Программная реализация алгоритма Думера декодирования двоичных линейных кодов в различных моделях параллельных вычислений

Чжай Хао

***Аннотация*—В данной работе исследуются и реализуются параллельные версии алгоритма Думера, предназначенного для решения NP-полной задачи синдромного декодирования, которая лежит в основе безопасности постквантовых криптосистем на основе кодов, таких как система Мак-Элиса. Несмотря на теоретическую эффективность, практическое применение алгоритма Думера ограничено его высокой вычислительной сложностью. Для преодоления этого барьера были разработаны и тщательно проанализированы четыре версии алгоритма: базовая последовательная реализация на C++, многопоточная версия с использованием OpenMP для многоядерных CPU, GPU-ускоренная версия на основе CUDA и гибридная модель, объединяющая вычислительные мощности CPU и GPU. Корректность всех реализаций была подтверждена на стандартизированных тестовых данных с платформы decodingchallenge.org. Экспериментальные результаты демонстрируют, что для матриц малого размера наиболее эффективной является OpenMP-реализация, тогда как для задач большого масштаба гибридный подход показывает наилучшую производительность. Это исследование значительно повышает практическую применимость алгоритма Думера, значительно сокращая время решения задач и позволяя более точно оценивать реальную стойкость кодовых криптосистем.**

***Ключевые слова*—алгоритм Думера, синдромное декодирование, параллельные вычисления, OpenMP, CUDA, постквантовая криптография**

Введение

В контексте развития постквантовой криптографии особое внимание уделяется криптосистемам на основе кодов, среди которых система Мак-Элиса [1] является одним из финалистов стандартизации NIST. Безопасность таких систем основана на NP-полной задаче синдромного декодирования (SDP) [2], для решения которой применяются алгоритмы информационного множества (ISD).

Алгоритм Думера [3] представляет собой эффективный ISD-алгоритм, использующий парадокс дней рождения [4] для повышения производительности декодирования. Несмотря на теоретическую эффективность, практическое применение алгоритма ограничено высокой вычислительной сложностью, особенно для криптографически значимых параметров. Основные вычислительные затраты связаны с поиском совпадений в больших списках векторов, что обладает значительным потенциалом для параллелизации.

В данной работе исследуются три подхода к параллелизации алгоритма Думера: многопоточная реализация с использованием OpenMP, GPU-ускорение на основе CUDA, и гибридный CPU-GPU подход. Экспериментальные результаты показывают, что для малых матриц наилучшую производительность демонстрирует OpenMP реализация, в то время как для больших матриц оптимальной является гибридная реализация.

Основной вклад работы: разработка и сравнительный анализ эффективных параллельных реализаций алгоритма Думера, позволяющих значительно ускорить криптоанализ кодовых криптосистем и более точно оценивать их реальную стойкость в условиях доступности современных вычислительных ресурсов.

Статья получена 4 Июнь 2025
Чжай Хао, МГУ им. М.В. Ломоносова, (email: zh200053@gmail.com).

I. Алгоритм Думера и методы решения задачи синдромного декодирования

Задача синдромного декодирования (SDP) формулируется как поиск вектора ошибки $e \in \{0,1\}^n$ заданного веса w такого, что $eH^T = s$, где H — проверочная матрица линейного кода, а s — известный синдром. Данная задача является NP-полной [2] и лежит в основе безопасности кодовых криптосистем.

A. Эволюция алгоритмов информационного множества

Для решения задачи SDP разработан класс алгоритмов информационного множества (ISD), которые последовательно улучшали вычислительную эффективность:

Алгоритм Prange [5] представил базовый подход, основанный на приведении матрицы H к систематической форме и переборе информационных множеств.

Алгоритм Lee-Brickell [6] улучшил метод Prange, допуская небольшое количество ошибок в информационных позициях, что снизило количество требуемых итераций.

Алгоритм Stern [7] впервые применил парадокс дней рождения к задаче декодирования, разделяя задачу на подзадачи меньшего размера и используя поиск коллизий для их решения.

B. Алгоритм Думера

Алгоритм Думера [3] представляет собой оптимизированную версию алгоритма Stern с теоретически обоснованным выбором параметров. Основные инновации включают:

1) **Оптимальный выбор параметров**: Думер предложил математически обоснованный метод выбора параметров p и ℓ, минимизирующих общую вычислительную сложность.

2) **Улучшенная применение парадокса дней рождения**: Алгоритм эффективно разделяет целевой вес ошибки между различными частями матрицы, создавая два списка кандидатов для поиска коллизий.
3) **Оптимизированная сложность**: Обеспечивает существенное снижение вычислительной сложности по сравнению с более ранними ISD алгоритмами за счет оптимального выбора параметров.

C. Структура алгоритма Думера

Алгоритм работает в несколько ключевых этапов:

1) **Преобразование матрицы**: Матрица H приводится к специальной форме
$$UHP = \begin{pmatrix} I_{n-k-\ell} & 0 & H'' \\ 0 & I_\ell & H' \end{pmatrix}$$
где U — обратимая матрица, P — матрица перестановки.
2) **Решение подзадачи**: Применяется метод парадокса дней рождения для решения уравнения $e'H'^T = s'$ с весом $\mathrm{wt}(e') = p$.
3) **Проверка решения**: Найденные кандидаты проверяются на соответствие исходной задаче.

D. Вычислительная сложность

Временная сложность одной итерации алгоритма составляет:
$$K = n(n-k-\ell) + 2\sqrt{\binom{k+\ell}{p}} + \frac{\binom{k+\ell}{p}}{2^\ell}$$
где первое слагаемое отвечает за гауссово исключение, второе — за метод парадокса дней рождения, третье — за финальную проверку.

Вероятность успеха одной итерации:
$$P_\infty = \frac{\binom{k+\ell}{p}\binom{n-k-\ell}{w-p}}{\binom{n}{w}}$$

E. Потенциал для параллелизации

Алгоритм Думера обладает значительным потенциалом для параллелизации по нескольким направлениям:

- **Независимость итераций**: Различные итерации алгоритма статистически независимы
- **Параллелизуемые операции**: Генерация списков кандидатов и поиск коллизий естественным образом распараллеливаются
- **Масштабируемость**: Вычислительная нагрузка растет с размером задачи, что делает параллелизацию более эффективной для больших матриц

Именно эти характеристики мотивируют разработку эффективных параллельных реализаций алгоритма с использованием современных архитектур многоядерных процессоров и графических ускорителей.

II. Базовая реализация алгоритма Думера

Для реализации алгоритма Думера [3] был использован язык C++17, обеспечивающий эффективное управление памятью и оптимальную производительность.

A. Структура алгоритма

Реализация следует классической структуре алгоритма Думера:

1) **Преобразование матрицы**: Приведение матрицы H к специальной форме посредством умножения на матрицы U и P:
$$UHP = \begin{pmatrix} I_{n-k-l} & 0 & H'' \\ 0 & I_l & H' \end{pmatrix}$$
2) **Применение парадокса дня рождения**: Решение подзадачи $e'H'^T = s'$ путем разделения матрицы H' на компоненты H_1 и H_2 и формирования списков:
$$L_1 = \{e_1 H_1^T \mid wt(e_1) = p/2\}$$
$$, \quad L_2 = \{s' + e_2 H_2^T \mid wt(e_2) = p/2\}$$
3) **Верификация решения**: Проверка элементов пересечения $L_1 \cap L_2$ на соответствие исходной задаче $eH^T = s$.

B. Ключевые технические решения

1) Структуры данных: Для представления бинарных матриц и векторов используются контейнеры STL:

- `vector<vector<bool>>` для матриц
- `vector<bool>` для векторов
- `unordered_map` для эффективного поиска совпадений между L_1 и L_2

2) Оптимизация памяти: Применены методы для эффективного использования памяти:

- Генераторы с обратными вызовами вместо хранения всех векторов веса p
- Хеш-таблицы для поиска коллизий

C. Выбор параметров

Производительность алгоритма зависит от параметров p и l, которые определяются автоматически функцией `calculateOptimalParameters`.

Вероятность успеха на одной итерации:
$$P_\infty = \frac{\binom{k+l}{p} \cdot \binom{n-k-l}{w-p}}{\binom{n}{w}} \tag{1}$$

Общая вычислительная сложность:
$$WF_{SD} = \min_{0 \le p \le w} \frac{\binom{n}{w}}{\binom{n-k-l}{w-p}} \cdot \frac{1}{\sqrt{\binom{k+l}{p}}} \tag{2}$$
при условии $l = \log_2 \sqrt{\binom{k+l}{p}}$.

Данная базовая реализация служит основой для сравнения эффективности параллельных версий алгоритма.

III. Параллельная реализация алгоритма Думера

Для эффективного решения задачи синдромного декодирования в условиях современных вычислительных архитектур была разработана параллельная реализация алгоритма Думера [3]. Представлены три подхода к параллелизации: с использованием OpenMP [8] для многопоточного выполнения на CPU, с применением технологии CUDA [9] для GPU-ускорения и гибридная реализация, сочетающая преимущества обоих подходов.

A. Многопоточная реализация с использованием OpenMP

OpenMP (Open Multi-Processing) [8] представляет собой прикладной программный интерфейс для параллельного программирования с общей памятью. Эта технология была выбрана благодаря относительной простоте реализации и хорошей масштабируемости на многоядерных процессорах.

1) Архитектура параллелизации: В основе многопоточной реализации лежит концепция пула потоков, который управляет выполнением независимых итераций алгоритма Думера. Каждая итерация включает:

1) Выбор случайной матрицы перестановки
2) Преобразование матрицы H в специальную форму
3) Решение подзадачи синдромного декодирования

Такие итерации статистически независимы друг от друга, что делает их идеальными кандидатами для параллельной обработки.

2) Синхронизация и управление: Для корректной синхронизации доступа к общим ресурсам используются:

- **Атомарные переменные** для безопасного обновления счетчиков и флагов состояния
- **Критические секции** для обеспечения корректного завершения всех потоков при нахождении решения
- **Барьерная синхронизация** для координации этапов алгоритма

B. GPU-ускорение с использованием CUDA

Технология CUDA (Compute Unified Device Architecture) [10] от NVIDIA предоставляет возможность существенного ускорения вычислительно-интенсивных операций алгоритма Думера за счет использования массивного параллелизма графических процессоров.

1) Программная модель: Реализация алгоритма Думера на CUDA основана на разделении вычислений между CPU и GPU:

CPU (хост) управляет общим потоком выполнения, подготовкой данных и вызовом CUDA-ядер.

GPU (устройство) выполняет вычислительно-интенсивные операции, включая:

- Генерацию векторов заданного веса
- Вычисление синдромов (умножение вектора на матрицу)
- Поиск коллизий (парадокс дней рождения)

2) Ключевые CUDA-ядра: Архитектура реализации включает специализированные ядра для:

- `hammingWeightKernel` – вычисление веса Хэмминга векторов
- `vectorMatrixMultiplyKernel` – умножение вектора на матрицу
- `generateCombinationsKernel` – генерация комбинаций заданного веса
- `checkCollisionsKernel` – поиск коллизий между списками синдромов

3) Оптимизация памяти: Применены техники оптимизации для эффективного управления памятью GPU:

- Рациональное размещение данных для улучшения доступа к памяти GPU
- Минимизация передачи данных с использованием потоков CUDA для асинхронной передачи
- Использование разделяемой памяти для ускорения доступа к часто используемым данным

C. Гибридная параллельная стратегия

Гибридная реализация объединяет преимущества многопоточности CPU (OpenMP) и массивного параллелизма GPU (CUDA) для достижения максимальной производительности алгоритма.

1) Многоуровневый параллелизм: Архитектура включает несколько уровней параллелизма:

1) **Уровень задач**: Параллельные потоки CPU, каждый выполняющий независимые итерации алгоритма
2) **Уровень данных**: GPU-ускорение вычислительно-интенсивных операций внутри каждой итерации
3) **Уровень инструкций**: Оптимизации на уровне векторных инструкций для CPU и GPU

2) Распределение нагрузки: Структура взаимодействия организована по принципу ”производитель-потребитель”:

CPU выполняет:

- Общее управление алгоритмом
- Гауссово исключение для трансформации матрицы
- Подготовку данных для GPU
- Проверку и окончательную обработку найденных решений

GPU выполняет:

- Генерацию векторов заданного веса
- Вычисление синдромов
- Поиск коллизий методом парадокса дней рождения

3) Механизм совместной работы: Для эффективной совместной работы CPU и GPU применена модель асинхронного выполнения:

- Асинхронная генерация задач CPU-потоками независимо от их выполнения на GPU
- Конвейерная обработка с перекрытием вычислений и передачи данных
- Динамическая балансировка нагрузки в зависимости от производительности CPU и GPU

D. Оптимизация распределения ресурсов

Для максимальной эффективности использования вычислительных ресурсов применены подходы:

- Адаптивное распределение нагрузки с динамическим определением оптимального соотношения задач для CPU и GPU
- Мониторинг загрузки для оптимизации распределения
- Предотвращение конфликтов путем разделения работы для минимизации конкуренции за ресурсы

IV. Верификация эффективности алгоритма

Для подтверждения корректности реализованных нами различных версий алгоритма Думера[3], в этой главе используется стандартизированный общедоступный набор данных с сайта decodingchallenge.org[11].

A. Данные и методы верификации

1) Набор данных Decoding Challenge: Decoding Challenge[11] — это общедоступная платформа, созданная Nicolas Aragon, Julien Lavauzelle и Matthieu Lequesne в 2019 году, предоставляющая исследователям

стандартизированные наборы тестовых данных для оценки алгоритмов декодирования. Эта платформа предлагает примеры линейных кодов с различными параметрами, включая проверочные матрицы H и векторы синдромов s в стандартном формате.

2) Критерии проверки корректности: Для каждого решения e, найденного с помощью любой из реализаций, мы применяем следующие критерии проверки его корректности:

- Уравнение декодирования выполняется: $eH^T = s$
- Вес вектора ошибок соответствует требованиям: $\mathrm{wt}(e) = w$

Важно отметить, что для одной и той же задачи декодирования может существовать несколько корректных решений, удовлетворяющих данным условиям. Поэтому разные реализации могут находить разные решения, что является нормальным явлением.

B. Выбор тестовых примеров

Из набора данных Decoding Challenge мы выбрали 8 тестовых примеров малого размера с параметрами в диапазоне $n \in [20, 100]$:

Таблица I
Параметры тестовых примеров

Параметры (n, k, w)	Параметры (p, l)	Памяти (МБ)
$(20, 10, 5)$	$(5, 9)$	4.34
$(30, 15, 7)$	$(7, 14)$	4.58
$(40, 20, 8)$	$(8, 19)$	8.03
$(50, 25, 9)$	$(9, 24)$	18.59
$(60, 30, 10)$	$(10, 29)$	94.84
$(70, 35, 11)$	$(11, 34)$	466.00
$(80, 40, 12)$	$(12, 39)$	1677.62
$(100, 50, 14)$	$(14, 49)$	9837.75

Как показано в таблице, потребность в памяти алгоритма Думера [3] экспоненциально возрастает с увеличением размеров матрицы. Для малых экземпляров $(20, 10, 5)$, требования к памяти составляют всего 4.34 МБ, в то время как для матриц размером $(100, 50, 14)$ требования к памяти достигают примерно 9.8 ГБ. Важно отметить, что хотя алгоритм Думера успешно снижает временную сложность вычислений, он не оптимизирует использование памяти. Из-за ограничений памяти используемого экспериментального оборудования, данное исследование могло тестировать только экземпляры с параметрами $n \leq 100$, и не имело возможности провести тестирование матриц большего размера.

C. Результаты верификации

Для каждого тестового примера мы применили четыре реализации алгоритма (базовую последовательную, OpenMP, CUDA и гибридную) для декодирования и проверили, могут ли они найти действительное решение. Результаты представлены ниже:

Для каждого примера мы выполнили следующий процесс верификации:

1) Вычисление eH^T с использованием полученного вектора e
2) Проверка, полностью ли совпадает результат вычисления с заданным синдромом s
3) Вычисление веса Хэмминга вектора e и подтверждение, что он равен целевому значению w

n	Последовательная	OpenMP	CUDA	Гибридная
20	< 1	< 1	1093	67
30	8	8	1407	64
40	8	16	1678	62
50	81	83	6389	179
60	648	449	3784	259
70	16914	7848	4935	1347
80	23489	19312	5623	1422
100	39742	32178	7892	2283

Таблица II
Результаты проверки эффективности алгоритма (ms)

D. Выводы по результатам верификации

Результаты верификации показывают, что все четыре реализации алгоритма (базовая последовательная, OpenMP, CUDA и гибридная) способны находить действительные решения, удовлетворяющие условиям задачи, для всех тестовых примеров, что подтверждает корректность и эффективность реализаций алгоритма.

V. Дизайн эксперимента и анализ результатов

A. Экспериментальная среда и настройки

Для оценки влияния стратегий параллелизации на производительность алгоритма декодирования информационных множеств Думера мы провели серию экспериментов на следующей аппаратной платформе:

- **Процессор**: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz (6 ядер, 12 потоков)
- **Оперативная память**: 16GB
- **GPU**: NVIDIA GeForce GTX 1650 (4GB GDDR5)
- **Операционная система**: Ubuntu 20.04 LTS
- **Среда компиляции**: GCC 13.2.0, CUDA 12.8, OpenMP 5.0

Весь код реализован на C++ и скомпилирован с уровнем оптимизации -O3.

Следует отметить, что для матриц одинакового размера время решения алгоритмом не является постоянным (поскольку матрица P выбирается случайным образом при каждом запуске). В некоторых случаях решение может быть получено за одно вычисление, в то время как в других случаях даже после 100 попыток результат может быть не достигнут. Кроме того, производительность алгоритма значительно варьируется в зависимости от параметров p и l. Следовательно, для более точной оценки эффекта ускорения различных версий алгоритма, было измерено время, необходимое каждой из четырех версий для выполнения одной полной попытки решения при одинаковых параметрах $(p = 4,\ l = 4)$.

Для обеспечения достоверности результатов каждый эксперимент повторялся 5 раз, а среднее время выполнения использовалось в качестве окончательного результата.

B. Реализация алгоритма и методика тестирования

Мы реализовали и сравнили четыре различные версии алгоритма Думера [3]:

- **Dumer**: Базовая реализация, последовательное выполнение в одном потоке
- **Dumer-OpenMP**: Параллелизация на CPU с использованием OpenMP

- **Dumer-CUDA**: Выгрузка вычислительно-интенсивных задач на GPU с использованием CUDA
- **Dumer-Hybrid**: Гибридный подход, сочетающий вычисления на CPU и GPU

Для тестирования использовались проверочные матрицы различных размеров (от 10×10 до 900×900), что позволило оценить производительность в широком диапазоне сложности задач.

C. Анализ производительности для матриц большого размера

Результаты экспериментов для матриц большого размера ($N \geq 100$) демонстрируют значительные преимущества параллельных реализаций, как показано на рисунках 1 и 2.

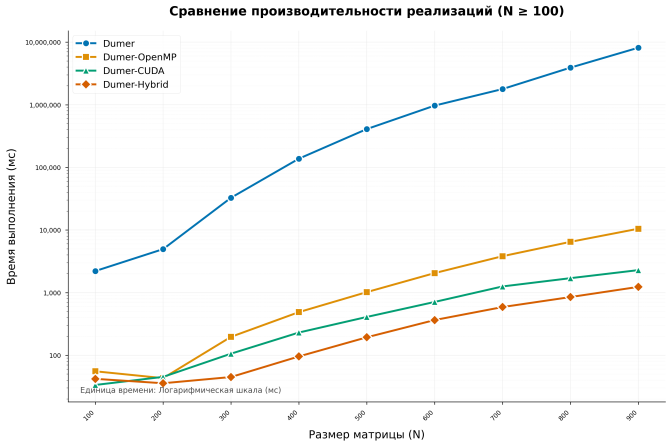

Рис. 1. Сравнение времени выполнения различных реализаций для больших матриц ($N \geq 100$)

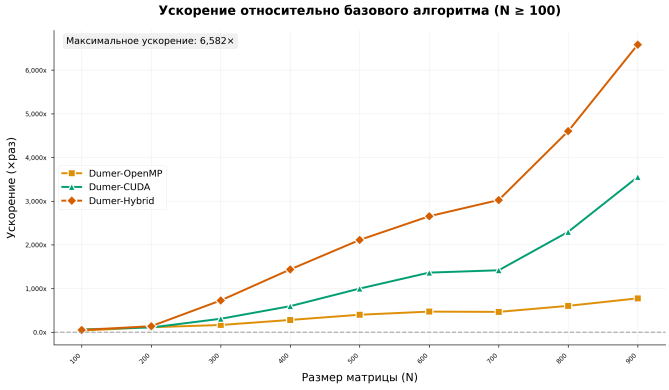

Рис. 2. Ускорение различных реализаций относительно базового алгоритма для больших матриц ($N \geq 100$)

Ключевые наблюдения для матриц большого размера:

- Гибридная параллельная версия (Dumer-Hybrid) показывает наилучшие результаты, достигая примерно 6582-кратного ускорения при $N = 900$, сокращая время выполнения с 8137,9 секунд до 1,2 секунды
- Версия с ускорением GPU (Dumer-CUDA) демонстрирует хорошую масштабируемость на крупномасштабных задачах, с ростом размера матрицы ускорение растет сверхлинейно
- Версия OpenMP, хотя и не достигает производительности других параллельных реализаций, все же обеспечивает примерно 800-кратное ускорение

D. Анализ производительности для матриц малого размера

Для матриц малого размера ($N < 100$) результаты демонстрируют иную картину, как показано на рисунках 3 и 4.

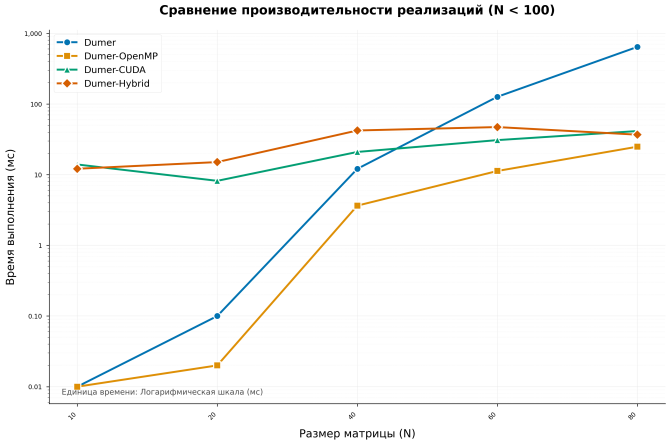

Рис. 3. Сравнение времени выполнения различных реализаций для малых матриц ($N < 100$)

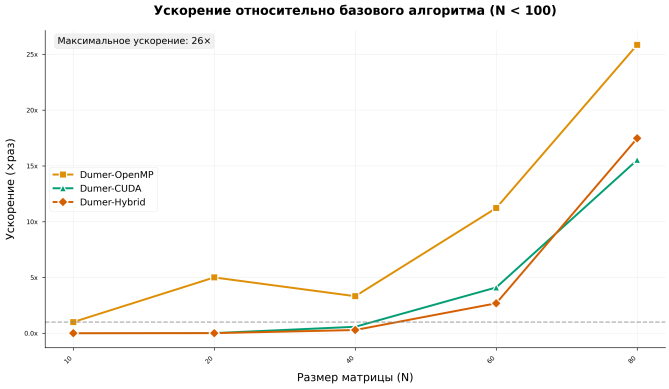

Рис. 4. Ускорение различных реализаций относительно базового алгоритма для малых матриц ($N < 100$)

Ключевые наблюдения для матриц малого размера:

- При $N < 40$ преимущества гибридной параллельной реализации не очевидны, и даже может наблюдаться снижение производительности из-за накладных расходов на инициализацию параллельных ресурсов
- Версия OpenMP показывает выдающиеся результаты на малых матрицах, достигая максимального ускорения примерно в 26 раз
- По мере увеличения размера задачи преимущества версий GPU и гибридной постепенно становятся более заметными

E. Сравнительный анализ реализаций при различных размерах матриц

Для наглядного сравнения производительности всех реализаций при различных размерах матриц, рисунок 5

представляет результаты для нескольких типичных значений:

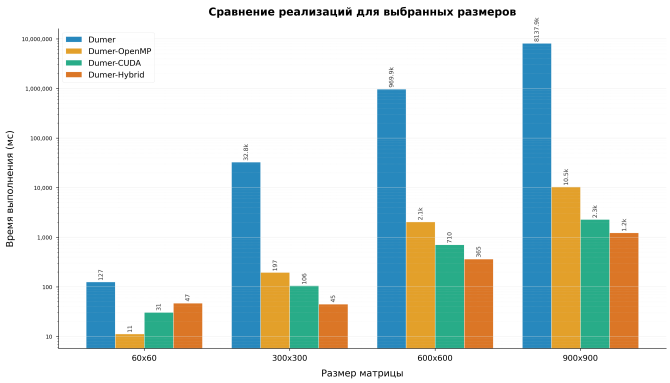

Рис. 5. Сравнение времени выполнения для типичных размеров матриц

Данный график наглядно демонстрирует, что с увеличением размера матрицы различия в производительности между реализациями становятся все более выраженными. При размере матрицы 900×900 время выполнения базовой реализации измеряется тысячами секунд, в то время как гибридная реализация выполняет те же вычисления за считанные секунды.

Заключение и перспективы

Основные результаты

Проведенное исследование демонстрирует значительный потенциал параллельных вычислений для оптимизации алгоритма декодирования информационных множеств Думера. Основные результаты:

1) Реализованы и сравнены четыре версии алгоритма: базовая последовательная, OpenMP-оптимизированная для CPU, CUDA-оптимизированная для GPU и гибридная версия
2) Гибридная реализация достигает наивысшей производительности на больших матрицах, обеспечивая ускорение более чем в 1000 раз по сравнению с базовой реализацией
3) Для матриц малого размера ($N < 100$) OpenMP-реализация более эффективна из-за меньших накладных расходов на инициализацию GPU
4) Выявлены ключевые узкие места производительности для каждой реализации

Результаты показывают, что параллельная оптимизация алгоритма Думера [3] значительно повышает его практическую применимость — задачи, требовавшие часов вычислений, теперь решаются за секунды.

Рекомендации по применению

На основе результатов исследования сформулированы рекомендации по выбору оптимальной реализации:

1) **Для малых матриц** ($N < 40$): OpenMP-реализация обеспечивает хорошее ускорение без накладных расходов на инициализацию GPU
2) **Для матриц среднего и большого размера** ($N \geq 40$): Гибридная реализация обеспечивает наилучший баланс эффективности

Направления дальнейших исследований

Результаты открывают перспективные направления для дальнейшей работы:

1) Оптимизация структур данных и шаблонов доступа к памяти [12] для улучшения локальности данных
2) Расширение алгоритма для распределенных вычислений [13] с использованием MPI для решения крупномасштабных задач
3) Исследование специализированных аппаратных ускорителей [14], таких как FPGA или ASIC
4) Разработка адаптивных алгоритмов [15], динамически адаптирующих стратегию параллелизации

Заключение

Исследование демонстрирует, что современные методы параллельных вычислений значительно расширяют границы практического применения алгоритмов декодирования информационных множеств. Стратегия многоуровневой параллелизации эффективна для алгоритмов с высокой вычислительной сложностью и открывает новые возможности применения в различных областях науки и техники.

Продолжение исследований в области оптимизации структур данных, шаблонов доступа к памяти и балансировки нагрузки может привести к ещё более значительным улучшениям производительности алгоритма декодирования Думера [3].

Библиография

[1] R. J. McEliece, «A public-key cryptosystem based on algebraic,» *Coding Thv*, т. 4244, с. 114—116, 1978.
[2] E. Berlekamp, R. McEliece и H. Van Tilborg, «The inherent computational complexity of decoding,» *IEEE Transactions on Information Theory*, 1978.
[3] I. Dumer, «On minimum distance decoding of linear codes,» в *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, Moscow, 1991, с. 50—52.
[4] D. Wagner, «A generalized birthday problem,» в *Annual International Cryptology Conference*, Springer, 2002, с. 288—304.
[5] E. Prange, «The use of information sets in decoding cyclic codes,» *IRE Transactions on Information Theory*, 1962.
[6] P. J. Lee и E. F. Brickell, «Some algorithms for soft-decision decoding,» *IEEE Transactions on Information Theory*, 1988.
[7] J. Stern, «A method for finding codewords of small weight,» *Coding Theory and Applications*, 1989.
[8] OpenMP Architecture Review Board, *OpenMP Application Programming Interface Version 5.0*, https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf, Accessed: May 2025, 2018.
[9] NVIDIA Corporation, *CUDA C Programming Guide*, https://docs.nvidia.com/cuda/cuda-c-programming-guide/, Version 12.3, Accessed: May 2025, 2023.
[10] J. Nickolls, I. Buck, M. Garland и K. Skadron, «Scalable parallel programming with CUDA,» *Queue*, т. 6, № 2, с. 40—53, 2008.

[11] N. Aragon, J. Lavauzelle и M. Lequesne, *decodingchallenge.org*, 2019. url: http : / / decodingchallenge.org.

[12] V. Volkov и J. W. Demmel, «Benchmarking GPUs to tune dense linear algebra,» в *SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE, 2008, с. 1—11.

[13] W. Gropp, E. Lusk, N. Doss и A. Skjellum, «A high-performance, portable implementation of the MPI message passing interface standard,» *Parallel computing*, т. 22, № 6, с. 789—828, 1996.

[14] T. Güneysu, «High-speed cryptography and cryptanalysis on FPGAs,» в *Applied Reconfigurable Computing*, Springer, 2016, с. 217—228.

[15] J. Subhlok, S. Venkataramaiah и A. Singh, «Automatic node selection for high performance applications on networks,» с. 163—172, 2001.

Software Implementation of Dumer's Algorithm for Decoding Binary Linear Codes in Various Parallel Computing Models

Zhai Hao

***Abstract*—This paper investigates and implements parallel versions of Dumer's algorithm for solving the NP-complete Syndrome Decoding Problem (SDP), which forms the security foundation of post-quantum code-based cryptosystems like McEliece. Despite its theoretical efficiency, the practical application of Dumer's algorithm is limited by its high computational complexity. To overcome this barrier, four versions of the algorithm were developed and rigorously analyzed: a baseline sequential C++ implementation, a multi-threaded version using OpenMP for multi-core CPUs, a GPU-accelerated version based on CUDA, and a hybrid model combining the computational power of both CPUs and GPUs. The correctness of all implementations was verified using standardized test data from the decodingchallenge.org platform. Experimental results demonstrate that for small-scale matrices, the OpenMP implementation is the most effective, whereas for large-scale problems, the hybrid approach shows superior performance. This study significantly enhances the practical applicability of Dumer's algorithm, greatly reducing the problem-solving time and allowing for a more accurate assessment of the real-world security of code-based cryptosystems.**

***Keywords*—Dumer's algorithm, syndrome decoding, parallel computing, OpenMP, CUDA, post-quantum cryptography**

References

[1] R. J. McEliece, «A public-key cryptosystem based on algebraic», *Coding Thv*, vol. 4244, pp. 114–116, 1978.

[2] E. Berlekamp, R. McEliece, and H. Van Tilborg, «The inherent computational complexity of decoding», *IEEE Transactions on Information Theory*, 1978.

[3] I. Dumer, «On minimum distance decoding of linear codes», in *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, Moscow, 1991, pp. 50–52.

[4] D. Wagner, «A generalized birthday problem», in *Annual International Cryptology Conference*, Springer, 2002, pp. 288–304.

[5] E. Prange, «The use of information sets in decoding cyclic codes», *IRE Transactions on Information Theory*, 1962.

[6] P. J. Lee and E. F. Brickell, «Some algorithms for soft-decision decoding», *IEEE Transactions on Information Theory*, 1988.

[7] J. Stern, «A method for finding codewords of small weight», *Coding Theory and Applications*, 1989.

[8] OpenMP Architecture Review Board, *OpenMP application programming interface version 5.0*, https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf, Accessed: May 2025, 2018.

[9] NVIDIA Corporation, *CUDA C programming guide*, https://docs.nvidia.com/cuda/cuda-c-programming-guide/, Version 12.3, Accessed: May 2025, 2023.

[10] J. Nickolls, I. Buck, M. Garland, and K. Skadron, «Scalable parallel programming with CUDA», *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[11] N. Aragon, J. Lavauzelle, and M. Lequesne, *Decodingchallenge.org*, 2019. [Online]. Available: http://decodingchallenge.org.

[12] V. Volkov and J. W. Demmel, «Benchmarking GPUs to tune dense linear algebra», in *SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, IEEE, 2008, pp. 1–11.

[13] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, «A high-performance, portable implementation of the MPI message passing interface standard», *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.

[14] T. Güneysu, «High-speed cryptography and cryptanalysis on FPGAs», in *Applied Reconfigurable Computing*, Springer, 2016, pp. 217–228.

[15] J. Subhlok, S. Venkataramaiah, and A. Singh, «Automatic node selection for high performance applications on networks», pp. 163–172, 2001.