**РЕАЛИЗАЦИЯ ДИСКРЕТНОГО ПРЕОБРАЗОВАНИЯ ХАРТЛИ НА МНОГОЯДЕРНЫХ УСКОРИТЕЛЯХ**

**Сергей Сергеевич Арсеньев-Образцов1, Волков Евгений Александрович2**

*1,2РГУ нефти и газа (НИУ) имени И.М. Губкина, г. Москва, Россия*

1arseniev@gubkin.ru

**Аннотация**

В статье представлены два подхода к параллельной реализации многомерного дискретного преобразования Хартли (ND-DHT). В первом случае рассматривается использование быстрого преобразования Хартли (FDHT), во втором – его матричное представление. Описан широкий спектр задач, для которого имеет смысл использовать предложенные методы. Рассмотрены преимущества и недостатки FDHT относительно алгоритма быстрого преобразования Фурье (Real-FFT). Приведены результаты тестирования параллельных алгоритмов (с использованием Fortran/C+OpenMP для CPU и Fortran/С+CUDA для GPGPU) на наборах данных разных размерностей: для изображений с разрешением от 128 × 128 до 4096 × 4096.

**Ключевые слова:** многомерное дискретное преобразование Хартли, быстрое преобразование Хартли, высокопроизводительные вычисления, OpenMP, CUDA, обработка изображений, GPGPU.

1. **Введение**

Дискретные преобразования – один из важнейших и широко используемых инструментов в области цифровой обработки сигналов. Они возникают в качестве дискретного аналога непрерывных интегральных преобразований. Наиболее известными и распространенными преобразованиями являются дискретное преобразование Фурье (DFT) и быстрое преобразование Фурье (FFT). В частности, DFT используется в таких важнейших областях, как сжатие аудио-данных (MP3) [1], спектральный анализ геоданных [2], корреляционный анализ и др. Их альтернативой является дискретное косинус преобразование (DCT), которое применяется в алгоритмах сжатия информации MPEG и JPEG.

Еще одним известным, но редко используемым преобразованием, является дискретное преобразование Хартли (DHT). В отличие от преобразования Фурье, оно вычисляется только в действительной области, что существенно повышает эффективность обработки больших данных, снижая требования к оперативной памяти. Одномерное (1D) дискретное преобразование Хартли имеет вид:

|  |  |
| --- | --- |
|  | (1) |

где [3], *H* – результат применения DHT ко входному дискретному сигналу *h* длиной *N*. Как и для DFT, вычисление DHT требует арифметических операций, где – размерность массива входных данных. Однако существуют *быстрые алгоритмы*, подобные FFT, при вычислении которых необходимо выполнить операций. Почти все известные алгоритмы вычисления FFT, например, алгоритм Cooley-Tukey, имеют прямые аналоги при реализации быстрого преобразования Хартли (FDHT).

Для обработки данных, не имеющих комплексной части, также были разработаны специальные алгоритмы *действительного* преобразования Фурье (Real-FFT). Они основаны на "эрмитовой" симметрии результата преобразования: , где *N* – длина обрабатываемого вектора. Этот факт позволяет улучшить производительность алгоритма примерно в два раза, за счёт уменьшения количества вычислительных операций и оптимизации использования памяти [4].

В работе [5] P. Duhamel и M. Vetterli показали, что для обработки сигналов, представленных в виде массивов чисел с плавающей запятой, процедуры FDHT несколько менее эффективны, чем аналогичные алгоритмы Real-FFT. Они показали, что FDHT, в лучшем случае, можно вычислить, совершив, по крайней мере, на две операции сложения больше, чем в соответствующем аналоге Real-FFT. Но с учетом архитектуры современных много/мульти ядерных процессоров производительность алгоритма, применяемого для анализа больших данных, больше определяется работой с многоуровневым кэшем вычислительных ядер (с учетом алгоритма предугадывания), а также работой с конвейерами или векторными регистрами ядер, чем простым подсчётом количества операций, что практически нивелирует разницу между Real-FFT и FDHT. В более поздней работе [6] M. Popovic и D. Sevic, исследуя производительность FFT и FDHT, пришли к выводу, что из-за схожести вычислительных структур алгоритмов на практике ни одно из преобразований не имеет существенного преимущества в производительности.

Одномерные дискретные преобразования сигналов являются только небольшой частью области применения дискретных преобразований. Чаще всего указанные алгоритмы используются в области обработки многомерных изображений, источником которых являются фотографии с мобильного телефона, видеотрансляции спортивных соревнований, системы дополненной и виртуальной реальности, данные с медицинских 3D сканеров, томограммы, полученные с помощью микро-компьютерной томографии (МКТ). Это лишь небольшой список примеров задач, где используются многомерные дискретные преобразования. Из-за специфики получения этих данных (с помощью фотоаппарата, видеокамеры, компьютерного томографа и т.п.) их разрешение почти всегда находится в определенном интервале – по каждому измерению от нескольких тысяч до десятков тысяч пикселей (вокселей). В связи с тем, что на практике приходится обрабатывать существенный объем таких медиа-данных (иногда это тысячи изображений, т.е. десятки и сотни гигабайт), при вычислении дискретных многомерных преобразований возникает необходимость использовать аппарат распараллеливания, учитывающий архитектуру современных процессоров CPU и различных ускорителей, например, GPGPU.

Примером таких задач, широко распространённых в последнее время, является обработка наборов данных для обучения моделей нейронных сетей. Мы считаем, что это направление применения дискретных преобразований в области искусственного интеллекта использует наибольшую часть имеющихся в мире вычислительных ресурсов. Например, широко известный набор данных EuRoC MAV, содержащий стереоизображения, синхронизированные с измерениями датчиков IMU, и использующийся для работы с моделями БПЛА [7], состоит из множества отснятых сцен, каждая из которых – это десятки тысяч изображений разрешением 1024х780. Общий объём данного набора данных около 50 ГБ.

Таким образом, можно сделать вывод, что в современной, недавно сформировавшейся науке о данных появился спектр задач со специфическими свойствами:

* Данные и результат состоят из действительных чисел;
* Массивы имеют размерность (разрешение), лежащую в интервале по каждому измерению (оси);
* Обычно они имеют ND интерпретацию (2D, 3D и т.д.).

Предлагается применять для обработки данных описанного типа дискретное преобразование Хартли. Для оценки его эффективности необходимо провести вычислительные эксперименты, рассмотрев несколько подходов к распараллеливанию. Наконец, важно сравнить полученные результаты с вычислением многомерных преобразований на основе Real-FFT, что является альтернативой предложенному методу.

1. **Вычисление дискретного N-мерного (ND) преобразования Хартли**
   1. **Представление ND преобразования как последовательность 1D преобразований**

Классическим подходом при вычислении дискретных ND преобразований, в том числе и преобразования Хартли, является представление ND преобразования в виде последовательности независимо вычисляемых 1D преобразований. Такой подход при его реализации позволяет выполнить тривиальное распараллеливание [8]. В случае вычисления преобразования Фурье это становится возможным из-за расщепления ядра преобразования:

|  |  |
| --- | --- |
| , | (2) |

где – размерность преобразования, – размерности преобразования вдоль каждой из осей, – вектор входных данных, – результат преобразования.

При вычислении преобразования Хартли применяются подстановки, предложенные R.N. Bracewell и H. Hao в работах [3] и [9]. Вариант в двумерном (2D) случае имеет вид:

|  |  |
| --- | --- |
|  | (3) |

из чего следует:

|  |  |
| --- | --- |
|  | (4) |

где, аналогично (2):

|  |  |
| --- | --- |
|  | (5) |

На Рисунке 1 приведена схема вычисления двумерного преобразования. Для вычисления 2D DHT для квадратной матрицы размерностью сначала выполняется 1D преобразований вдоль оси , а потом 1D преобразований вдоль оси либо наоборот (см. Рисунок 1).

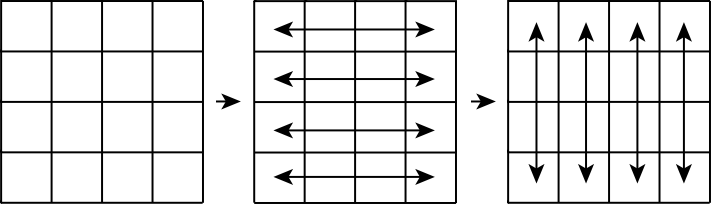


Рисунок 1. Нахождение 2D преобразования Хартли путём вычисления 1D преобразований Хартли вдоль оси Х, а затем – вдоль оси Y.

* 1. **Быстрые алгоритмы дискретных преобразований**

В большинстве приложений одномерные преобразования вычисляются с помощью быстрых алгоритмов [10]. Для вычисления обычного преобразования Хартли требуется арифметических операций. Однако быстрые алгоритмы, аналогичные Cooley-Tukey, позволяют провести вычисления за операций. Такое ускорение достигается, так или иначе, благодаря идее декомпозиции: входящий сигнал рекурсивно разбивается на два (или больше) подсигнала с последующим вычислением преобразования от каждого из них. По этой причине наибольшей производительности можно добиться только при обработке векторов длины . Существуют алгоритмы вычисления FFT и FDHT для последовательностей четных длин или, например, длин, являющихся простыми числами [11]. Однако в общем случае алгоритм быстрого преобразования не будет максимально эффективен.

* 1. **Матричное представление преобразования Хартли**

Альтернатива вычислению DHT основана на его матричном представлении:

|  |  |
| --- | --- |
|  | (6) |

где, – входной вектор исходных данных, – результат применения преобразования Хартли, – матрица Вандермонда:

|  |  |
| --- | --- |
|  | (7) |

Матричный способ представления преобразования даёт возможность использовать лучшие схемы распараллеливания задач на GPGPU [12]. Необходимо отметить, что при серии вычислений DHT одного и того же размера матрица (*7*) вычисляется лишь один раз – это имеет решающую роль при преобразовании Хартли многомерных массивов данных.

В результате, для вычисления дискретных преобразований (в частности, преобразования Хартли) для указанных специфических типов данных, представляющих собой несколько сотен/тысяч изображений (3D томография, наборы изображений для обучения нейронных сетей, различные 3D данные и данные больших размерностей – геология, геофизика (сейсмика), разработка больших месторождений углеводородного сырья с моделями, представленными мелкой гидродинамической сеткой, управление морскими платформами и т.п.), предлагается заменить использование быстрых 1D преобразований Хартли с использованием одномерных массивов на матричные одномерные преобразования Хартли, ориентированные на использование ускорителей, таких как, например, GPGPU.

* 1. **Распараллеливание**

При работе с вычислительными системами с распределённой памятью существует два подхода к распараллеливанию многомерных преобразований. Первый основывается на представлении ND преобразования в виде последовательности 1D преобразований (Раздел 2.1). В случае использования распределённой памяти этот подход требует доступа к одномерным массивам на вычислительных узлах (нодах, процессах) в определённые моменты времени, что осуществляется путём глобального перемещения данных (обычно операциями коллективного обмена типа *all-to-all*). В литературе такие подходы, применительно к дискретным преобразованиям, часто называют алгоритмами транспонирования. Альтернативным параллельным методом ND-FFT, ориентированным на внутреннюю структуру алгоритма FFT, является метод двоичного обмена. В данной работе мы рассматриваем только первый подход, который в целом считается более эффективным для решения больших задач. Сравнительный обзор обоих методов представлен в работе [13].

В случае обработки многомерных преобразований на системах с общей памятью нет необходимости в перераспределении данных между вычислителями, что существенно сокращает время вычислений, но ограничивает количество обрабатываемых данных.

Таким образом, при вычислении многомерных дискретных преобразований, в частности, преобразования Хартли, на различных вычислительных системах подходят несколько парадигм параллельного программирования:

1. Fortran/C+MPI+OpenACC/OpenMP: при использовании гетерогенных высокопроизводительных вычислительных систем с распределённой памятью, использующие процессоры IBM с ускорителями NVidia или AMD, а также спецпроцессоры типа Intel Xeon Phi или Sunway SW26010 [14];
2. Fortran/C+OpenMP: для систем с общей памятью, обычно ориентированных на мульти ядерные CPU для вычисления ND-DHT с помощью FDHT. Также можно использовать эту парадигму для программирования алгоритмов дискретных преобразований на систему CPU+ускоритель, используя последние расширения стандарта OpenMP;
3. Fortran/C+CUDA/OpenACC: в случае использования систем с общей памятью (ориентированных на GPGPU или другие ускорители) для вычисления ND-DHT, используя матричное представление одномерного преобразования.
4. **Результаты**

Эксперименты по определению производительности были выполнены на компьютере с процессором Intel(R) Core(TM) i7-7700K 4.20GHz, 64ГБ ОЗУ и графическим ускорителем NVIDIA GeForce GTX 1060 3 GB.

На Рисунке 2 представлены результаты эксперимента: для данных разных размерностей был выполнен ряд вычислительных испытаний, а именно, для *N* изображений, имеющих разрешения 128 × 128, 256 × 256, 512 × 512, 1024 × 1024, 2048 × 2048 и 4096 × 4096. Были вычислены 2D преобразования Фурье на основе Real-FFT, а также 2D преобразования Хартли, как на основе последовательности 1D быстрых преобразований, так и с помощью матричного представления преобразования.

Для вычисления FHT и Real-FFT были использованы классические итеративные реализации алгоритмов Cooley-Tukey, описанные в работе [15]. Для распараллеливания 2D-FHT и 2DReal-FFT была применена технология OpenMP (Open Multi-Processing) – набор директив расширения языка программирования прагмами и библиотеки функций для распараллеливания кода на мультипроцессорных системах с общей памятью, таких, как многоядерные процессоры и кластеры.

Для тестирования параллельного алгоритма на GPU была использована технология CUDA. Задача умножения матриц – базовая задача в параллельных вычислениях на GPGPU, по причине чего было разработано большое количество различных подходов к распараллеливанию. В нашей работе используется алгоритм произведения матриц, предложенный в [16], в котором ядро (функция, которая выполняется непосредственно на графическом ускорителе) работает на множестве потоков, каждый из которых вычисляет только один элемент результирующей матрицы.

На Рисунке 2 и Таблицах 1-6 продемонстрировано, что в случае обработки небольшого количества изображений эффективнее использовать FHT+OpenMP, но с ростом *N* лучше работает DHT+CUDA. Видно, что в зависимости от разрешения изображений – (a), (b), (c), (d), (e) и (f) – изменяется и порог, при котором счёт быстрее выполняется на GPU. Наконец, необходимо отметить, что при DHT+CUDA имеет близкую к линейной сложность, как и FHT+OpenMP, что становится возможным благодаря распараллеливанию на GPGPU.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество изображений, N | FHT+OpenMP, секунды | Real-FFT, секунды | DHT+CUDA, секунды |
| 500 | 0.084 | 0.083 | 0.101 |
| 1000 | 0.167 | 0.166 | 0.144 |
| 2000 | 0.343 | 0.362 | 0.209 |
| 5000 | 0.852 | 0.832 | 0.423 |
| 10000 | 1.678 | 1.663 | 0.766 |
| 25000 | 4.232 | 4.205 | 1.81 |
| 50000 | 8.656 | 9.902 | 3.613 |
| 100000 | 16.941 | 16.838 | 7.113 |

Таблица 1. Время обработки N изображений разрешением 128×128

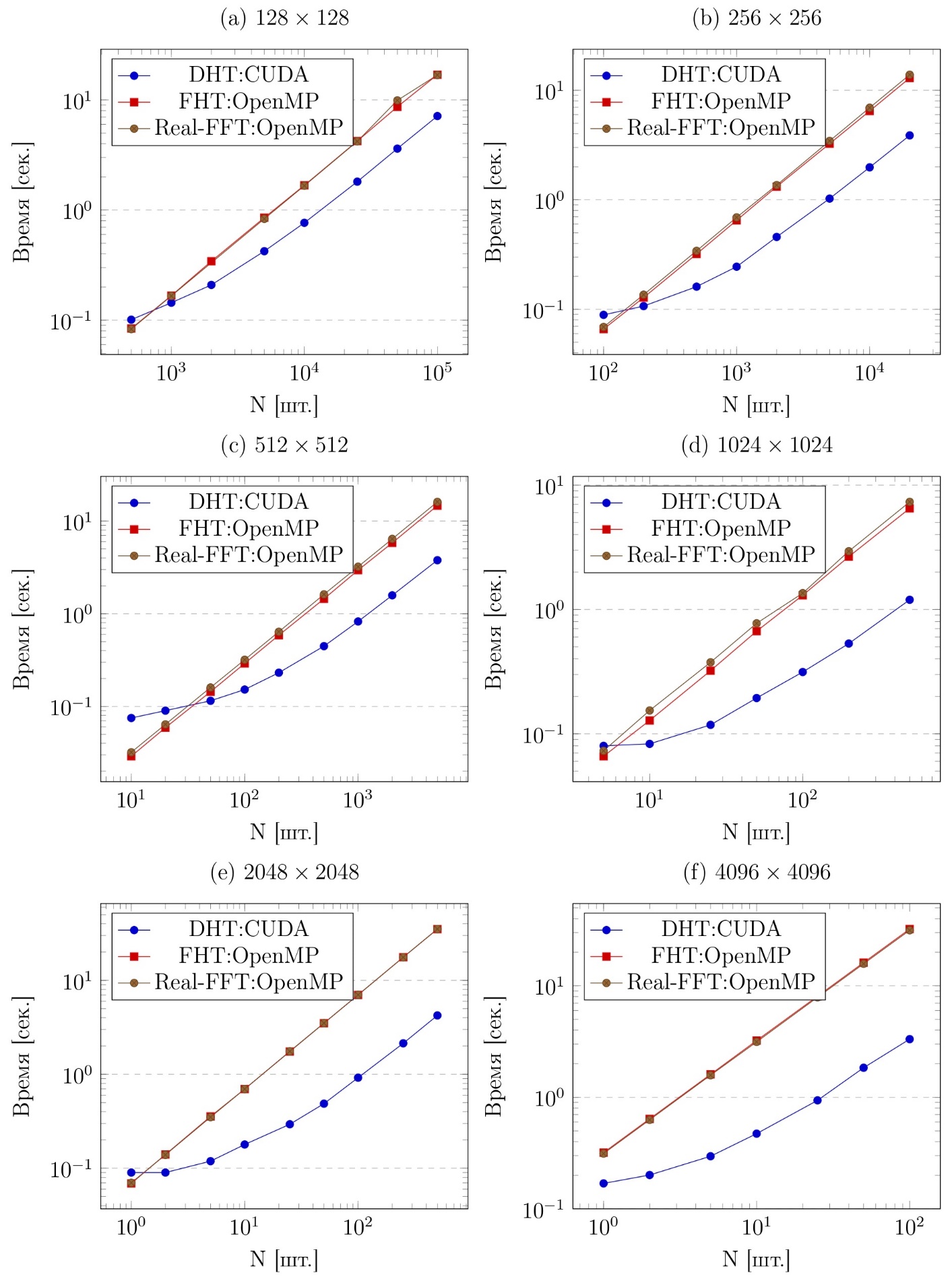


Рисунок 2. Время обработки N изображений разрешением (a)128×128, (b)256×256, (c)512×512, (d)1024×1024, (e)2048×2048, (f)4096×4096.

|  |  |  |  |
| --- | --- | --- | --- |
| Количество изображений, N | FHT+OpenMP, секунды | Real-FFT, секунды | DHT+CUDA, секунды |
| 100 | 0.066 | 0.069 | 0.089 |
| 200 | 0.128 | 0.136 | 0.107 |
| 500 | 0.32 | 0.342 | 0.161 |
| 1000 | 0.649 | 0.691 | 0.245 |
| 2000 | 1.319 | 1.362 | 0.458 |
| 5000 | 3.252 | 3.449 | 1.026 |
| 10000 | 6.468 | 6.929 | 1.979 |
| 20000 | 12.947 | 13.874 | 3.873 |

Таблица 2. Время обработки N изображений разрешением 256×256

|  |  |  |  |
| --- | --- | --- | --- |
| Количество изображений, N | FHT+OpenMP, секунды | Real-FFT, секунды | DHT+CUDA, секунды |
| 10 | 0.029 | 0.032 | 0.075 |
| 20 | 0.059 | 0.064 | 0.09 |
| 50 | 0.144 | 0.16 | 0.115 |
| 100 | 0.291 | 0.319 | 0.152 |
| 200 | 0.586 | 0.639 | 0.231 |
| 500 | 1.447 | 1.618 | 0.447 |
| 1000 | 2.953 | 3.217 | 0.825 |
| 2000 | 5.815 | 6.415 | 1.58 |
| 5000 | 14.676 | 16.102 | 3.78 |

Таблица 3. Время обработки N изображений разрешением 512×512

|  |  |  |  |
| --- | --- | --- | --- |
| Количество изображений, N | FHT+OpenMP, секунды | Real-FFT, секунды | DHT+CUDA, секунды |
| 5 | 0.066 | 0.073 | 0.08 |
| 10 | 0.128 | 0.154 | 0.083 |
| 25 | 0.322 | 0.376 | 0.118 |
| 50 | 0.669 | 0.774 | 0.194 |
| 100 | 1.3 | 1.353 | 0.314 |
| 200 | 2.659 | 2.94 | 0.532 |
| 500 | 6.501 | 7.336 | 1.197 |

Таблица 4. Время обработки N изображений разрешением 1024×1024

|  |  |  |  |
| --- | --- | --- | --- |
| Количество изображений, N | FHT+OpenMP, секунды | Real-FFT, секунды | DHT+CUDA, секунды |
| 1 | 0.069 | 0.07 | 0.09 |
| 2 | 0.14 | 0.139 | 0.09 |
| 5 | 0.356 | 0.35 | 0.119 |
| 10 | 0.697 | 0.697 | 0.179 |
| 25 | 1.747 | 1.757 | 0.294 |
| 50 | 3.5 | 3.507 | 0.487 |
| 100 | 7 | 7.013 | 0.921 |
| 250 | 17.633 | 17.6 | 2.138 |
| 500 | 35.092 | 35.171 | 4.233 |

Таблица 5. Время обработки N изображений разрешением 2048×2048

|  |  |  |  |
| --- | --- | --- | --- |
| Количество изображений, N | FHT+OpenMP, секунды | Real-FFT, секунды | DHT+CUDA, секунды |
| 1 | 0.319 | 0.313 | 0.169 |
| 2 | 0.64 | 0.628 | 0.201 |
| 5 | 1.605 | 1.578 | 0.296 |
| 10 | 3.226 | 3.138 | 0.472 |
| 25 | 8.058 | 7.884 | 0.939 |
| 50 | 16.133 | 15.75 | 1.841 |
| 100 | 32.282 | 31.46 | 3.323 |

Таблица 6. Время обработки N изображений разрешением 4096×4096

1. **Выводы**

В работе был предложен параллельный алгоритм реализации на GPGPU многомерного дискретного преобразования Хартли на основе матричного представления. В результате численного эксперимента можно сделать следующие выводы:

* при вычислении *N* 2D-DHT предложенным алгоритмом достигнута большая производительность, по сравнению с вычислением преобразования на основе одномерных FDHT;
* косвенно подтверждены результаты, полученные в работе [6], об отсутствии практического преимущества при вычислении FDHT и Real-FFT;
* по результатам экспериментов установлена близкая к линейной сложность при вычислении N 2D-DHT для указанного в статье типа данных.

Необходимо отметить, что для применения данного подхода на практике при обработке данных больших размерностей требуются дополнительные исследования, чтобы убедиться в его применимости к более сложным и эффективным системам, таким как вычислительные ускорители Intel Xeon Phi или суперкомпьютеры Sunway TaihuLight и ИИ-ориентированный Sunway Tianhe-2 на основе многоядерного микропроцессора SW26010.

1. **Список литературы**
2. Moon H. G. A low-complexity design for an mp3 multi-channel audio decoding system //IEEE transactions on audio, speech, and language processing. – 2011. – Т. 20. – №. 1. – С. 314-321.
3. Tahmasebi P., Sahimi M., Caers J. MS-CCSIM: accelerating pattern-based geostatistical simulation of categorical variables using a multi-scale search in Fourier space //Computers & Geosciences. – 2014. – Т. 67. – С. 75-88.
4. Bracewell R. N. et al. Fast two-dimensional Hartley transform //Proceedings of the IEEE. – 1986. – Т. 74. – №. 9. – С. 1282-1283.
5. Sorensen H. V. et al. Real-valued fast Fourier transform algorithms //IEEE Transactions on acoustics, speech, and signal processing. – 1987. – Т. 35. – №. 6. – С. 849-863.
6. Duhamel P., Vetterli M. Improved Fourier and Hartley transform algorithms: Application to cyclic convolution of real data //IEEE Transactions on Acoustics, Speech, and Signal Processing. – 1987. – Т. 35. – №. 6. – С. 818-824.
7. PopoviC M., SeviC D. A new look at the comparison of the fast Hartley and Fourier transforms //IEEE Transactions on Signal Processing. – 1994. – Т. 42. – №. 8. – С. 2178-2182.
8. Burri M. et al. The EuRoC micro aerial vehicle datasets //The International Journal of Robotics Research. – 2016. – Т. 35. – №. 10. – С. 1157-1163.
9. Dalcin L., Mortensen M., Keyes D. E. Fast parallel multidimensional FFT using advanced MPI //Journal of Parallel and Distributed Computing. – 2019. – Т. 128. – С. 137-150.
10. Hao H., Bracewell R. N. A three-dimensional DFT algorithm using the fast Hartley transform //Proceedings of the IEEE. – 1987. – Т. 75. – №. 2. – С. 264-266.
11. Wang Z. Fast algorithms for the discrete W transform and for the discrete Fourier transform //IEEE Transactions on Acoustics, Speech, and Signal Processing. – 1984. – Т. 32. – №. 4. – С. 803-816.
12. Rader C. M. Discrete Fourier transforms when the number of data samples is prime //Proceedings of the IEEE. – 1968. – Т. 56. – №. 6. – С. 1107-1108.
13. Bell N., Garland M. Efficient sparse matrix-vector multiplication on CUDA. – Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008. – Т. 2. – №. 5.
14. Foster I. T., Worley P. H. Parallel algorithms for the spectral transform method //SIAM Journal on Scientific Computing. – 1997. – Т. 18. – №. 3. – С. 806-837.
15. Arsenyev-Obraztsov S. S., Volkov E. A., Plusch G. O. Proposals on 3D parallel edge-preserving filtration for x-ray tomographic digital images of porous medium core plugs //IOP Conference Series: Materials Science and Engineering. – IOP Publishing, 2019. – Т. 700. – №. 1. – С. 012053.
16. Takahashi D. Fast Fourier transform algorithms for parallel computers. – Springer Singapore, 2019.
17. Тумаков Д. Н. и др. Технология программирования CUDA. – 2017.