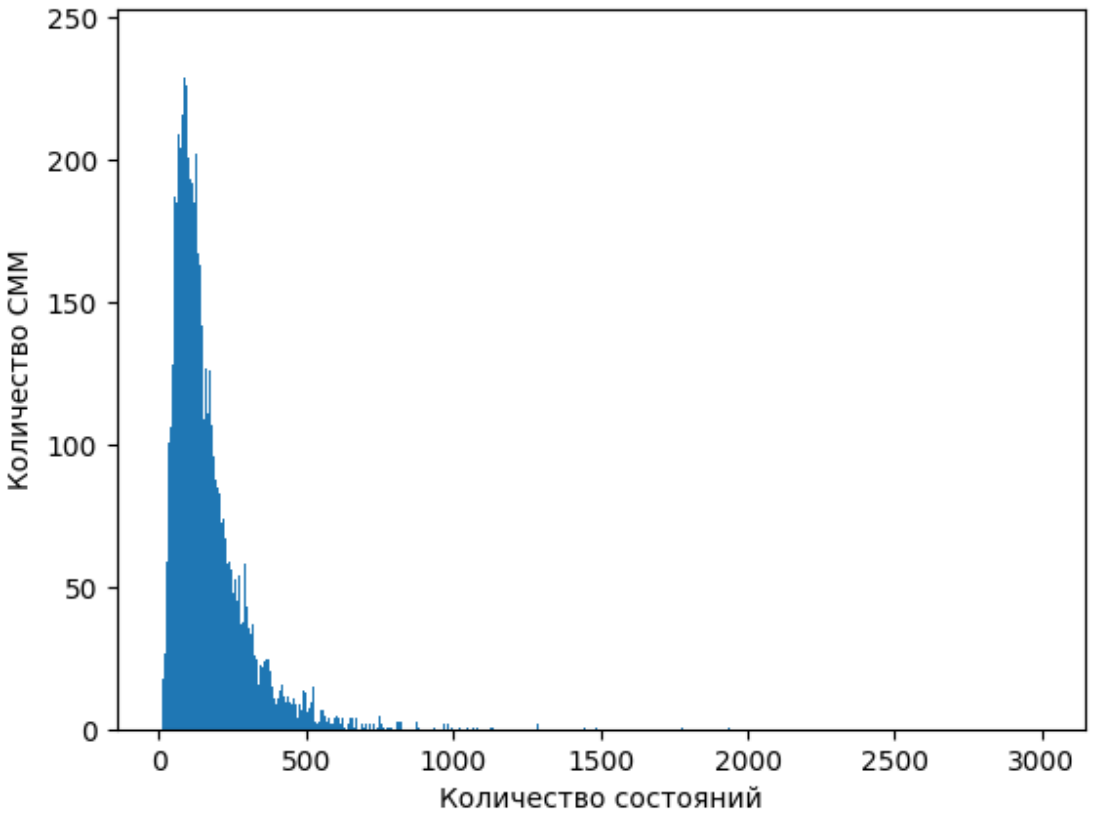


Рис. 3: График зависимости количества CMM от количества состояний в них

Были подготовлены следующие наборы последовательностей наблюдений:

- 3 последовательности по 3500 наблюдений;
- 3 последовательности по 7000 наблюдений;
- 16 последовательностей размером от 38 до 7096;
- 50 последовательностей по 3500 наблюдений.

Первый, второй и четвертый наборы были сгенерированы случайным образом, в то время как третий был взят из БД PFAM и приведен к формату *ess*, который описан в разделе 3.2.2.

4.2. Анализ результатов

Для получения времени выполнения каждая реализация алгоритма Витерби запускалась 10 раз, и из этих результатов бралась медиана. Измерялось время обработки всего набора последовательностей при зафиксированной СММ конкретной реализацией. Если реализация специализированная, то также измерялось время, необходимое на выполнение специализации.

Для реализации с использованием SuiteSparse:GraphBLAS были получены следующие результаты, представленные на рисунках 4, 5, 6, 7 и 8. Использовались специализированные версии первого и второго уровня, для специализированной версии третьего уровня не хватило оперативной памяти. Специализация второго уровня сильно медленнее других реализаций, поэтому она не отображена на графиках. В таблице 1 указано общее время обработки с учётом затрат на специализацию.

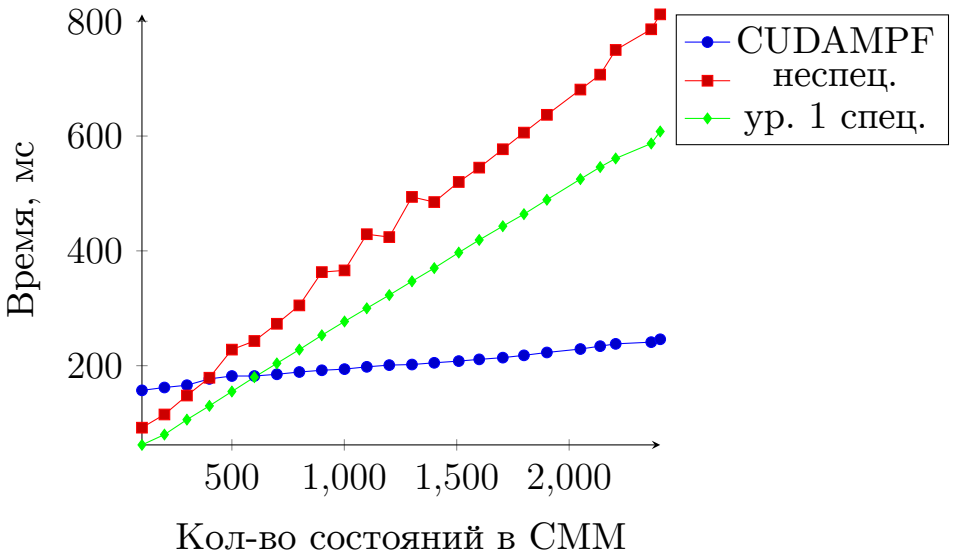


Рис. 4: GraphBLAS, 3 x 3500 наблюдений, меньше — лучше

Как можно видеть из графиков и таблицы, специализированная версия первого уровня превосходит неспециализированную. Стоит отме-

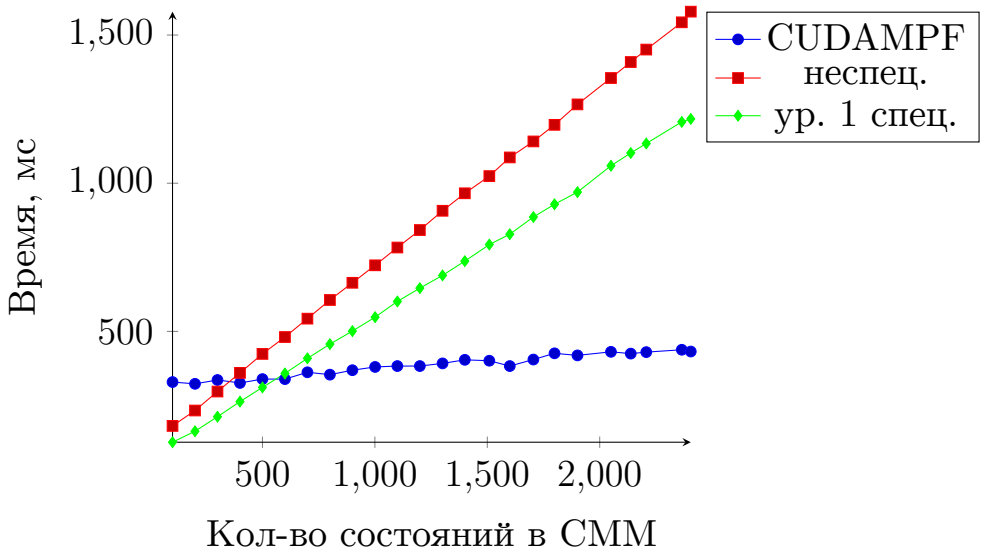


Рис. 5: GraphBLAS, 3 x 7000 наблюдений, меньше — лучше

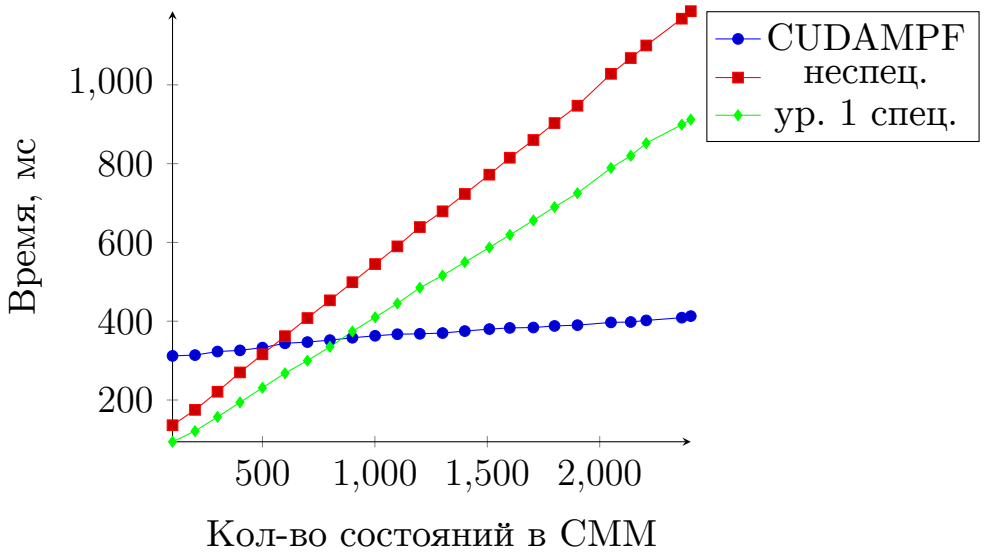


Рис. 6: GraphBLAS, набор данных из БД PFAM, меньше — лучше

	CUDAMPF	неспец.	Ур. 1	Ур. 2
3 x 3500	4854	10765	8062	215329
3 x 7000	9209	21062	16152	387464
Набор из PFAM	8796	15864	12036	298269
50 x 3500	103036	176263	134259	2921104

Таблица 1: GraphBLAS, общее время обработки с учетом затрат времени на специализацию, мс

тить, что для использованных данных специализация первого уровня хуже CUDAMPF примерно в 1,3-1,7 раза. Также важным является тот

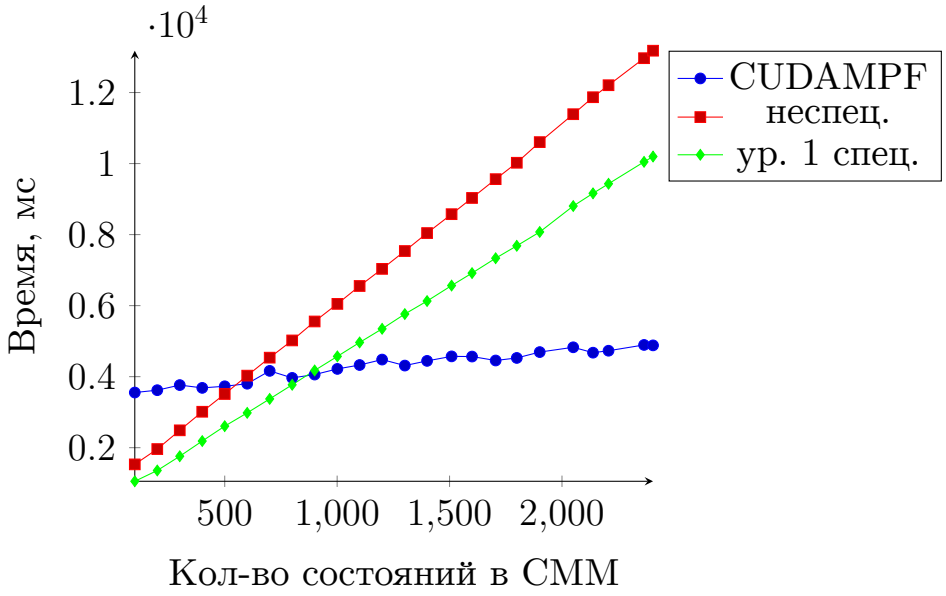


Рис. 7: GraphBLAS, 50 x 3500 наблюдений, меньше — лучше

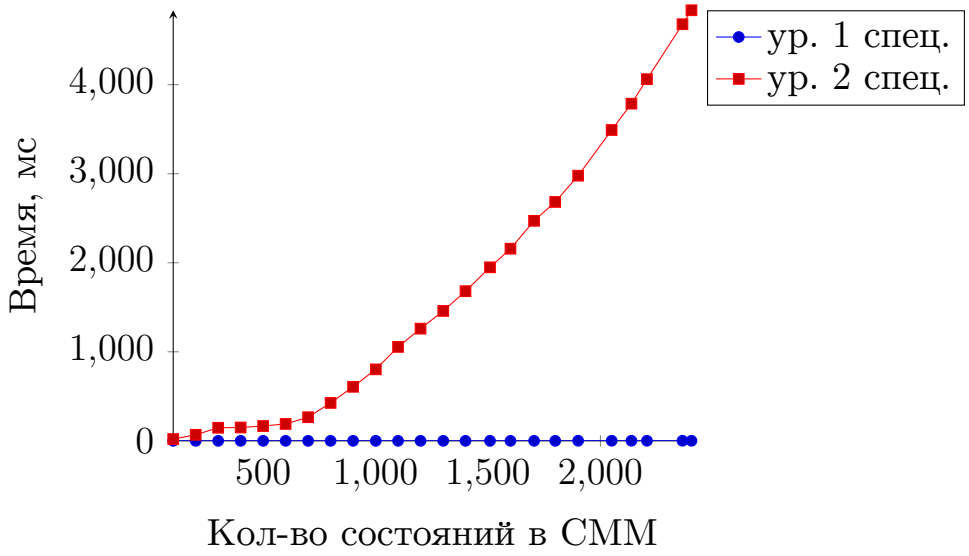


Рис. 8: GraphBLAS, время на специализацию, меньше — лучше

факт, что при обработке CMM с количеством состояний меньше 500, специализированная версия производительнее CUDAMPF. Как видно из рисунка 3, таких CMM подавляющее большинство.

При измерении специализированных реализаций с использованием библиотеки cuASR были получены следующие результаты, которые представлены на рисунках 9, 10, 11, 12 и 13, а также в таблице 2. Здесь использовалась специализация первого и второго уровня, но для CMM, у которых состояний больше чем 2000, на видеокарте не хватило памя-

ти для специализации второго уровня.

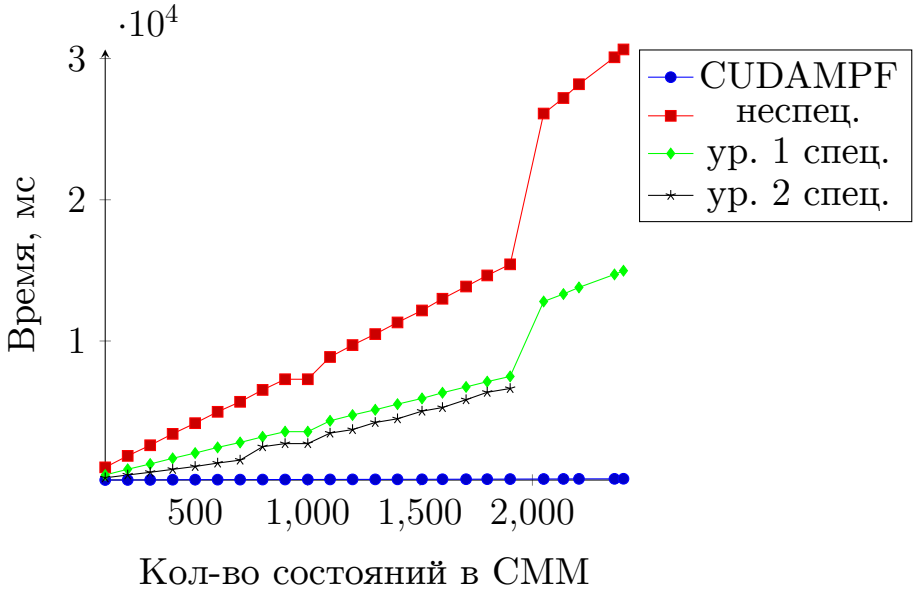


Рис. 9: cuASR, 3 x 3500 наблюдений, меньше — лучше

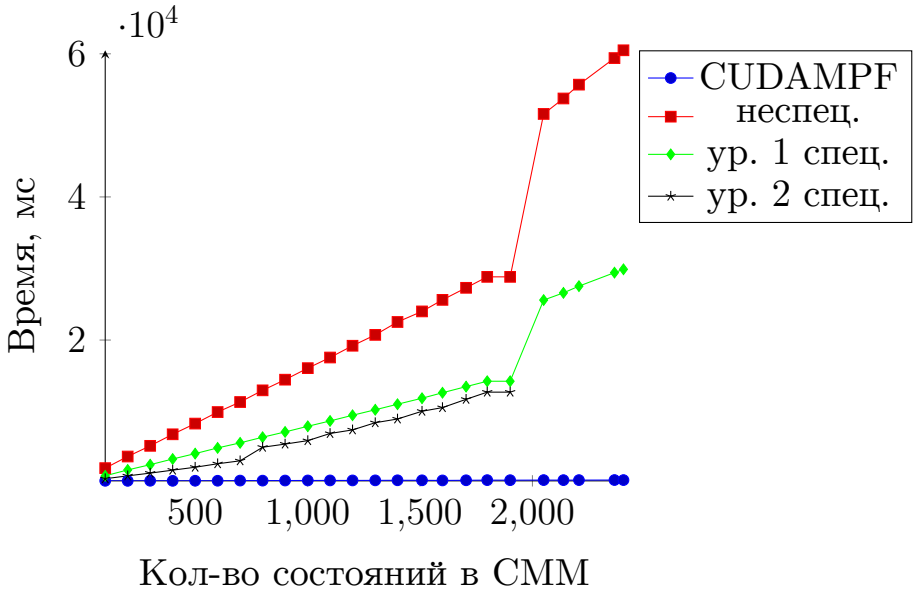


Рис. 10: cuASR, 3 x 7000 наблюдений, меньше — лучше

Резкий скачок, который появляется при обработке СММ с количеством состояний более чем 2000 объясняется тем, что количество ядер CUDA на использованной видеокарте равно 1920. Исходя из полученных результатов, можно сделать вывод, что при повышении специализации до второго уровня, снижается время обработки последовательностей. В конкретном случае, реализации оказались не такими эффек-

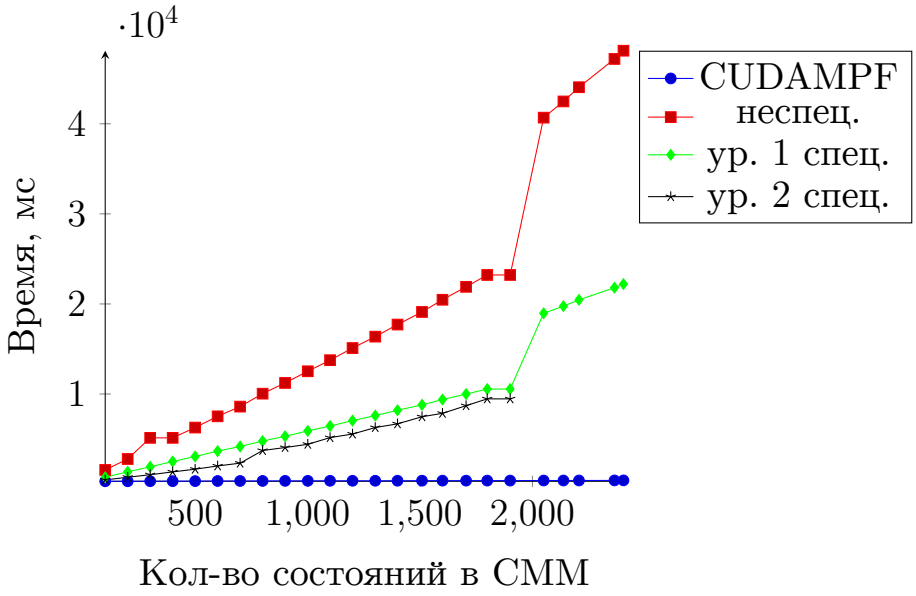


Рис. 11: cuASR, набор данных из БД PFAM, меньше — лучше

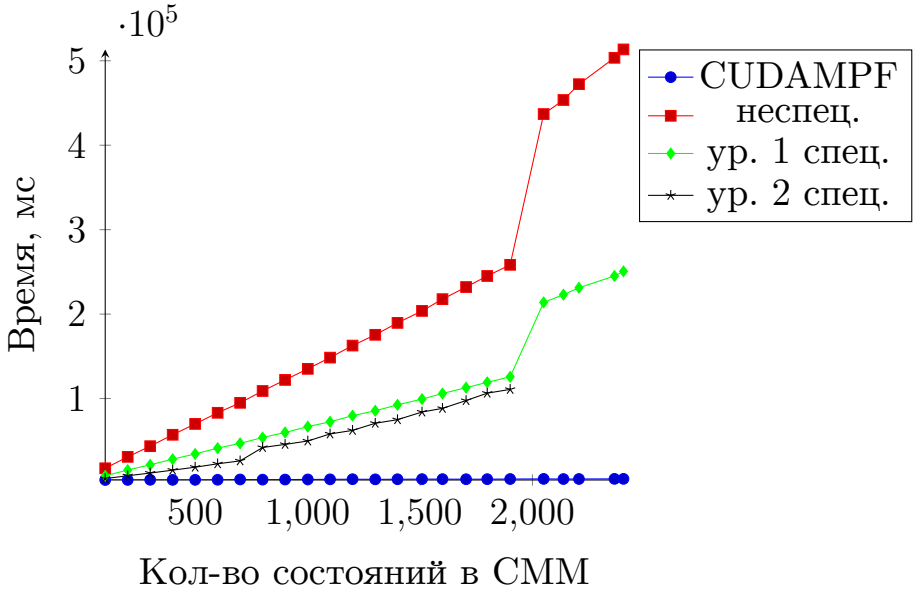


Рис. 12: cuASR, 50 x 3500 наблюдений, меньше — лучше

	CUDAMPF	неспец.	Ур. 1	Ур. 2
3 x 3500	3666	154415	77758	85977
3 x 7000	7053	305578	153216	145176
Набор из PFAM	6777	241687	114214	114766
50 x 3500	78930	2596532	1271980	1025473

Таблица 2: cuASR, общее время обработки с учетом затрат времени на специализацию, кол-во состояний CMM меньше 2000, мс

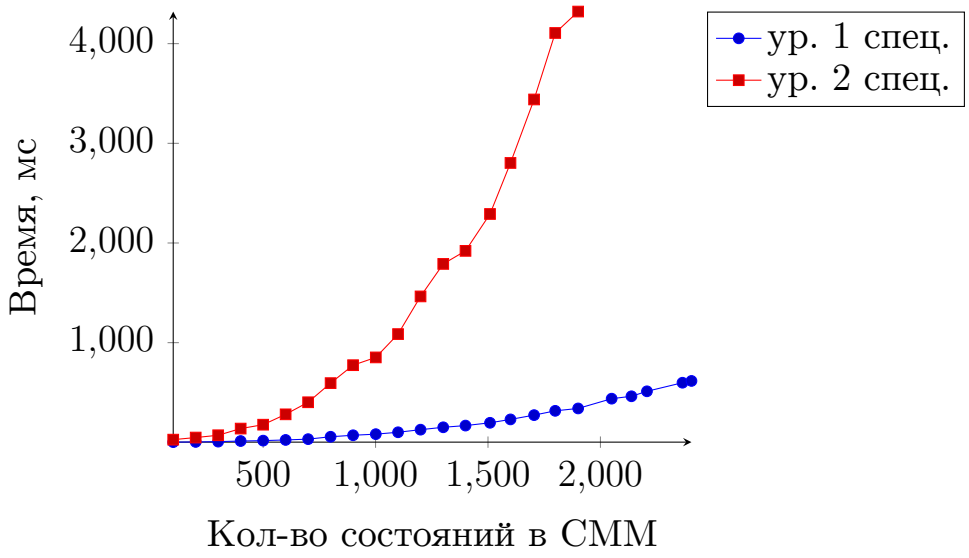


Рис. 13: cuASR, время на специализацию, меньше — лучше

тивными по сравнению с CUDAMPF и GraphBLAS, но применение специализации дает существенный прирост производительности.

При анализе проведённых экспериментов видно, что специализированный алгоритм Витерби производительнее, чем неспециализированная версия, то есть специализация дает прирост по скорости обработке последовательностей наблюдений за счет уменьшения количества матричных операций. Стоит также отметить, что реализация алгоритма Витерби с использованием библиотек линейной алгебры существенно проще, чем аналогичная реализация с использованием NVIDIA CUDA за счет более высокого уровня абстракции.

5. Заключение

В ходе работы были получены результаты, перечисленные ниже.

- Выполнен обзор предметной области:
  - рассмотрен алгоритм Витерби и его реализации HMMER и CUDAMPF;
  - изучена техника специализации.
- Реализованы и протестированы две реализации специализированного алгоритма Витерби, описанного с помощью алгебраической структуры полукольцо *Min-plus* и матричных операций:
  - с использованием библиотеки SuiteSparse:GraphBLAS для выполнения на центральном процессоре;
  - с использованием библиотеки CUSP для выполнения на графических процессорах общего назначения.
- Проведены эксперименты на данных из репозитория CUDAMPF. Установлено, что специализированный алгоритм Витерби первого уровня производительнее неспециализированной версии.

Статья *Viterbi Algorithm Specialization Using Linear Algebra* была принята на конференции SEIM 2021, а также проведена публичная презентация результатов [21].

Исходя из вышеперечисленных результатов, специализация алгоритма Витерби, выраженного методами линейной алгебры, скрытой марковской моделью может дать значительный прирост производительности.

Список литературы

[1] Aggarwal C. Linear Algebra and Optimization for Machine Learning: A Textbook. — 2020. — 01. — ISBN: 978-3-030-40343-0.

[2] Partial Evaluation Applied to Ray Tracing : Rep. ; Executor: P. H. Andersen : 1995.

[3] Davis T. A. Algorithm 1000: SuiteSparse:GraphBLAS: Graph Algorithms in the Language of Sparse Linear Algebra. — Vol. 45. — URL: https://doi.org/10.1145/3322125.

[4] Thakkar V., Kannan R., Sao P. et al. Dense Semiring Linear Algebra on Modern CUDA Hardware. — 2021. — Minisymposium at SIAM CSE'21: GraphBLAS: Tools, Algorithms, and Applications. URL: https://github.com/hpcgarage/cuASR.

[5] Design of the GraphBLAS API for C / A. Buluç, T. Mattson, S. McMillan et al. // IPDPS Workshops. — 2017. — P. 643–652.

[6] Eddy S. R. What is a hidden Markov model? // Nature Biotechnology. — Vol. 22. — P. 1315–1316. — URL: https://doi.org/10.1038/nbt1004-1315.

[7] Eddy S. R. Accelerated Profile HMM Searches // PLoS computational biology. — 2011. — Vol. 7, no. 10. — P. e1002195–e1002195.

[8] Gentle J. Numerical Linear Algebra for Applications in Statistics. — Springer-Verlag, 1998.

[9] HMMer. — URL: https://github.com/EddyRivasLab/hmmer (online; accessed: 2020-06-01).

[10] Jiang H., Ganesan N. CUDAMPF: a multi-tiered parallel framework for accelerating protein sequence search in HMMER on CUDA-enabled GPU // BMC bioinformatics. — 2016.

[11] Jones N., Gomard C., Sestoft P. Partial Evaluation and Automatic Program Generation. — 1993.

[12] NVIDIA Cutlass. — URL: https://github.com/NVIDIA/cutlass (online; accessed: 2021-25-05).

[13] Partial Evaluation of Logic Programs in Vector Spaces / C. Sakama, H. Nguyen, T. Sato, K. Inoue. — EasyChair Preprint no. 172, EasyChair, 2018.

[14] Petropoulos A., Chatzis S. P., S.Xanthopoulos. A novel corporate credit rating system based on Student's-t hidden Markov models // Expert Systems with Applications. — 2016. — Vol. 53. — P. 87–105.

[15] Pfam: The protein families database in 2021 / J. Mistry, S. Chuguransky, L. Williams et al. // Nucleic Acids Research. — 2020. — 10. — Vol. 49, no. D1. — P. D412–D419.

[16] Query compilation in PostgreSQL by specialization of the DBMS source code / E. Sharygin, R. Buchatskiy, R. Zhuykov, A. Sher // Programming and Computer Software. — 2017. — 11. — Vol. 43. — P. 353–365.

[17] Rabiner L. R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition // Readings in Speech Recognition. — San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1990. — P. 267–296. — ISBN: 1558601244.

[18] Spec_Viterbi. — URL: https://github.com/IvanTyulyandin/Spec_Viterbi (online; accessed: 2021-25-05).

[19] Szeliski R. Computer Vision: Algorithms and Applications. — 1st edition. — Berlin, Heidelberg : Springer-Verlag, 2010. — ISBN: 1848829345.

[20] Theodosis E., Maragos P. Analysis of the Viterbi Algorithm Using Tropical Algebra and Geometry // 2018 IEEE 19th International

Workshop on Signal Processing Advances in Wireless Communications (SPAWC). — 2018. — P. 1–5.

[21] Tyulyandin I., Berezun D., Grigorev S. Viterbi Algorithm Specialization Using Linear Algebra. — to be appear in Proceedings of the Sixth Conference on Software Engineering and Information Management 2021 (SEIM 2021) Saint Petersburg, Russia, April 17, 2021.

[22] Tyurin A., Berezun D., Grigorev S. Optimizing GPU programs by partial evaluation. — 2020. — 02. — P. 431–432.

[23] Viterbi A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm // IEEE Transactions on Information Theory. — 1967. — Vol. 13, no. 2. — P. 260–269.

[24] Wakabayashi K. Silent HMMs: Generalized Representation of Hidden Semi-Markov Models and Hierarchical HMMs // Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing.