УДК 004.272.2

ОБРАБОТКА БОЛЬШИХ МАССИВОВ ИЗМЕРИТЕЛЬНОЙ ИНФОРМАЦИИ С ПРИМЕНЕНИЕМ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ

Т.А. Ализаде
Заведующий лабораторией методов распознавания и интеллектуального анализа сигналов Института систем управления (Азербайджан), кандидат технических наук, доцент
tahiralizada1973@gmail.com

А.А. Мустафаев
Аспирант Института систем управления (Азербайджан)
akshinmustafayev@outlook.com

Т.А. Ализаде

Окончил Бакинский государственный университет. Область научных интересов связана с обработкой сигналов и разработкой методов и алгоритмов спектрального анализа, распознаванием технического состояния объектов нефтедобычи.

А.А. Мустафаев

Окончил Азербайджанский государственный университет нефти и промышленности. Область научных интересов связана с разработкой алгоритмов и программ распараллеливания вычисления спектральных характеристик сигналов.

Аннотация. В настоящее время использование параллельных вычислений в информационных системах становится необходимостью, особенно когда требуется обработка больших объёмов информации в режиме реального времени. Этому способствует, прежде всего, появление и доступность многоядерных процессоров. В последнее время были разработаны новые методы оптимизации, появился фреймворк параллельного программирования, который открыл новые перспективы для решения численно сложных задач.

Исследована обработка измерительной информации с применением параллельных вычислений на примере спектрального анализа сигналов методом Фурье. Предлагается метод вычисления коэффициентов ряда Фурье, разработанный с учетом использования многоядерных процессоров с целью существенного сокращения времени обработки. Результаты вычислительных экспериментов показывают, что применение параллельного программирования позволяет в разы увеличить производительность вычислений при анализе Фурье.

Ключевые слова: параллельные вычисления, коэффициенты ряда Фурье, многоядерные процессоры, анализ сигналов.

Введение. Исторически сложилось так, что классическими источниками больших данных являются интернет вещей и социальные медиа [1]. Методы обработки больших данных изначально ориентировались на данные, соответствующие этим источникам. Однако в последние годы стали интенсивно развиваться методы обработки больших данных в промышленности и научных исследованиях, когда данными является измерительная информация, поступающая с измерительных устройств [2]. В данной работе рассмотрено применение параллельных вычислений для обработки измерительной информации на примере спектрального анализа сигналов классическим методом Фурье. В

дальнейшем планируется расширить данное исследование путём применения параллельных вычислений для неклассических методов Фурье, таких как БПФ (быстрое преобразование Фурье), оконное преобразование Фурье, полиспектральный анализ [3].

Периодический сигнал может быть представлен в течение определенного интервала времени в виде линейной комбинации ортогональных функций. Если эти ортогональные функции являются тригонометрическими, то представление в виде ряда Фурье будет выглядеть как тригонометрический ряд Фурье. Для вычисления коэффициентов тригонометрического ряда Фурье функции $f(x)$ которая определена на интервале $[0,\ T]$, используется следующая формула [4-6]:

$$f(x) = a_0 + \sum_{n=1}^{\infty}(a_n cos\left(\frac{2\pi nx}{T}\right) + b_n sin\left(\frac{2\pi nx}{T}\right))$$

где a_0 – среднее значение функции (нулевой коэффициент), a_n и b_n – коэффициенты ряда Фурье.

Ряд Фурье всегда является периодическим по своей природе, даже если исходная функция $f(x)$ не является периодической. Периоды $sin\left(\frac{2\pi nx}{T}\right)$ и $cos\left(\frac{2\pi nx}{T}\right)$ равны $\frac{2\pi}{n}$ поэтому наибольший период, $T = 2\pi$, получается из $n = 1$ членов, и ряд Фурье имеет период 2π. Это означает, что ряд должен представлять периодические функции с периодом 2π.

Для расчета коэффициентов мы будем использовать формулы, приведенные ниже [7-13]:

1 Для среднего значения a_0:

$$a_0 = \frac{1}{N}\sum_{k=0}^{N-1} f(x_k)$$

2 Для коэффициентов a_n:

$$a_n = \frac{2}{N}\sum_{k=0}^{N-1} f(x_k)\, cos\left(\frac{2\pi kn}{N}\right), n = 1,2, \ldots$$

3 Для коэффициентов b_n:

$$b_n = \frac{2}{N}\sum_{k=0}^{N-1} f(x_k)\, sin\left(\frac{2\pi kn}{N}\right), n = 1,2, \ldots$$

где x_k представляют собой точки выборки сигнала, N – количество точек выборки, k – индекс точки.

Понимание процессов обработки сигналов и оптимизации производительности. Производители оборудования предоставляют инструменты, библиотеки и среды для разработки, тестирования, проверки и масштабирования параллельных алгоритмов и рабочих нагрузок как для реальных приложений, так и для исследовательских целей. Например, Hewlett Packard Enterprise (HPE) предоставляет среду *Cray Compiler Environment*, совместимую со многими стандартами, включая стандарт *OpenMP*, которая выполняет анализ кода во время компиляции и автоматически генерирует высоко оптимизированный код [14].

Технические характеристики аппаратных вычислительных средств, используемых для проведения расчетов и испытаний, приведены в таблице 1 [15].

Таблица 1. Технические характеристики аппаратных вычислительных средств

Технические характеристики	***Сервер HP ProLiant DL 360 Gen9***
Процессор	Intel Xeon e5-2660 v3 2.6GHZ

Сокет	Intel Socket 2011
Операционная система	Windows Server 2016
Микроархитектура	Sandy Bridge-EP
Память	256 GB
Количество ядер	8
Количество потоков	16
Кэш L1	64 KB (на ядро)
Кэш L2	256 KB (на ядро)
Кэш L3	20 MB (общее)
Процессор	Intel Xeon e5-2660 v3 2.6GHZ

Обработка сигналов оказывает большое влияние на многие области, начиная от связи и аудиоанализа и заканчивая биомедицинской инженерией и радарными системами. Анализ частотных составляющих сигнала, который часто выполняется с помощью преобразования Фурье, является одним из основных методов обработки сигналов. С увеличением размера сигнала эта операция становится все более требовательной к вычислительным ресурсам. Обработка больших сигналов на современном оборудовании без оптимизации проблематична из-за сложности этих вычислений [16].

Для решения этой проблемы было разработано множество алгоритмов, однако, поскольку наборы данных и требования к обработке в режиме реального времени продолжают расти, даже использование этих алгоритмов может оказаться не столь эффективным. Именно здесь на помощь приходит параллелизм. Используя возможности современных многоядерных процессоров, параллелизм позволяет разбивать вычисления на небольшие блоки и распределять нагрузку между многими ядрами, значительно сокращая время выполнения.

В этой части мы рассмотрим реализацию и оптимизацию алгоритмов обработки сигналов на C++ [17], уделяя особое внимание следующему:

1 Генерация тестовых сигналов, имитирующих реальные сигналы.

Функция *generateSignal* возвращает результат дискретной версии непрерывной функции $f(x)$ путем ее дискретизации через регулярные интервалы в определенном диапазоне. Это действие начинается с разбиения диапазона на равноотстоящие друг от друга интервалы numPoints. Каждый из этих интервалов соотносится с размером шага (*step*). Для каждой точки выборки функция вычисляет координату *x* и вычисляет значение $f(x)$, а затем сохраняет результат в векторе с именем *signal*. Выбор данной реализации обусловлен тем, что в отличие от генерации случайных чисел, она предоставляет более адекватные результаты для анализа. Ниже приведен фрагмент кода генерации тестового сигнала:

```
std::vector<double> generateSignal(int numPoints)
{
    std::vector<double> signal(numPoints);
    double step = 2 * PI / numPoints;
    for (int i = 0; i < numPoints; ++i)
    {
        double x = -PI + i * step;
        signal[i] = f(x);
```

```
    }
    return signal;
}
```

2 Реализация непараллельных параллельных вычислений коэффициентов ряда Фурье.

Простое представление вычисления коэффициентов ряда Фурье из сгенерированного сигнала находится в функции *computeFourierCoefficients*. Стоит обратить внимание, что в этой функции не используются методы распараллеливания.

```
void computeFourierCoefficients(const std::vector<double> &signal, int maxN, std::vector<double> &an, std::vector<double> &bn)
{
    int numPoints = signal.size();
    double step = 2 * PI / numPoints;
    // Compute a0
    an[0] = 0.0;
    for (int i = 0; i < numPoints; ++i)
    {
        an[0] += signal[i];
    }
    an[0] *= step / (2 * PI);
    // Compute an and bn
    for (int n = 1; n <= maxN; ++n)
    {
        double an_local = 0.0;
        double bn_local = 0.0;
        for (int i = 0; i < numPoints; ++i)
        {
            double x = -PI + i * step;
            an_local += signal[i] * cos(n * x);
            bn_local += signal[i] * sin(n * x);
        }
        an[n] = (2.0 / numPoints) * an_local;
        bn[n] = (2.0 / numPoints) * bn_local;
    }
}
```

Предлагаемая нами функция *computeFourierCoefficientsParallel* повышает эффективность за счет разделения вычислений между несколькими потоками. Внешний цикл, в котором вычисляются коэффициенты, разделяется между потоками. Количество потоков определяется *OpenMP* автоматически, по умолчанию оно обычно равно количеству ядер. Каждый поток вычисляет определенный диапазон частот независимо друг от друга. Этот метод позволяет оптимально использовать многоядерные процессоры, что ускоряет вычисления, особенно для больших сигналов [18-21]:

```
void computeFourierCoefficientsParallel(const std::vector<double> &signal, int maxN, std::vector<double> &an, std::vector<double> &bn)
{
    int numPoints = signal.size();
    double step = 2 * PI / numPoints;
    // Compute a0. Parallelizing a0 computation may introduce overhead
    // a0 is computed differently from an and bn
    an[0] = 0.0;
    for (int i = 0; i < numPoints; ++i)
    {
        an[0] += signal[i];
    }
    an[0] *= step / (2 * PI);
    // Compute an and bn (parallelized outer loop over n)
```

```
#pragma omp parallel for
    for (int n = 1; n <= maxN; ++n)
    {
        double an_local = 0.0;
        double bn_local = 0.0;
        for (int i = 0; i < numPoints; ++i)
        {
            double x = -PI + i * step;
            an_local += signal[i] * cos(n * x);
            bn_local += signal[i] * sin(n * x);
        }
        an[n] = (2.0 / numPoints) * an_local;
        bn[n] = (2.0 / numPoints) * bn_local;
    }
}
```

3 Оценка производительности для демонстрации преимуществ использования параллелизма.

Измерение производительности выполнения различных методов осуществляется с помощью библиотеки *std::chrono* [22]. Расчет времени выполнения определяется путем вычитания времени начала и окончания функции. Для записи точного времени используются часы с высоким разрешением. Этот подход показывает, как каждый метод справляется с увеличением объема данных, и подчеркивает, как параллелизм сокращает время вычислений, особенно для больших сигналов.

Блок-схема распараллеливания. Блок-схема распараллеливания (рисунок 1) показывает, как задачи разделяются и выполняются параллельно для оптимизации производительности при выполнении функции *computeFourierCoefficientsParallel* где:

1 Вычисление a_0 выполняется последовательно, поскольку оно не зависит от частотных членов и, таким образом, распараллеливание его вычислений может привести к дополнительным расходам вычислительных ресурсов. Этот коэффициент вычисляется путем суммирования значений сигналов.

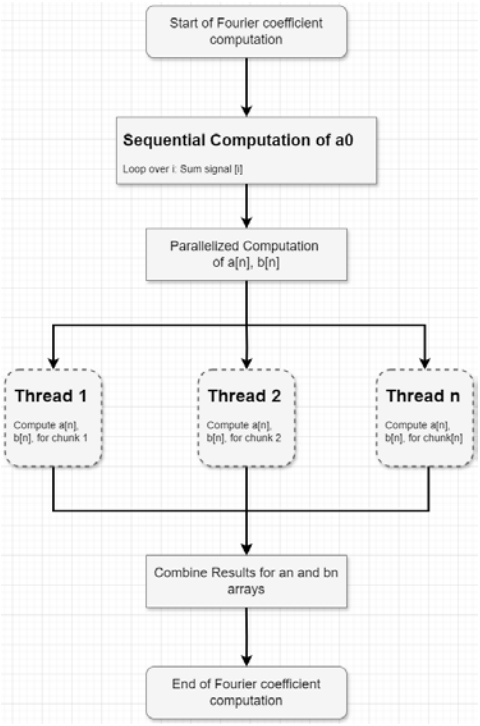

Рисунок 1. Блок-схема распараллеливания

2 В коде функции перед внешним циклом *for* указывается строка с *#pragma omp parallel for*. По умолчанию *OpenMP* выбирает количество потоков на основе доступных ядер. После определения количества ядер каждый поток закрепляется за определенным ядром с определенным диапазоном значений n, который отражает различные гармонические частоты. В процессе вычислений суммируются произведения сигнала с косинусами и синусами для каждого назначенного n. Потоки работают параллельно, поэтому между ними нет пересечений или зависимостей. После того как каждый поток завершает вычисления для назначенного ему диапазона, результаты для a_n и b_n объединяются в массивы.

Компьютерные эксперименты и анализ их результатов. Выполнив вышеизложенный код, мы получили результаты компьютерных экспериментов при непараллельной и параллельной реализациях: в таблице 2 показано соотношение между количеством коэффициентов ряда Фурье и временем вычислений (в миллисекундах) при непараллельной и параллельной реализациях.

Таблица 2. Результаты компьютерных экспериментов

Количество коэффициентов ряда Фурье	***Время при непараллельном вычислении, мс***	***Время при параллельном вычислении, мс***
10	0.8183	0.5169
100	6.9809	2.293
200	13.9411	3.9712
300	21.1175	6.644
400	27.8889	7.2111
500	34.8119	9.2266
1000	71.1401	19.4607

Соотношение между количеством коэффициентов ряда Фурье при непараллельной и параллельной реализациях показано на рисунке 2.

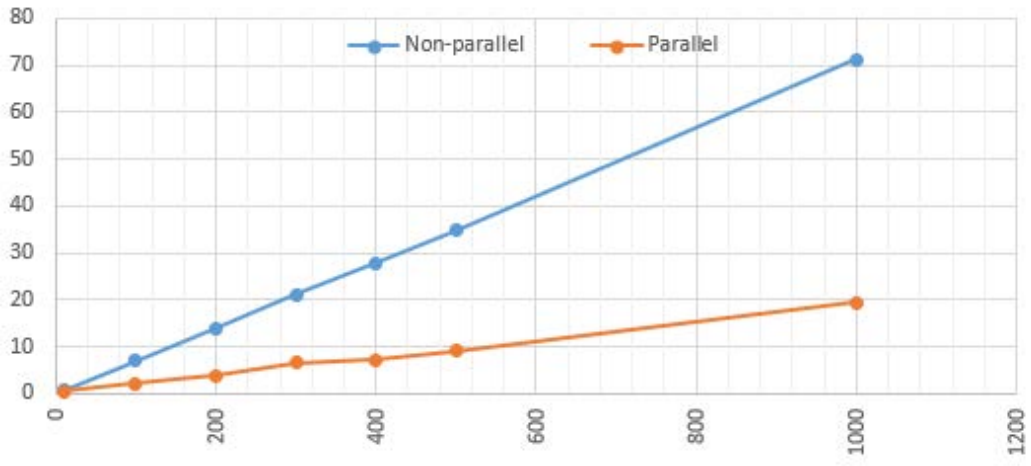

Рисунок 2. Зависимость числа коэффициентов ряда Фурье от времени для непараллельной и параллельной реализаций

На основании полученных результатов мы можем определить следующие ключевые

моменты:

– время выполнения непараллельной реализации значительно увеличивается с ростом числа коэффициентов ряда Фурье. Это говорит об ограниченности однопоточного подхода, при котором все вычисления выполняются последовательно. Например, при небольшом количестве термов (10) время выполнения непараллельной реализации равно 0,8183 миллисекунды. Однако при увеличении числа терминов до 1000 время увеличивается до 71,1401 миллисекунды, что указывает на нелинейный рост вычислительных затрат с увеличением числа членов;

– параллельная реализация демонстрирует меньшее время вычислений по сравнению с непараллельным вариантом. Например, при 1000 членах время параллельного выполнения (19,4 миллисекунды) значительно меньше, чем у непараллельного варианта (71,1 миллисекунды). Таким образом, скорость параллельных вычислений приблизительно в 3,7 раза выше непараллельных.

Заключение. Результаты данного исследования подтверждают эффективность параллельных вычислений коэффициентов ряда Фурье, особенно в сценариях с большим количеством коэффициентов. Использование параллельного фреймворка *OpenMP* позволило устранить узкое место в вычислениях, связанное с последовательной обработкой. Достигнутый прирост производительности показывает пользу распределения вычислительной нагрузки на несколько ядер процессора. Даже для небольшого количества коэффициентов ряда Фурье параллельная версия работает лучше, чем последовательная. Для дальнейшей оптимизации вычислений можно использовать специализированные стандарты, такие как интерфейс передачи сообщений (*MPI*) для распределенных систем. Данная работа показывает, что параллельные вычисления важны не только с точки зрения эффективности, но и позволяют справиться с требованиями, предъявляемыми современной обработкой сигналов. Результаты, полученные в рамках данной работы, могут быть развиты и в дальнейшем применены для оптимизации вычислений в других методах анализа сигналов, таких как БПФ (быстрое преобразование Фурье), оконное преобразование Фурье, полиспектральный анализ [3] и т.д.

Список литературы

[1] Min Chen, Shiwen Mao, Yin Zhang, Victor C.M. Leung. Big Data. Related Technologies, Challenges, and Future Prospects. Spinger, 2014. — 100 p. doi.org/10.1007/978-3-319-06245-7

[2] Черняк Л. Большие Данные — новая теория и практика // Открытые системы. СУБД, №10, 2011. www.osp.ru/os/2011/10/13010990

[3] Новиков А.К. Полиспектральный анализ. СПб.: ЦНИИ им. акад. А. Н. Крылова, 2002. - 178 с.

[4] R. E. Edwards, “Fourier Series: A Modern Introduction Volume 1”, Институт перспективных исследований, Австралийский национальный университет, Canberra, Второе издание, Декабрь. 2012, с. 14-43.

[5] Антоний Зигмунд, “Trigonometric Series”, Cambridge University Press, Февраль. 2015, с. 1-4.

[6] Марк А. Пински, "Introduction to Fourier Analysis and Wavelets", Северо-Западный университет, 2008, с. 13-34.

[7] Данхэм Джексон, "Fourier Series and Orthogonal Polynomials", American Mathematical Soc., 1941, с. 1-42.

[8] Тим Олсон, "Applied Fourier Analysis", Birkhäuser New York, NY, 2017, с. 1-3, 19-30, 45-80.

[9] Дэвид В. Каммлер, "A First Course in Fourier Analysis", Cambridge University Press, 2010, с. 1-40.

[10] Libretexts, "Fourier Trigonometric Series", [Онлайн], Доступно по адресу: https://math.libretexts.org/Bookshelves/Differential_Equations/Introduction_to_Partial_Differential_Equations_(Herman)/03%3A_Trigonometric_Fourier_Series/3.02%3A_Fourier_Trigonometric_Series.

[11] Университет Северной Каролины, "Trigonometric Fourier Series", [Онлайн], Доступно по адресу: https://people.uncw.edu/hermanr/pde1/pdebook/Fourier.pdf.

[12] Википедия, “Fourier series”, [Онлайн], Доступно по адресу: https://en.wikipedia.org/wiki/Fourier_series.

[13] Tutorialspoint, "Trigonometric Fourier Series – Definition and Explanation", [Онлайн], Доступно по адресу: https://www.tutorialspoint.com/trigonometric-fourier-series-definition-and-explanation.

[14] Hewlett Packard, "Cray Compiler Environment", [Онлайн], Доступно по адресу: https://cpe.ext.hpe.com/docs/24.03/cce/index.html#overview.

[15] TechPowerUP, “Intel Xeon E5-2660”, [Онлайн], Доступно по адресу: https://www.techpowerup.com/cpu-specs/xeon-e5-2660.c976.

[16] Кули, Дж. У. и Тьюки, Дж. У. (1965), “An Algorithm for the Machine Calculation of Complex Fourier Series”, Математика вычислений, с. 297–301.

[17] Справочник по C++, "Standard Containers”, [Онлайн], Доступно по адресу: https://cplusplus.com/reference/stl.

[18] CsInParallel, "Parallel Computing in the Computer Science Curriculum", [Онлайн], Доступно по адресу: https://csinparallel.org/csinparallel/ppps/openmp.html.

[19] Microsoft, "Enable OpenMP Support", [Онлайн], Доступно по адресу: https://learn.microsoft.com/en-us/cpp/build/reference/openmp-enable-openmp-2-0-support?view=msvc-170.

[20] OpenMP Документация, "Guide to parallelizing computations in C++ for multi-core processors", [Онлайн], Доступно по адресу: https://www.openmp.org.

[21] OpenMP Примеры, "OpenMP Application Programming Interface Examples", [Онлайн], Доступно по адресу: https://www.openmp.org/wp-content/uploads/openmp-examples-5.2.2-final.pdf.

[22] Справочник по C++, "Date and time utilities" 2024, [Онлайн], Доступно по адресу: https://en.cppreference.com/w/cpp/chrono.

Авторский вклад

Ализаде Тахир Али – постановка задачи, консультации при проведении компьютерных экспериментов и анализе их результатов, формирование структуры статьи.

Мустафаев Акшин Ариф – разработка программного кода для классических и параллельных вычислений коэффициентов ряда Фурье, проведение компьютерных экспериментов, анализ полученных результатов.

PROCESSING LARGE ARRAYS OF MEASUREMENT INFORMATION USING PARALLEL COMPUTING

T.A. Alizada
Head of Laboratory of methods of recognition and intelligent analysis of signals of Institute of Control Systems (Azerbaijan), PhD of Technical sciences, Associate Professor

A.A. Mustafayev
Postgraduate student at the Institute of Control Systems (Azerbaijan)

Annotation. Currently, the use of parallel computing in information systems has become a necessity, especially when it comes to processing large volumes of data in real time. This is primarily driven by the emergence and availability of multicore processors. In recent years, new optimization methods have been developed, and a parallel programming framework has been introduced, opening up new prospects for solving numerically complex tasks.

This study explores the processing of measurement data employing parallel computing, using the example of spectral signal analysis via the Fourier method. A method for computing Fourier series coefficients is proposed, designed with multicore processors in mind, aiming to significantly reduce processing time. The results of computational experiments demonstrate that using parallel programming can greatly increase computation performance during Fourier analysis.

Keywords: parallel computing, Fourier series coefficients, multi-core processors, signal analysis.