Санкт-Петербургский государственный университет

Кафедра системного программирования

Тарасенко Никита Дмитриевич

Распараллеливание на GPGPU фильтров Forward и Backward-Forward в задаче поиска гомологов генов скрытыми марковскими моделями

Выпуская квалификационная работа

Научный руководитель:
ст. преп Сартасов С. Ю.

Рецензент:
канд. физ.-мат. наук, ведущий инженер-программист Бычков А. Б.

Санкт-Петербург
2018

SAINT PETERSBURG UNIVERSITY

Software engineering

Tarasenko Nikita

Parallelizing Forward и Backward-Forward filters on GPGPU in a problem of searching genes homologs by hidden Markov models

Graduation Thesis

Scientific supervisor:
senior lecturer Stanislav Sartasov

Reviewer:
cand. of phys.-math. sciences, senior software engineer Alexandr Bychkov

Saint Petersburg
2018

Оглавление

Введение

Идентификация и изучение сетей взаимосвязанных белков позволяет лучше понять молекулярные механизмы биологических процессов, понять физиологию и патологию клетки, а в конечном итоге и всего организма. Однако для определения функции белка необходимы трудоемкие экспериментальные исследования. Таким образом, возникает нарастающее отставание между получением биологических последовательностей (генов и белков) и определением функции этих последовательностей.

Парадоксальность ситуации состоит в том, что объём получаемых функций белка намного больше того, который можно осмыслить, проверить и использовать в эксперименте. Из-за этого использование эффективных алгоритмов и мощного оборудования необходимо для более качественного и быстрого анализа последовательностей в различных исследованиях.

Одим из часто используемых инструментов для работы с последовательностями является инструмент HMMer [2], который содержит в себе полезные функции для анализа последовательностей нуклеотидов. Однако главным недостатком его является тот, что он реализован для CPU. Для более высокой производительности можно использовать GPU с его многопоточностью и высокими вычислительными мощностями, и работы в этом направлении уже ведутся.

Алгоритмы *Forward* и *Forward-Backward* активно используются в инструменте HMMer для построения скрытых марковских моделей и анализа последовательностей, но существующая реализация этих алгоритмов требует большого колличества процессорного времени [3]. Существующие аналоги на GPU дают прирост производительности в среднем в 2,5 раза, что не является пределом.

1. Постановка задачи

Целью данной работы является создание эффективной версии алгоритмов *Forward* и *Forward-Backward* на GPU и проведение анализа полученных результатов производительности по сравнению с существующей реализацией в инструменте HMMer на CPU. Для ее достижения были сформулированы следующие задачи:

- сделать обзор предметной области и существующих решений;
- разработать модель оптимизации алгоритмов *Forward* и *Backward*;
- реализовать протоипы алгоритмов *Forward* и *Backward*, основанные на разработанной схеме распараллеливания посредством технологии *NVIDIA CUDA*;
- провести сравнение реализованного инструмента с существующим аналогом.

2. Литературный обзор

2.1. Технология NVIDIA CUDA

CUDA (Compute Unified Device Architecture) [7] - параллельная вычислительная платформа и модель программирования, которая позволяет разработчикам использовать графические процессоры NVIDIA для эффективной обработки общих задач. Когда основной процесс запускает ядро GPU, генерируется большое количество потоков. Каждый поток выполняет экземпляр ядра GPU.

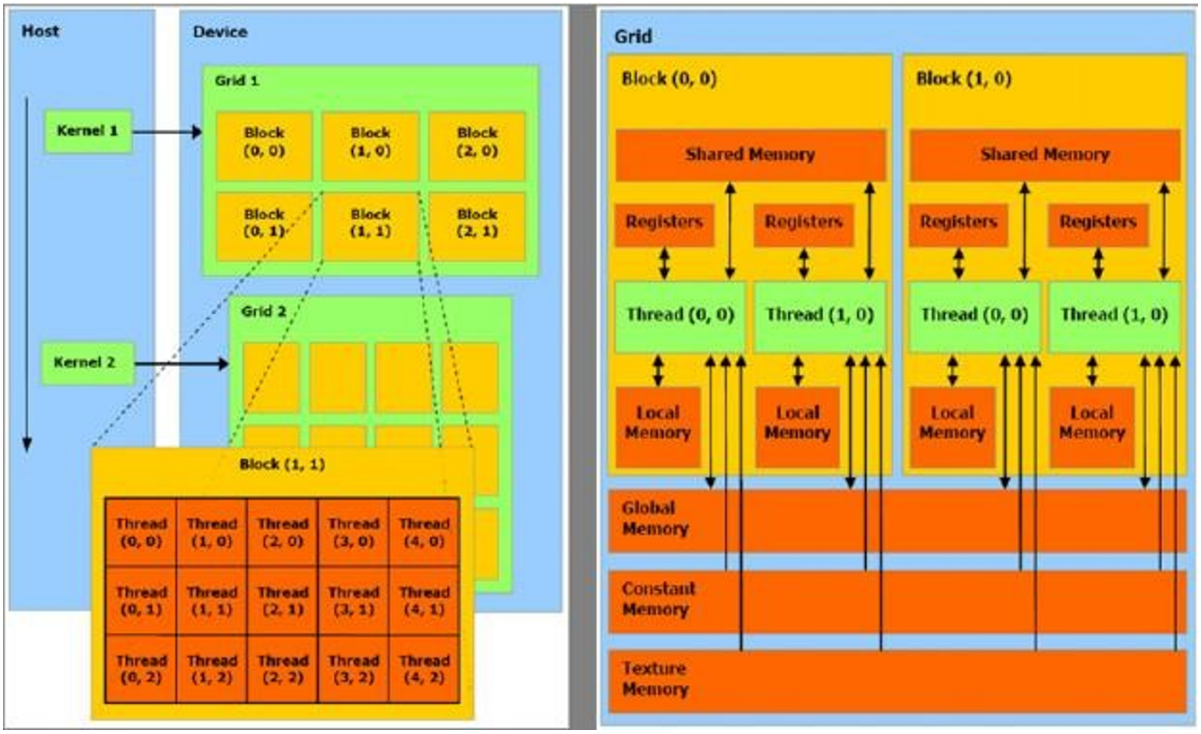


Рис. 1: Внутреннее строение ядра CUDA [8]

2.2. Инструмент HMMer

HMMer – бесплатный и часто используемый программный пакет для анализа последовательности. Его главная особенность заключается в идентификации гомологичных[1] белковых или нуклеотидных последовательностей и их выравнивания. Он обнаруживает гомологию, срав-

[1]Гомология - это существование общего предка между парой структур или генов

нивая профиль скрытых Марковских моделей (CMM) с одной последовательностью или с базой данных последовательностей. Профили CMM создаются из множественного выравнивания последовательностей.

2.2.1. Скрытые Марковские модели (CMM)

Скрытые марковские модели являются формальной основой для создания вероятностных моделей линейной последовательности. CMM описываются следующим образом:

- множество состояний: $Q = \{q_1, q_2, ..., q_n\}$;
- выходной алфавит: $V = \{v_1, v_2, ..., v_m\}$;
- вероятность нахождения в состоянии q_i, в момент времени $t = 0$: $\pi(i)$;
- вероятность перехода из состояния q_i в состояние q_j: $\{a_{ij}\}$;
- вероятность выброса: $\{b_j(k)\}$, где $b_j(k) = Pr[$выброс v_k в момент времени $t|$ в состоянии q_j в момент времени $t]$;

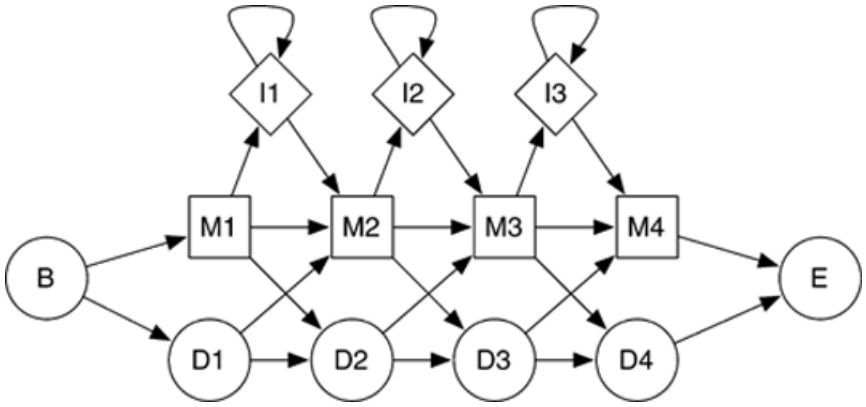


Рис. 2: Пример CMM

В инструменте HMMer используются расширенная версия CMM – Plan7 (Рис. 3). Данная модель, в отличии от стандартной, позволяет находить несколько совпадений во входящей последовательности.

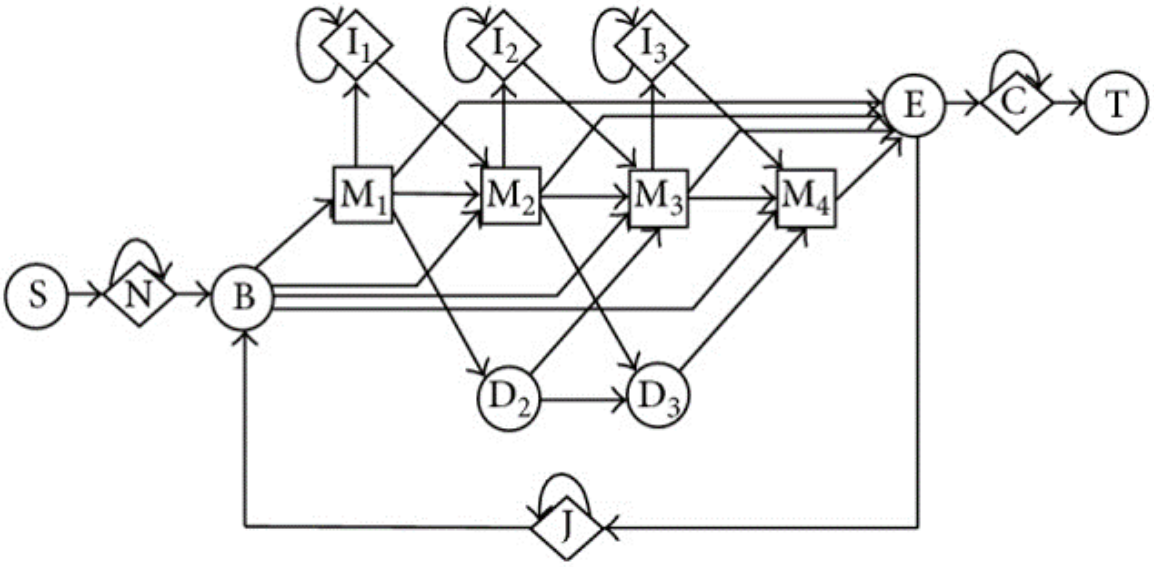


Рис. 3: Пример Plan7 [6]

2.2.2. Алгоритм Forward

Для вычисления вероятности похожести исследуемой цепочки для данной модели применяют алгоритм Forward. Чтобы получить искомую вероятность, проводится суммирование по всем возможным путям, которые могут привести к рассматриваемой цепочке. Для эффективного вычисления данной величины используется динамическое программирование.

Формально алгоритм Forward можно описать следующим образом:

Input: вероятности перехода $a[N, N]$, Вероятности выброса $b[N, T]$, состояния СММ $state[N]$, рассматриваемая последовательность $seq[T]$
Output: оценка последовательности $forward[seq[T], T]$

function FORWARD($a[N, N], b[N, T], state[N], seq[T]$)
  create $forward[N + 2, T]$
  for all s in $state$ **do**
    $forward[s, 0] \leftarrow a[0, s] * b[s, seq[0]]$
  end for
  for t **from** 2 to $T - 1$ **do**
    for s **from** 1 to N **do**
      $forward[s, t] \leftarrow \sum_{q=1}^{N} forward[q, t - 1] * a[q, s] * b[s, seq[t]]$

end for
end for
$forward[seq[T], T] \leftarrow \sum_{s=1}^{N} forward[s, T] * a[s, seq[T]]$
end function

2.2.3. Алгоритм Backward

Данный алгоритм позволяет расчитать вероятность рассматриваемой цепочки от момента времени $t+1$ и до конца. По схеме вычисления Backward совпадает с алгоритмом Forward:

Input: вероятности перехода $a[N, N]$, Вероятности выброса $b[N, T]$, состояния СММ $state[N]$, рассматриваемая последовательность $seq[T]$
Output: оценка последовательности $backward[0, T]$

function BACKWARD($a[N, N], b[N, T], state[N], seq[T]$)
create $backward[N + 2, T]$
for all s **from** $N - 1$ to 1 **do**
$backward[s, T] \leftarrow a[T, seq[s + 1]]$
end for
for t **from** $T - 1$ to 1 **do**
for s **from** 1 to N **do**
$backward[s, t] \leftarrow \sum_{q=1}^{N} backward[q, t + 1] * a_[q, s] * b[s, seq[t + 1]]$
end for
end for
$backward[0, T] \leftarrow \sum_{s=1}^{N} backward[s, 1] * a[0, s] * b[s, seq[1]]$
end function

2.2.4. Алгоритм Forward-Backward

Данный алгоритм используется при "обучении" СММ, корректируя вероятности перехода и выброса. Также используется для вычисления апостериорной вероятности[2], которая является произведением резуль-

[2]Апостериорная вероятность - условная вероятность случайного события при условии того, что известны апостериорные данные, т.е. полученные после опыта

татов алгоритмов Forward и Backward.

2.3. Обзор статей

В статье [3] анализируется работа инструмента HMMer. В таблице (Рис. 4) показаны затраты процессорного времени на примере работы модуля HMMer – JackHMMer. Видно, что на алгоритмы Forward и Backward выпадает 67% процентов всей работы процессора. Высокозатратность обусловлена спецификой данных алгоритмов - они являются рекурсивными. В HMMer данные алгоритмы реализованы посредством динамического программирования. Однако даже с этим улучшением эти операции остаются ресурсоемкими.

Jackhmmer

Name	Thread samples	CPU clock (%)	Thread events
forward_engine	64	36.57%	136192000
backward_engine	54	30.86%	114912000
p7_Viterbi	34	19.43%	72352000
p7_Decoding	9	5.14%	19152000
p7_Null2_ByExpectation	7	4.00%	14896000
p7_alidisplay_Create	1	0.57%	2128000
p7_oprofile_FGetEmission	1	0.57%	2128000
is_multidomain_region	1	0.57%	2128000
rescore_isolated_domain	1	0.57%	2128000
p7_MSVFilter	1	0.57%	2128000
get_postprob	1	0.57%	2128000
esl_hmm_Forward	1	0.57%	2128000

Рис. 4: Обзор затрат ресурсов CPU на примере JackHMMer[3]

В той же статье говорится про ускорение некоторых функций HMMer с помощью технологии NVIDIA CUDA. Авторы в своих эксперементах смогли добиться ускорения в среднем в 2-2.5 раз на входящих данных разной длины.

Существует оптимизированная реализация инструмента HMMer, которая использует векторные команды процессора (CPU). При такой реализации создатели HMMer получают ускорения в 1,5 раза. Однако

все упирается в возможности процессора и на данный момент большей скорости добиться не удастся.

Существует проект с открытым кодом CUDAMPF [1], который содержит адаптированные для работы в CUDA алгоритмы MSV/SSV и Viterbi [5]. По сути CUDAMPF содержит в себе отдельные модули, которые можно использовать при переносе всей функциональности HMMer на GPU. Из-за этого факта данный проект был выбран в качестве основы для данной работы.

3. Модель оптимизации

Учитывая тот факт, что оба алгоритма Forward и Backward являются рекурсивными, проводить оптимизацию следует на уровне вычислений.

Первая идея для оптимизации – конвертирование данных в более удобный для взаимодействия вид. Изначально вся CMM описывается несколькими матрицами размерности NxM (N – длина рассматриваемой последовательности, M – размер CMM). Для дальнейшей работы необходимо провести векторизацию данных. Каждая матрица упаковывается в векторы размера в 32 элемента, количество таких векторов зависит от размера CMM. Соответствующий элемент в векторах соответствует определенному состоянию CMM.

Вторая идея – вычисление определенных значений для каждого состояния CMM параллельно. Для этого в ядре CUDA выделяется 32 потока (соответствует 1 варпу[3]) для вычисления всех значений для каждого состояния на определенном шаге (Рис. 5). В конце каждого шага происходит суммирование всех полученых результатов, итог которого используется далее в следующем шаге.

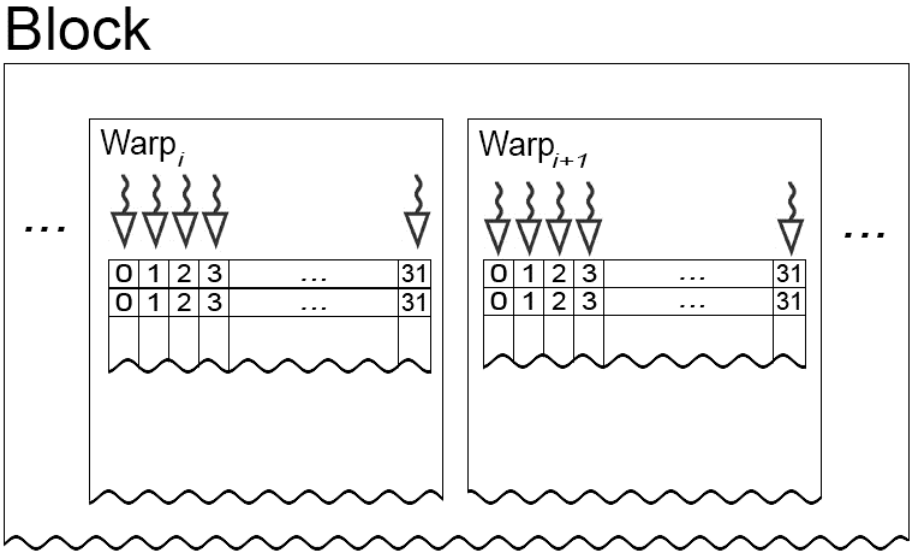


Рис. 5: Схема распараллеливания

[3]Варп(англ. Warp) – единица исполнения мультипроцессора, содержащая 32 потока

Если ресурсы видеокарты позволяют использовать большее количество потоков, то имеет смысл выделить несколько варпов, которые будут обрабатывать входящую базу последовательностей параллельно.

4. Реализация

Как говорилось ранее, после вычисления промежуточных значений необходимо их просуммировать. Для этих целей были использованы специальные инструкции процессоров CUDA – *shfl* команды. Они позволяют производить вычисления игнорируя чтение/запись в память, обращаясь непосредственно к регистрам разных потоков внутри одного варпа, что при обычной работе невозможно. Таким образом *shfl* команды позволяют сэкономить память мультипроцессоров, исключив чтение/запись (Рис. 6).

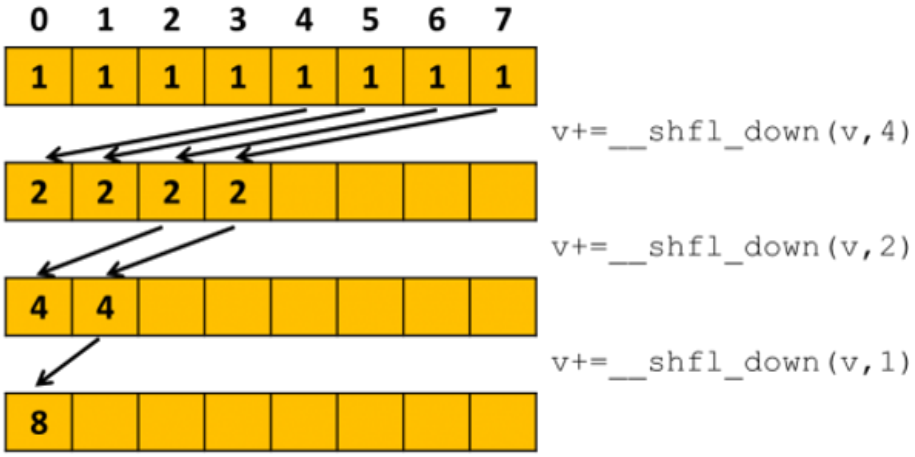


Рис. 6: Пример работы одной из *shfl* команд – *shfl_ down*[4]

Еще одной особенностью реализации является уход от хранения результатов вычислений в матрице. На каждом шаге при вычислениях используются предыдущие значения, но хранение всех результатов вычислений ведет за собой выделение большого размера памяти, что в случае с видеокартами является критичной проблемой. По этой причине очевидным шагом будет хранение только предыдущих результатов, т.к. по факту для конкретной рассматриваемой последовательности нам интересно значение не на каком-то шаге, а итоговый результат.

5. Апробация

В апробации использовалась система со следующими характеристиками:

- CPU: AMD Ryzen 1500X
- GPU: NVIDIA Nvidia GeForce GTX 1060 6Gb

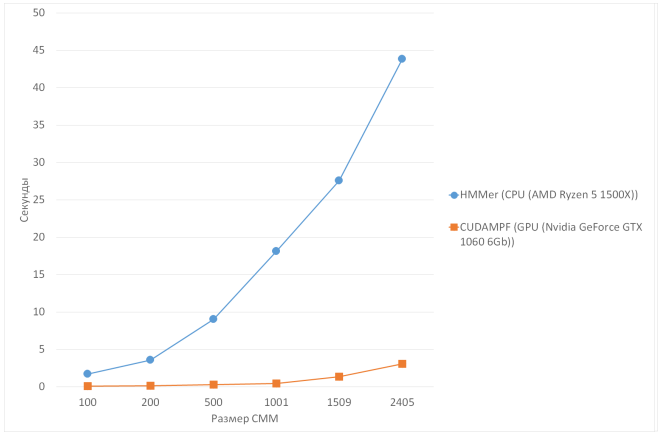
Во время тестов использовалась база последовательностей размера 2000 шт. длиной в 400 элементов каждая.

Таблица 1: Результат апробации для алгоритма Forward (сек)

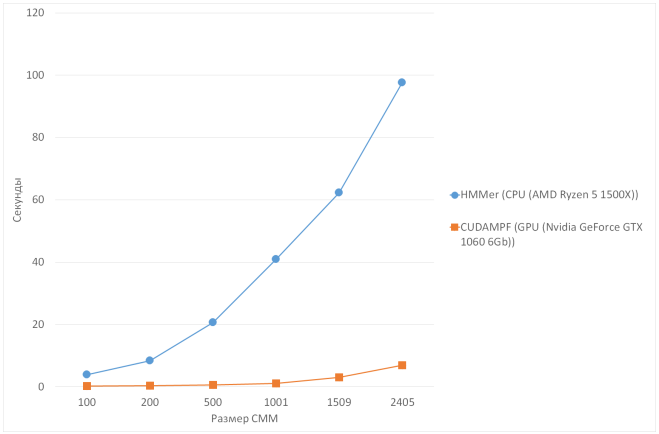
	Размер СММ					
Реализация (архитектура)	100	200	500	1001	1509	2405
HMMer (CPU (AMD Ryzen 5 1500X))	1.73	3.6	9.04	18.12	27.58	43.84
CUDAMPF(GPU (Nvidia GeForce GTX 1060 6Gb))	0.115	0.148	0.299	0.465	1.369	3.075

Таблица 2: Результат апробации для алгоритма Forward-Backward (сек)

	Размер СММ					
Реализация (архитектура)	100	200	500	1001	1509	2405
HMMer (CPU (AMD Ryzen 5 1500X))	3.96	8.37	20.71	40.97	62.33	97.6
CUDAMPF(GPU (Nvidia GeForce GTX 1060 6Gb))	0.274	0.356	0.704	1.148	3.127	6.934



(a) Forward



(b) Forward-Backward

Рис. 7: Сравнение производительности алгоритмов на разных архитектурах

Заключение

В ходе выполнения данной выпускной квалификационной работы были достигнуты следующие результаты:

- проведен анализ предметной области и существующих решений;
- разработана схема распараллеливания;
- реализованы прототипы алгоритмов, основанные на разработанной схеме;
- проведена апробация данных прототипов и сделан сравнительный анализ с существующими решениями.

Полученно ускорение алгоритмов в 15-30 раз по сравнению с последовательными версиями алгоритмов Forward и Forward-Backward.

Список литературы

[1] CUDAMPF. — URL: https://github.com/Super-Hippo/CUDAMPF.

[2] Eddy S. R. HMMer. — URL: http://hmmer.org/.

[3] Fahian Ahmed Saddam Quirem Gak Min, Lee Byeong Kil. Hotspot Analysis Based Partial CUDA Acceleration of HMMER 3.0 on GPGPUs. — 2012. — URL: https://pdfs.semanticscholar.org/bf4f/dd7e5d42d8d60e82198169893b6abed11893.pdf.

[4] Faster Parallel Reductions on Kepler. — URL: https://devblogs.nvidia.com/faster-parallel-reductions-kepler.

[5] Hanyu Jiang Narayan Ganesan. CUDAMPF: a multi-tiered parallel framework for accelerating protein sequence search in HMMER on CUDA-enabled GPU. — 2016. — URL: https://bmcbioinformatics.biomedcentral.com/track/pdf/10.1186/s12859-016-0946-4.

[6] Modern Computational Techniques for the HMMER Sequence Analysis.

[7] NVIDIA CUDA. — URL: https://developer.nvidia.com/cuda-zone.

[8] On The Parallelization Of Integer Polynomial Multiplication.