MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 29, NO. 1, 2022
journal homepage: www.mais-journal.ru

ALGORITHMS

The Algorithm of Angular Superresolution Using the Cholesky Decomposition and its Implementation Based on Parallel Computing Technology

S. E. Mishchenko[1], N. V. Shatskiy[2]

DOI: 10.18255/1818-1015-2022-1-6-19

[1]Rostov-on-Don Institute of Radiocommunications, 130 Nansena av., Rostov-on-Don 344010, Russia.
[2]Academician A. L. Mints Radiotechnical Institute, 8 March st., 10/1, Moscow 127083, Russia.

MSC2020: 78A50 78M50 68W10
Research article
Full text in Russian

Received February 3, 2022
After revision March 14, 2022
Accepted March 16, 2022

An algorithm of angular superresolution based on the Cholesky decomposition, which is a modification of the Capon algorithm, is proposed. It is shown that the proposed algorithm makes it possible to abandon the inversion of the covariance matrix of input signals. The proposed algorithm is compared with the Capon algorithm by the number of operations. It is established that the proposed algorithm, with a large dimension of the problem, provides some gain both when implemented on a single-threaded and multithreaded computer. Numerical estimates of the performance of the proposed and original algorithm using parallel computing technology CUDA NVidia are obtained. It is established that the proposed algorithm saves GPU computing resources and is able to solve the problem of constructing a spatial spectrum with an increase in the dimension of the covariance matrix of input signals by almost two times.

Keywords: digital array antennas; Capon super-resolution algorithm; Cholesky decomposition,bordering method; parallel computing.

INFORMATION ABOUT THE AUTHORS

Sergey E. Mishchenko, correspondence author — orcid.org/0000-0002-3210-1485. E-mail: mihome@yandex.ru
Leading researcher, Doctor of Engineering Sciences, Professor.

Nikolay V. Shatskiy — orcid.org/0000-0002-6365-7734. E-mail: shteiz@mail.ru
Head of the Integrated Department, PhD in Engineering Sciences, Associate Professor.

For citation: S. E. Mishchenko and N. V. Shatskiy, "The Algorithm of Angular Superresolution Using the Cholesky Decomposition and its Implementation Based on Parallel Computing Technology", *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 6-19, 2022.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 29, № 1, 2022
сайт журнала: www.mais-journal.ru

ALGORITHMS

Алгоритм углового сверхразрешения с использованием разложения Холецкого и его реализация на основе технологии параллельных вычислений

С. Е. Мищенко[1], Н. В. Шацкий[2]

DOI: 10.18255/1818-1015-2022-1-6-19

[1]ФГУП "Ростовский научно-исследовательский институт радиосвязи", ул. Нансена, д. 130, г. Ростов-на-Дону, 344010 Россия.
[2]Радиотехнический институт имени академика А. Л. Минца, ул. 8 Марта, д. 10, стр. 1, г. Москва, 127083 Россия.

УДК 621.396.677
Научная статья
Полный текст на русском языке

Получена 3 февраля 2022 г.
После доработки 14 марта 2022 г.
Принята к публикации 16 марта 2022 г.

Предложен алгоритм углового сверхразрешения на основе разложения Холецкого, представляющий собой модификацию алгоритма Кейпона. Показано, что предложенный алгоритм позволяет отказаться от обращения ковариационной матрицы входных сигналов. Проведено сравнение предложенного алгоритма с алгоритмом Кейпона по числу операций. Установлено, что предложенный алгоритм при большой размерности задачи обеспечивает некоторый выигрыш как при реализации на однопоточном, так и на многопоточном вычислителе. Получены численные оценки быстродействия предложенного и исходного алгоритма с использованием технологии параллельных вычислений CUDA NVIDIA. Установлено, что предложенный алгоритм обеспечивает экономию вычислительных ресурсов GPU и способен решать задачу построения пространственного спектра при увеличении размерности ковариационной матрицы входных сигналов почти в два раза.

Ключевые слова: цифровые антенные решетки; алгоритм сверхразрешения Кейпона; разложение Холецкого; метод окаймления; параллельные вычисления

ИНФОРМАЦИЯ ОБ АВТОРАХ

Сергей Евгеньевич Мищенко автор для корреспонденции	orcid.org/0000-0002-3210-1485. E-mail: mihome@yandex.ru ведущий научный сотрудник, доктор технических наук, профессор.
Николай Витальевич Шацкий	orcid.org/0000-0002-6365-7734. E-mail: shteiz@mail.ru начальник комплексного отдела, кандидат технических наук, доцент.

Для цитирования: S. E. Mishchenko and N. V. Shatskiy, "The Algorithm of Angular Superresolution Using the Cholesky Decomposition and its Implementation Based on Parallel Computing Technology", *Modeling and analysis of information systems*, vol. 29, no. 1, pp. 6-19, 2022.

Введение

Алгоритмы пространственно-временной адаптивной обработки сигналов находят применение в радарах с цифровыми антенными решетками (ЦАР) для обнаружения источников радиоизлучения и определения их параметров: угловых координат — в пространственной области, радиальной скорости — в частотной области [1—4]. При построении пространственного спектра размерность задачи оптимальной обработки связана с числом приемных каналов ЦАР, в частотной области размерность задачи обусловлена числом зондирующих импульсов. В многоэлементных ЦАР основная сложность реализации алгоритмов адаптивной обработки сигналов связана с необходимостью формирования ковариационной матрицы входных сигналов (КМВС) большой размерности и обращения данной матрицы [5]. Эти операции желательно осуществлять в реальном масштабе времени.

Необходимость обработки больших объемов информации заставляет совершенствовать существующие алгоритмы пространственно-временной обработки и использовать технологии параллельных вычислений. В настоящее время в области параллельных вычислений широкое применение находит технология CUDA (Compute Unified Device Architecture) – программно-аппаратная архитектура параллельных вычислений. Применение параллельных вычислений позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы NVIDIA Corporation. Обычно выигрыш в производительности от использования технологий параллельной обработки оценивают применительно к простейшим алгоритмам векторных вычислений. Например, в работе [6] было показано, что использование технологии CUDA позволяет сократить время формирования КМВС почти в 30 раз для ЦАР, содержащей $M = 1024$ приемных каналов. Если же устройство обработки поддерживает технологию CUDA Direct Access [7], то формирование КМВС теоретически осуществимо в реальном масштабе времени, поскольку потребует для каждого нового временного среза выполнить последовательно M^2 комплексных умножений новых данных и M^2 комплексных сложений результатов умножения с элементами ранее сформированной матрицы. Последовательности операций умножения и сложения являются независимыми и могут выполняться в параллельных потоках вычислений.

В связи с этим дальнейшее повышение быстродействия алгоритмов пространственно-временной адаптивной обработки неразрывно связано с их реализацией на основе технологии параллельных вычислений.

Цель работы состоит в разработке алгоритма адаптивной обработки сигналов не требующего обращения КМВС, и оценке эффективности реализации данного алгоритма с использованием технологии параллельных вычислений.

1. Обоснование алгоритма и аналитическая оценка его сложности при использовании технологии параллельных вычислений

Пусть в процессе функционирования M-элементной ЦАР сформирована КМВС $\mathbf{R}$. В соответствии с алгоритмом Кейпона для формирования пространственного спектра заранее формируют набор векторов гипотез $\mathbf{V}(\theta_i, \varphi_i)$, где $i = 1, 2, ..., N_i$, после формирования КМВС выполняют ее обращение одним из известных алгоритмов и строят пространственный спектр по формуле

$$I(\theta_i, \varphi_i) = \frac{1}{\mathbf{V}(\theta_i, \varphi_i) \cdot \mathbf{R}^{-1} \cdot \mathbf{V}^H(\theta_i, \varphi_i)}, \tag{1}$$

Здесь символ H обозначает операцию сопряжения Эрмита, а вектор гипотеза $\mathbf{V}(\theta_i, \varphi_i)$ обеспечивает фазирование ЦАР в одном из N_i направлений (θ_i, φ_i).

Отсюда следует, что процесс обработки сигналов по формуле (1) состоит в выполнении двух этапов:

- обращение корреляционной матрицы $\mathbf{R}$;

- циклический перебор N_i векторов-гипотез, для каждой из которых сначала выполняют умножение вектора на матрицу, а затем, находят скалярное произведение результирующего вектора и вектора-гипотезы.

В известной литературе рассматриваются различные методы решения систем линейных уравнений и обращения матриц. В монографии [8] кроме алгоритмов линейной алгебры приводятся оценки числа операций, необходимые для их реализации. Одним из наиболее эффективных методов обращения матриц, позволяющим учесть эрмитовы свойства матрицы $\mathbf{R}$, является метод окаймления [8—10]. Вычислительной схеме метода окаймления соответствуют выражения:

$$\left(\mathbf{R}_1^{-1}\right)_{1,1} = \frac{R_{1,1}}{R_{1,1}R_{2,2} - \left|R_{1,2}^2\right|};\ \left(\mathbf{R}_1^{-1}\right)_{1,2} = -\frac{R_{2,1}}{R_{1,1}R_{2,2} - \left|R_{1,2}^2\right|}; \tag{2}$$

$$\left(\mathbf{R}_1^{-1}\right)_{2,1} = -\frac{R_{1,2}}{R_{1,1}R_{2,2} - \left|R_{1,2}^2\right|} = \left(\mathbf{R}_1^{-1}\right)^*_{1,2};\ \left(\mathbf{R}_1^{-1}\right)_{2,2} = \frac{R_{2,2}}{R_{1,1}R_{2,2} - \left|R_{1,2}^2\right|}; \tag{3}$$

$$\mathbf{R}_m = \begin{pmatrix} \mathbf{R}_{m-1} & \mathbf{u}_m \\ \mathbf{u}_m^H & R_{m,m} \end{pmatrix}; \tag{4}$$

$$\mathbf{R}_m^{-1} = \begin{pmatrix} \mathbf{R}_{m-1}^{-1} + \frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)^H & -\frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right) \\ -\frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)^H & \frac{1}{\rho_{m,m}} \end{pmatrix}; \tag{5}$$

$$\rho_{m,m} = R_{m,m} - \mathbf{u}_m^H\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right). \tag{6}$$

В выражении (3) символ $*$ обозначает операцию комплексного сопряжения.

На первом этапе при реализации метода окаймления выполняют обращение матрицы размером 2×2 с использованием выражений (2) и (3).

При реализации на многопоточном вычислителе метода окаймления число последовательных однотипных операций в этом случае составит: $\lceil 2/P \rceil + \lceil 3/P \rceil$ умножений; 1 сложение и 1 деление. Здесь и далее $\lceil x \rceil$ обозначает операцию округления x до ближайшего старшего целого, а P – число параллельных потоков вычислений.

После этого необходимо постепенно увеличивать ранг обратной матрицы до M, т. е. выполнить еще $M - 1$ циклов, в ходе каждого из которых выполняют операции в соответствии с (4)-(6). Формулы (4)-(6) включают операции: умножение матрицы на вектор, скалярное произведение.

Умножение матрицы размером $m \times m$ на вектор-столбец потребует вычислить: $\lceil m^2/P \rceil$ умножений и $\lceil m(m-1)/P \rceil$ сложений, а скалярное произведение – $\lceil m/P \rceil$ умножений и $\lceil (m-1)/P \rceil$ сложений (вычитаний).

Каждый m-ый из $M - 1$ циклов алгоритма обращения матрицы методом окаймления требует выполнить последовательность действий, приведенных в таблице 1. С учетом данных таблицы 1

Table 1. The computing difficulty by (4)-(6) for preassigned m

Описание действия	Число операций ($m = 3, ..., M$)
произведение матрицы на вектор $\mathbf{R}_{m-1}^{-1}\mathbf{u}_m$	умножений $\lceil (m-1)^2/P \rceil$; сложений $\lceil (m-1)(m-2)/P \rceil$
скалярное произведение векторов $\mathbf{u}_m^H\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)$	умножений $\lceil (m-1)/P \rceil$; сложений $\lceil (m-2)/P \rceil$
диагональный элемент $1/\rho_{m,m}$	1 сложение; 1 деление
элементы матрицы $\frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)^H$	умножений $2\lceil (m-1)^2/P \rceil$
умножение $\frac{1}{\rho_{m,m}}\left(\mathbf{R}_{m-1}^{-1}\mathbf{u}_m\right)$	умножений $\lceil (m-1)/P \rceil$

Таблица 1. Сложность вычислений согласно (4)-(6) при заданном m

получим следующие оценки сложности алгоритма обращения матрицы методом окаймления на многопоточном вычислителе. Число последовательно выполняемых операций умножения $O_{\times}^{BM}(M,P)$, сложения $O_{+}^{BM}(M,P)$ и деления $O_{/}^{BM}(M,P)$ равно:

$$O_{\times}^{BM}(M,P) = \lceil 2/P \rceil + \lceil 3/P \rceil \sum_{m=1}^{M} \left(3\lceil (m-1)^2/P \rceil + 2\lceil (m-1)/P \rceil \right); \tag{7}$$

$$O_{+}^{BM}(M,P) = 1 + \sum_{m=1}^{M} \left(\lceil (m-1)(m-1)/P \rceil + \lceil (m-2)/P \rceil + 1 + \lceil (m-1)^2/P \rceil \right); \tag{8}$$

$$O_{/}^{BM}(M) = M - 2. \tag{9}$$

В монографии [8] приведены следующие оценки вычислительных затрат при реализации метода окаймления для обращения матрицы:

- $0,5\left(M^3 + M^2 - 2M\right)$ комплексных умножений;
- $0,5\left(M^3 - M^2\right)$ комплексных сложений;
- M комплексных делений.

Сопоставление выражений (7) и (8) с соответствующими оценками из [8] позволяет заключить, что выигрыш от распараллеливания может быть очень существенным по операциям комплексного умножения и сложения.

Дальнейшая реализация алгоритма Кейпона состоит в том, что многократно выполняют умножение вектора на обратную матрицу. Это соответствует:

- $N_i M^2$ комплексных умножений;
- $N_i M\,(M-1)$ комплексных сложений.

После этого рассчитывают скалярное произведение двух векторов, первый из которых получен в результате умножения обратной матрицы на вектор-гипотезу и на исходный вектор гипотезу. На данном этапе имеем:

- $N_i M$ комплексных умножений;
- $N_i\,(M-1)$ комплексных сложений.

При переходе к многопоточному вычислителю суммарные затраты на реализацию алгоритма Кейпона составляют:

$$O_{\times}^{Capon}(M,P,N_i) = O_{\times}^{BM}(M,P) + N_i\lceil M/P \rceil^2 P + N_i\lceil M/P \rceil; \tag{10}$$

$$O_{+}^{Capon}(M,P,N_i) = O_{+}^{BM}(M,P) + N_i\lceil M(M-1)/P \rceil + N_i\lceil (M-1)/P \rceil; \tag{11}$$

$$O_{/}^{Capon}(M) = O_{/}^{BM}(M) = M - 2. \tag{12}$$

Можно также получить суммарную оценку сложности, если учесть, что операции действительной арифметики при комплексном умножении распараллеливаются и сводятся к одному умножению и сложению. Операцию деления будем рассматривать как совокупность трех операций: двух умножений и сложения. В результате получим:

$$O_{\Sigma}^{Capon}(M,P,N_i) = 2O_{\times}^{Capon}(M,P,N_i) + O_{+}^{Capon}(M,P,N_i) + 3O_{/}^{Capon}(M). \tag{13}$$

В качестве альтернативного варианта реализации алгоритма Кейпона предложим алгоритм, в котором пространственный спектр представляется выражением

$$I(\theta_i,\varphi_i) = \frac{1}{\mathbf{V}(\theta_i,\varphi_i)\cdot\mathbf{X}(\theta_i,\varphi_i)}, \tag{14}$$

где вектор-столбец $\mathbf{X}(\theta_i, \varphi_i)$ для каждой из гипотез соответствует решению системы уравнений

$$\mathbf{R} \cdot \mathbf{X}(\theta_i, \varphi_i) = \mathbf{V}^H(\theta_i, \varphi_i). \tag{15}$$

Существуют различные методы решения систем уравнений без обращения матрицы коэффициентов при неизвестных. Применим для решения системы уравнений (15) разложение Холецкого (метод квадратного корня в [8]), в соответствии с которым

$$\mathbf{R} = \mathbf{L} \cdot \mathbf{L}^H, \tag{16}$$

где $\mathbf{L}$ – нижнетреугольная матрица, элементы которой определяют по формулам [8]:

$$L_{1,1} = \sqrt{R_{1,1}};\ L_{m,1} = \frac{R_{m,1}}{L_{1,1}}; \tag{17}$$

$$L_{m,m} = \sqrt{R_{m,m} - \sum_{p=1}^{m-1} L^*_{m,p} L_{m,p}};\ m = 2, ..., M; \tag{18}$$

$$L_{m,n} = \frac{1}{L_{n,n}} \left(R_{m,n} - \sum_{p=1}^{n-1} L^*_{n,p} L_{m,p} \right);\ n = 2, ..., M-1;\ m = n+1, ..., M. \tag{19}$$

Для определения искомого вектора $\mathbf{X}(\theta_i, \varphi_i)$ требуется решить две системы уравнений методом исключения [11]:

$$\mathbf{L}^H \cdot \mathbf{X}(\theta_i, \varphi_i) = \mathbf{V}'(\theta_i, \varphi_i);\ \mathbf{L} \cdot \mathbf{V}'(\theta_i, \varphi_i) = \mathbf{V}^H(\theta_i, \varphi_i). \tag{20}$$

Отсюда следует, что алгоритм сверхразрешения, основанный на разложении Холецкого также состоит из двух этапов, первый из которых состоит в формировании нижнетреугольной матрицы $\mathbf{L}$ по формулам (17)–(19), а второй состоит в том, что для различных наборов векторов-гипотез решают системы уравнений (20), а затем вычисляют скалярное произведение текущего вектора-гипотезы и вектора, соответствующего решению систем линейных уравнений (15).

При реализации данного алгоритма система линейных уравнений решается с использованием меньшего числа операций, однако отсутствие обратной матрицы приводит к тому, что разложение Холецкого необходимо выполнить один раз, а решение систем уравнений (20), в которых матрица коэффициентов при неизвестных является треугольной – многократно.

В соответствии с оценками [8] для решения системы линейных уравнений методом квадратного корня необходимо выполнить:

- $\frac{1}{6}\left(M^3 + 9M^2 + 2M\right)$ комплексных умножений;
- $\frac{1}{6}\left(M^3 + 6M^2 - 7M\right)$ комплексных сложений;
- M делений;
- M операций извлечения квадратного корня.

В настоящее время операцию извлечения квадратного корня выполняют с использованием итерационного метода Ньютона [12—14]. При этом существуют эффективные в вычислительном отношении процедуры, которые позволяют ограничить число итераций за счет очень близкой начальной оценки [13]. Примем, что для отыскания квадратного корня методом Ньютона достаточно 4 итерации. Поскольку при вычислении квадратного корня оперируют только с действительными числами, то можно считать, что одна итерация соизмерима с одним произведением двух комплексных чисел, а наличие операции извлечения корня по вычислительным затратам требует только $4M$ операций комплексного умножения, которые не могут быть распараллелены.

Число операций, необходимое для решения двух систем уравнений с треугольными матрицами коэффициентов несложно оценить на простых численных примерах, из которых следует, что они

определяются выражениями для суммы членов арифметической прогрессии от 1 до M с шагом 1. Для двух систем уравнений число умножений и сложений будет одинаковым и равно $M(M-1)$. Эти операции следует исключить из приведенных выше оценок, для того, чтобы оценить сложность выполнения разложения Холецкого.

Рассмотрим теперь схему реализации разложения Холецкого на многопоточном вычислителе. Реализацию алгоритма начинают с заполнения первого диагонального элемента и внедиагональных элементов в первом столбце.

Для этого шага необходимо вычислить квадратный корень, выполнить одну операцию деления и $\lceil (M-1)/P \rceil$ операций умножения.

Далее последующие этапы состоят в заполнении элементов столбцов, в которых число внедиагональных элементов уменьшается от $M-2$ до 0. Описания соответствующих шагов и сложность их оценки приведены в таблице 2.

Table 2. The realization difficulty of the Cholesky decomposition for one iteration of m on a multithreaded computer

Таблица 2. Сложность реализации разложения Холецкого для одной итерации m на многопоточном вычислителе

Описание действия	Число операций
вычисление диагонального элемента m-го столбца $(1 < m \le M)$	умножений $\lceil (m-1)2/P \rceil$; сложений $2\lceil (m-1)/P \rceil$; 1 квадратный корень
вычисление внедиагональных элементов	умножений $2\lceil (m-1)/P \rceil$; сложений $2\lceil (m-2)/P \rceil$; 1 деление

На основании таблицы 2 и операций для заполнения первого этапа получим число последовательно выполняемых операций умножения:

$$O_{\times}^{Cholesky}(M,P) = \lceil (M-1)/P \rceil + 3\sum_{m=2}^{M-1} \lceil (m-1)/P \rceil; \tag{21}$$

сложения:

$$O_{+}^{Cholesky}(M,P) = 2\sum_{m=2}^{M-1} \left(\lceil (m-1)/P \rceil + \lceil (m-2)/P \rceil \right); \tag{22}$$

$O_{/}^{Cholesky}(M) = M$ делений; $O_{\sqrt{}}^{Cholesky}(M) = M$ операций извлечения квадратного корня. Полученные соотношения демонстрируют не только существенное сокращение числа операций при использовании параллельных вычислений при реализации разложения Холецкого по сравнению с методом окаймления. Как видно, элементы треугольной матрицы, определяемой при реализации алгоритма могут записываться на месте элементов исходной матрицы. Это позволяет сохранить ресурсы памяти вычислителя для работы. Этот вывод согласуется и с результатами анализа алгоритма в [8].

Теперь оценим вычислительные затраты, необходимые при многократном вычислении значений пространственного спектра в различных направлениях для предлагаемого алгоритма.

Алгоритм сверхразрешения, основанный на разложении Холецкого, требует решения систем уравнений (20) и вычисления скалярного произведения векторов.

Каждая из систем уравнений (20) решается методом исключения. Этот метод несложно реализовать на однопоточном вычислителе. Оценка его сложности в случае многопоточного вычислителя может быть выполнена следующим образом.

Каждая из треугольных матриц при решении систем уравнений (20) содержит $0,5\left(\lceil M/P \rceil^2 - \lceil M/P \rceil\right)$ полностью заполненных блоков коэффициентов и $\lceil M/P \rceil$ блоков, которые

представляют собой треугольные матрицы. При этом алгоритм решения системы уравнений методом исключения при распараллеливании можно разбить на следующие этапы:

- решение системы P уравнений с треугольной матрицей коэффициентов методом исключения;
- переход к следующим P уравнениям, приведение их к виду системы уравнений с треугольной матрицей коэффициентов.

Выполнение первого этапа включает P операций деления. Всего для всей исходной системы уравнений потребуется M операций деления. Эти операции не могут выполняться параллельно.

Алгоритм решения системы P уравнений с треугольными коэффициентами начинается с вычисления первого компонента вектора решения. Для его вычисления необходима одна операция деления. Далее необходимо откорректировать последующие $P-1$ уравнений, исключив из них первый столбец коэффициентов. При выполнении этих операций одновременно для всех уравнений потребуется одна операция умножения и одна операция сложения. Продолжая эти действия можно заключить, что для решения рассматриваемой системы P уравнений достаточно выполнить $P-1$ операций комплексного умножения и столько же операций сложения (вычитания). Поскольку общее число блоков подобных систем уравнений равно $\lceil M/P \rceil$, то число умножений и сложений будет равно $\lceil M/P \rceil (P-1)$ операций соответственно.

Приведение следующих P строк исходной системы уравнений к системе уравнений с треугольной матрицей коэффициентов требует вычисления P^2 умножений и P^2 вычитаний (сложений). При выполнении параллельных вычислений эти оценки следует уменьшить в P раз, т.е. число умножений и сложений станет равным P.

Для произвольного по счету блока уравнений из P строк число удаляемых коэффициентов для приведения матрицы при неизвестных к треугольному виду будет изменяться дискретно, а общее число таких коррекций будет соответствовать числу блоков заполненных коэффициентов. Отсюда следует, что выполнение данного этапа для всей системы уравнений потребует $0,5\left(\lceil M/P \rceil^2 - \lceil M/P \rceil\right) P$ последовательностей операций комплексного умножения и столько же операций сложения.

Отсюда следует, что число операций с комплексными числами для реализации предлагаемого алгоритма на многопоточном вычислителе равно:

$$O_{\times}^{Sgstd}(M,P,N_i) = O_{\times}^{Cholesky}(M,P) + N_i\left(\lceil M/P \rceil^2 - \lceil M/P \rceil\right) P + N_i \lceil M/P \rceil (P-1); \tag{23}$$

$$O_{+}^{Sgstd}(M,P,N_i) = O_{+}^{Cholesky}(M,P) + N_i\left(\lceil M/P \rceil^2 - \lceil M/P \rceil\right) P + N_i \lceil M/P \rceil (P-1); \tag{24}$$

$$O_{/}^{Sgstd}(M,N_i) = O_{/}^{Cholesky}(M) + N_i M;\ O_{\sqrt{}}^{Sgstd}(M) = O_{\sqrt{}}^{Cholesky}(M) = M. \tag{25}$$

Аналогично выражению (13) запишем суммарную оценку числа операций при реализации предложенного алгоритма в виде

$$O_{\Sigma}^{Sgstd}(M,P,N_i) = 2O_{\times}^{Sgstd}(M,P,N_i) + O_{+}^{Sgstd}(M,P,N_i) + 3O_{/}^{Sgstd}(M,P,N_i) + 4M. \tag{26}$$

Таким образом, предложенный алгоритм построения пространственного спектра отличается от алгоритма Кейпона тем, что не требует операции обращения КМВС. Это приводит к тому, что при построении пространственного спектра в каждом направлении приходится решать две системы линейных уравнений с треугольными коэффициентами. Такое усложнение алгоритма может приводить к существенному замедлению решения задачи на однопоточном вычислителе. Однако полученные аналитические выражения показывают, что использование технологии параллельных вычислений может существенно изменить вычислительную эффективность предложенного алгоритма сверхразрешения.

В связи с этим ниже были получены оценки сложности классического алгоритма Кейпона и предлагаемого алгоритма углового сверхразрешения при реализации на многопоточном вычислителе.

2. Результаты численных исследований

На рисунке 1 представлены результаты сопоставления сложности алгоритма Кейпона, при реализации которого используется обращение матрицы, и предложенного алгоритма, основанного на использовании разложения Холецкого. Данные результаты получены при $N_i = 1801$. Две верхние кривые соответствуют зависимостям сложности алгоритмов при их реализации на однопоточном вычислителе ($P = 1$). Данные зависимости демонстрируют сопоставимость сложности обоих алгоритмов. Нижние кривые получены при $P = 32$. Из их сравнения следует, что при увеличении числа потоков с ростом размерности задачи возрастает эффективность предложенного алгоритма сверхразрешения. При $M = 4000$ сложность предложенного алгоритма, реализованного на многопоточном вычислителе при $P = 32$, по сравнению с оценкой для $P = 1$ уменьшилась в 43 раза. Число операций при реализации классического алгоритма Кейпона с увеличением числа потоков до 32 сократилось в 22 раза.

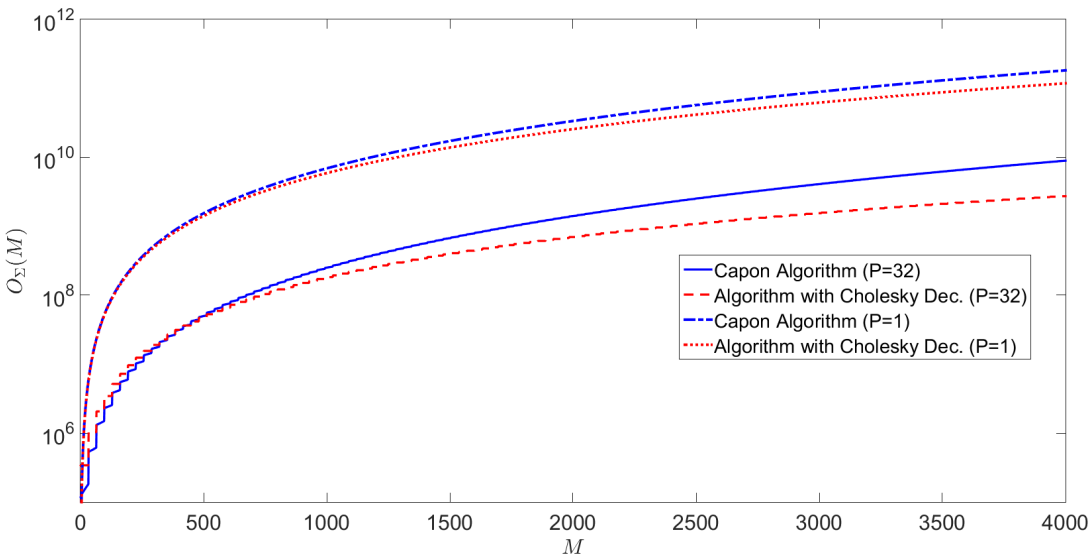

Fig. 1. Theoretical comparing of difficulty of suggested algorithm and Capon's algorithm

Рис. 1. Теоретическое сравнение сложности предложенного алгоритма и алгоритма Кейпона

Для подтверждения полученных аналитически результатов была написана программа на языке Python, содержащая процедуры: разложения Холецкого, обращения эрмитовой матрицы по методу окаймления, умножения комплексного вектора на эрмитову матрицу и скалярного произведения векторов. При реализации алгоритмов сверхразрешения при вычислениях на CPU использовались функции линейной алгебры модуля Numpy Python.

Ниже представлен фрагмент кода программы, реализующей рассматриваемые алгоритмы сверхразрешения при условии, что КМВС уже сформирована и загружена в память CPU. В тексте фрагмента программы не приведен код для формирования КМВС и векторов-гипотез.

```
import numpy as np                  # Инициализация функций модуля Numpy Python
tria(v1,L):                         # Процедура решения систем уравнений (20)
    M = v1.size                     # v1 – Вектор-гипотеза, L – Нижнетреугольная матрица
    v1[0] = v1[0]/L[0,0]
    for i in range(1,M):
        tmp = 0
```

```
        for j in range(i):
            tmp += L[i,j]*v1[j]
        v1[i] = (v1[i]-tmp)/L[i,i]          # Результат решения первой системы уравнений из (20)
    v1[M-1] = v1[M-1]/L[M-1,M-1]
    for i in range(M-2,-1,-1):
        tmp = 0
        for j in range(i+1,M):
            tmp += L[j,i].conjugate()*v1[j]
        v1[i] = (v1[i]-tmp)/L[i,i]
    return v1                               # Возвращает вектор решения обеих систем уравнений
V1 = np.dot(np.linalg.inv(R),V)             # Умножение обратной матрицы на вектор-гипотезу V
ICPU=1/abs(np.dot(V.transpose().conjugate(),V1))        # Результат расчета в соответствии с (1)
L = np.linalg.cholesky(R)                               # Разложение Холецкого КМВС R
V1 = tria(V,L)                                          # Решение систем уравнений (20)
ICPU-Chol = 1/abs(np.dot(V1.transpose().conjugate(),V)) # Результат расчета в соответствии с (14)
```

Параллельные вычисления осуществлялись с использованием функций CUDA модуля Numba Python. Далее представлен фрагмент текста программы, в котором оформлены процедуры, реализующие основные этапы рассматриваемых алгоритмов сверхразрешения с использованием технологии CUDA. Распределение ресурсов вычислителя для реализации параллельных вычислений обеспечивается на этапе компиляции процедур с декоратором @cuda.jit.

```
from numba import cuda                          # Инициализация функций модуля Numba Python
@cuda.jit('void(complex128[:,:],complex128[:,:])')  # Декорирование функции
def cuda_cholesky(R, L):                        # Процедура разложения Холецкого матрицы R
    L[0,0] = cmath.sqrt(abs(R[0,0]))            # Первый диагональный элемент
    M = L[0,:].size
    abs(cmath.sqrt(L.size))
    for i in range(1,M):
        L[i,0] = R[i,0]/abs(L[0,0])             # Заполнение первого столбца
    for i in range(1,M):
        tmp = 0
        for j2 in range(i):
            tmp += L[i,j2]*L[i,j2].conjugate()
            L[i,i] = cmath.sqrt(R[i,i]-tmp)     # Заполнение диагонального элемента
        for j2 in range(i+1,M):
            tmp = 0
            for j1 in range(i):
            tmp += L[j2,j1]*L[i,j1].conjugate()
        L[j2,i] = (R[j2,i] - tmp)/L[i,i]        # Заполнение внедиагональных элементов

@cuda.jit('void(complex128[:],complex128[:],complex128[:])')
def cuda_vect_mul(v1,v2,C):                     # Процедура скалярного произведения векторов
    M = v1.size
    for i in range(M):
        C[0] += v1[i]*v2[i].conjugate()

@cuda.jit('void(complex128[:],complex128[:,:],complex128[:])')
```

```
def mul_VxM(V,L,V1):                          # Процедура умножения матрицы на вектор
    M = V1.size
    for i in range(M):
        V1[i] = 0
        for j in range(M):
            V1[i] += V[j]*L[j,i]

@cuda.jit('void(complex128[:,:],complex128[:,:],complex128[:],complex128[:],complex128[:])')
def okaimlen(R,L,V1,V2):              # Процедура обращения матрицы методом окаймления
    tmp = R[0,0]*R[1,1]-abs(R[0,1])**2
    L[0,0] = R[1,1]/tmp; L[0,1] = -R[1,0].conjugate()/tmp
    L[1,0] = L[0,1].conjugate(); L[1,1] = R[0,0]/tmp; M = V2.size
    for i in range(2,M):
        for j in range(i):
            V2[j] = R[i,j]
        for i1 in range(i):
            V1[i] = 0
            for j in range(i):
                V1[i1] += V2[j]*L[j,i1]
        tmp = 0
        for i1 in range(i):
            tmp += V1[i1]*V2[i1].conjugate()
        L[i,i] = 1/(R[i,i] - tmp)
        for j in range(i):
            L[j,i] = -(V1[j]*L[i,i]).conjugate(); L[i,j] = L[j,i].conjugate()
            for k in range(i):
                L[j,k] = L[j,k] + L[i,i]*V1[j].conjugate()*V1[k]
```

Помимо приведенных процедур также использовалась процедура cuda_tria(v1,L), которая отличалась от процедуры tria(v1,L) только использованием декоратора:
@cuda.jit('void(complex128[:],complex128[:,:])').

Наконец, ниже приведен фрагмент текста программы, в которой рассматривалась реализация алгоритма Кейпона с использованием технологии CUDA и реализованных процедур.

```
device = cuda.get_current_device()          # Инициализация видеоадаптера для вычислений
tpb = [16,16]; Nmax = 108
for N in range(3,Nmax):                     # Индекс N определяет размерность КМВС R
    bpg = int(np.ceil(N/16))
    d_R = cuda.to_device(R)                 # Загрузка данных в память видеоадаптера
    d_V1 = cuda.to_device(V1); d_V2 = cuda.to_device(V2)
    d_V0 = cuda.to_device(V0); d_C = cuda.to_device(C)
    d_L = cuda.to_device(L)
    okaimlen[bpg, tpb](d_R,d_L,d_V1,d_V2)   # Обращение КМВС методом окаймления
    mul_VxM[bpg, tpb](d_V0,d_L,d_V1)        # Умножение обратной матрицы на вектор-гипотезу
    cuda_vect_mul[bpg, tpb](d_V0,d_V1,C)    # Скалярное произведение векторов
    C = d_C.copy_to_host()                  # Извлечение результата из памяти видеоадаптера
```

При реализации предложенного алгоритма в тексте программы вместо строк, в которых выполнялось обращение матрицы и умножение обратной матрицы на вектор исполльзовались команды

```
cuda_cholesky[bpg, tpb](d_R,d_L) # Разложение Холецкого
cuda_tria[bpg, tpb](d_V1,d_L)    # Решение систем уравнений (20)
```

Приведенные фрагменты программ не содержат функций, связанных с оценкой временных затрат на выполнение тех или иных этапов алгоритмов и выводом результатов этих оценок. Кроме того, в текстах программ отсутствует код, обеспечивающий формирование КМВС **R** и векторов гипотез **V**. При проведении исследований учитывалось, что используемый видеоадаптер имеет ограниченный объем памяти 4 Гб, что не позволяло загружать весь массив векторов-гипотез в память GPU. В связи с этим временные оценки соответствовали одному вектору-гипотезе, а затем умножались на заданное значение N_i. При этом формула для оценки быстродействия имела вид

$$T(M) = T_1(M) + N_i \cdot T_2(M), \tag{27}$$

где $T_1(M)$ – время на обращение матрицы **R** для исходного алгоритма или время выполнения разложения Холецкого этой же матрицы для предложенного алгоритма; $T_2(M)$ – время на проверку одного вектора-гипотезы.

При проведении численных исследований использовалась ЭВМ со следующими характеристиками:

- процессор – Intel® Core™ i7-2600K CPU @ 3.40 GHz 3.40 GHz;
- объем ОЗУ– 8 ГБ;
- видеоадаптер – NVIDIA GeForce GTX 550 Ti 4ГБ ОЗУ.

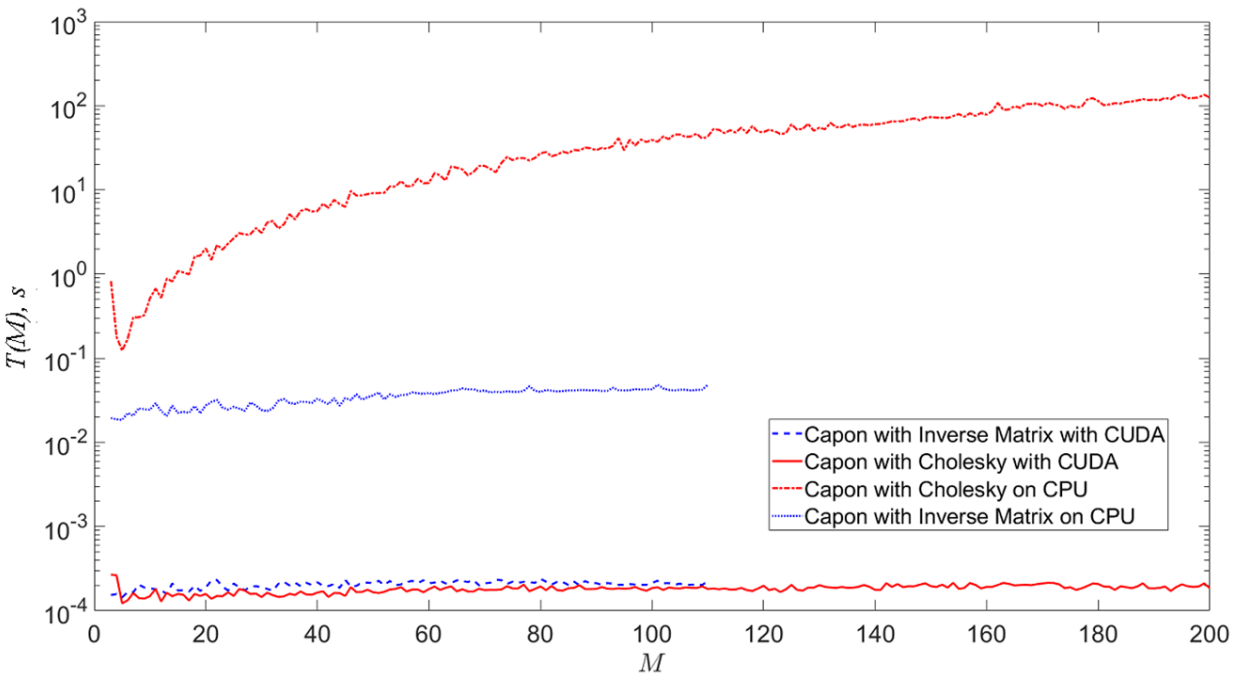

Fig. 2. The performance of the proposed algorithm and the Capon algorithm when implemented on CPU and GPU

Рис. 2. Быстродействие предложенного алгоритма и алгоритма Кейпона при реализации на CPU и GPU

Полученные результаты приведены на рисунке 2. Верхняя кривая отражает оценку быстродействия предложенного алгоритма без использования вычислений на GPU. Как видно, эта зависимость монотонно возрастает, что обусловлено необходимостью многократного повторения решения системы уравнений (20) с использованием процедуры tria. Код этой процедуры не претендует на оптимальность по быстродействию и не относится к библиотеке стандартных функций модуля Numpy Python. Вторая сверху кривая получена при реализации исходного алгоритма Кейпона с использованием библиотечных функций. Две нижние кривые получены при реализации алгоритмов с использованием вычислений на GPU. При этом штриховая кривая получена при реализации исходного алгоритма Кейпона, а сплошная нижняя кривая – с использованием предложенного алгоритма. Видно, что предложенный алгоритм ни в чем не уступает, а по эффективности использования памяти видеоадаптера превосходит исходный алгоритм Кейпона.

Оценки быстродействия алгоритмов сверхразрешения, использующих вычисления на GPU не учитывают временные затраты на загрузку данных для вычислений в память видеоадаптера. Максимальное время загрузки данных при построении пространственного спектра не превышало 40 мс.

Заключение

Предложенный алгоритм углового сверхразрешения, востребованный при обработке сигналов в цифровых антенных решетках, отличается от известного алгоритма Кейпона исключением операции обращения матрицы и использованием разложения Холецкого.

Показано, что предложенный алгоритм при построении пеленгационного рельефа требует многократного решения систем уравнений вида (20). Это приводит к тому, что данный алгоритм при реализации на CPU существенно уступает обычному алгоритму Кейпона.

Установлено, что при использовании параллельных вычислений предложенный алгоритм не уступает исходному алгоритму Кейпона с обращением КМВС по быстродействию, а также при реализации более эффективно использует память GPU. При этом допустимая размерность задачи увеличивается почти в два раза.

References

[1] R. Klemm, *Principles of Space-Time Adaptive Processing*. London: IEE, 2002.

[2] M. Parker, *Radar Basics-Part 4: Space-time adaptive processing*, 2011. [Online]. Available: http://www.eetimes.com/design/programmable-logic/4217308/Radar-Basics-Part-4-Space-time-adaptive-processing.

[3] W. Bürger, *Space-Time Adaptive Processing: Algorithms*, 2006. [Online]. Available: http://ftp.rta.nato.int/public//PubFullText/RTO/EN/RTO-EN-SET-086///EN-SET-086-07.pdf.

[4] S. Nefedov, I. Kruchkov, M. Noniashvili, G. Lesnikov, and N. Soloviev, "A Review of the Signal Space-Time Adaptive Processing (STAP) Basic Techniques in Space-Based Radars with Synthetic Aperture)", *Vestnik MGTU im N.E. Baumana. Ser. "Priborostroenie"*, no. 8, pp. 251–258, 2012.

[5] M. Ratynskij, *Adaptation and Superresolution in Array Antennas*. Moscow: Radio i svyaz, 2003.

[6] S. Tulenev, A. Savinkov, and V. Vereitin, "Application of CUDA Parallel Programming Technology to Increase the Speed Of Calculation of Superresolution Algorithms in Direction Finding", *Vestnik Voronezhskogo instituta MVD Rossii*, no. 2, pp. 196–203, 2021.

[7] *Developing a Linux Kernel Module using GPUDirect RDMA*. [Online]. Available: https://docs.nvidia.com/cuda/gpudirect-rdma/index.html.

[8] V. Voevodin, *Numerical Methods of Algebra*. Moscow: Nauka, 1966.

[9] A. Novikov, D. Gabriel'yan, E. Novikova, and N. Shatskiy, *Device to Covariance Matrix of Interference Sygnals Inversion*, Rus. Patent 2 562 389, Sept. 2015.

[10] M. Zvezdina, O. Komova, N. Shatskiy, and A. Shokov, "Hermitian matrix inversion algorithm", *Vestnik Donskogo gosudarstvennogo universiteta*, no. 2(81), pp. 78–84, 2015.

[11] F. Gantmacher, *The Theory of Matrices*. New York: Chelsea Publishing Company, 1959.

[12] T. Shoup, *A Practical Guide to Computer Methods for Engineers*. New York: Prentice-Hall, Inc., Englewood Cliffs, 1979.

[13] C. Lomont, "Fast Inverse Square Root", vol. 32, 2003. [Online]. Available: http://www.lomont.org/papers/2003/InvSqrt.pdf.

[14] N. Shilov, D. Kondratyev, I. Anureev, E. Bodin, and A. Promsky, "Platform-independent Specification and Verification of the Standard Mathematical Square Root Function", *Modeling and Analysis of Information Systems*, vol. 25, no. 6, pp. 637–666, 2018. DOI: 10.18255/1818-1015-2018-6-637-666.