**УДК 621.396.96:004.383.3**

# ОСОБЕННОСТИ РЕАЛИЗАЦИИ АЛГОРИТМОВ ОБРАБОТКИ РАДИОЛОКАЦИОННОЙ ИНФОРМАЦИИ ПОВЫШЕННОЙ СЛОЖНОСТИ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ CUDA

ГУНИЧ[1] С. П., КОЗЛОВ[2] С. В., ЛЕ[2] ВАН КЫОНГ

*[1]Республиканское научно-производственное унитарное предприятие «Центр радиотехники Национальной академии наук Беларуси»*
*[2]Белорусский государственный университет информатики и радиоэлектроники (г. Минск, Республика Беларусь)*

*E-mail: soniahunich@gmail.com*

**Аннотация.** С позиций программной реализации с использованием технологии Cuda рассмотрена структура алгоритма первичной обработки радиолокационной информации повышенной сложности при длительном когерентном накоплении (КН) отраженного сигнала. Алгоритм предусматривает вычисление спектров «быстрого» времени принимаемых сигналов в каждом периоде повторения, компенсацию миграции дальности и ее производных в спектральной области с одновременным умножением на отсчеты комплексной частотной характеристики (КЧХ) согласованного фильтра (СФ), вычисление выходных сигналов СФ в каждом периоде повторения и массивов быстрого преобразования Фурье (БПФ) в «медленном» времени по каждому столбцу полученной матрицы. Дана оценка вычислительной сложности указанного алгоритма с позиций организации параллельных вычислений на графических процессорах Nvidia. Приведены особенности реализации алгоритма обработки с использованием встроенных функций библиотеки Cuda и результаты измерения производительности при реализации алгоритма.

**Abstract.** The structure of the algorithm for the primary processing of radar information of increased complexity with long-term coherent accumulation of the reflected signal is considered from the standpoint of software implementation using the CUDA technology. This algorithm provides compensation for the migration of range for the calculation of the spectra of the "fast" time of the received signals in each repetition period. Also, algorithm derivatives calculation of the output signals of the SF in each repetition period, and arrays of fast Fourier transform in "slow" time for each column of the resulting matrix in the spectral domain with simultaneous multiplication by the samples of the complex frequency response of the matched filter. The computational complexity is estimated from the standpoint of organizing parallel computations on Nvidia GPUs. The implementation features are given as the usage of the built-in functions of the CUDA library and as the results of performance measurement.

## Постановка задачи

Рассматриваемый алгоритм представляет собой экономную версию алгоритма длительного КН отраженного сигнала [1] на основе последовательности операций БПФ/обратного БПФ по «быстрому» (строки) и «медленному» (столбцы) времени с промежуточной коррекцией миграции дальности и ее производных. Входными данными для алгоритма является матрица $N_r \times N_s$ комплексных отсчетов $\dot{U}_{k,m}$ принимаемой реализации, где $m=\overline{1,N_s}$ - индекс «быстрого» времени в пределах одного периода повторения; $k=\overline{1,N_r}$ - индекс «медленного» времени по периодам повторения. Блок-схема алгоритма приведена на рис. 1., где сбоку от каждого блока указано минимальное число операций комплексного умножения для его реализации. Алгоритм предполагает вычисление спектров принимаемой реализации по $N_s$ отсчетам в $N_r$ периодах повторения, коррекцию отсчетов полученного множества спектров по ожидаемым параметрам движения цели $\mathbf{b}$, умножения спектров на отсчеты $\dot{G}_m^{\text{сф}}$ КЧХ согласованного фильтра, вычисление обратного БПФ по «быстрому» времени и заключительной операции БПФ по «медленному» времени (столбцам) с последующим переходам к квадратам модулей полученного спектра. Пунктиром показан вариант, когда матрица $\mathbf{Q}$ корректирующих коэффициентов рассчитывается предварительно и загружается из памяти. Возможные значения $N_s = 2^n$, $n = 12...18$ и $N_r = 50...300$. С учетом проведения БПФ по целой степени 2, перед заключительным БПФ по медленному времени осуществляется дополнение нулями

результатов обратного БПФ до $N_r^{\text{бпф}} = 2^{[\log_2 N_r]}$ отсчетов, где $[x]$ - операция выделение ближайшего большего целого.

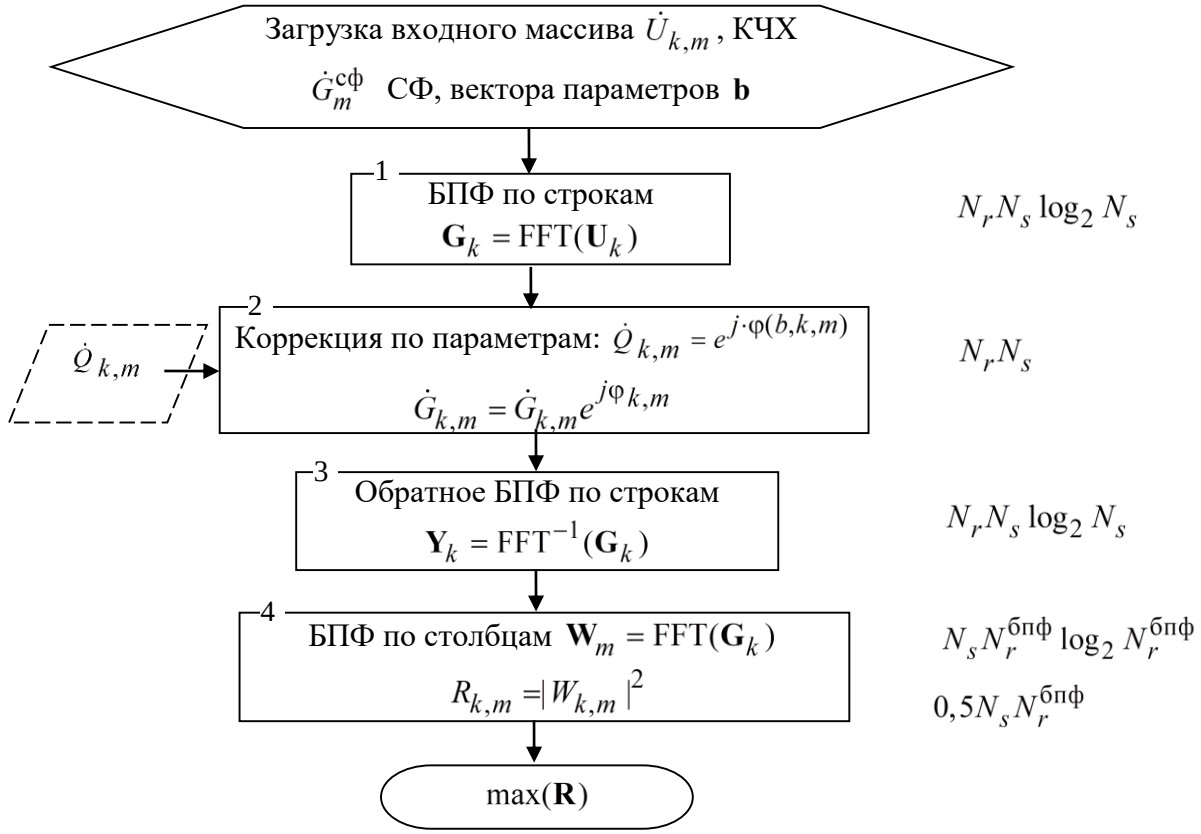



**Рис. 1.** Блок-схема алгоритма

Общее число операций комплексного умножения при однократной реализации и загрузкой предварительно рассчитанной матрицы $\mathbf{Q}$ оценивается величиной

$$O_1 = 2N_r N_s \log_2 N_s + N_r N_s + N_s N_r^{\text{бпф}} \log_2 N_r^{\text{бпф}} + 0{,}5 N_s N_r^{\text{бпф}}. \quad (1)$$

Для $n = 18$, $N_r = N_r^{\text{бпф}} = 128$ имеем $O_1 = 1{,}5$ GFLOP (миллиардов умножений). С учетом того, что операция комплексного умножения соответствует четырем операциям обычного умножения, общее число умножения составит $O_1^{/} = 6$ GFLOP. При расчете матрицы $\mathbf{Q}$ «на лету» требуемое число операций возрастает на 20…30%.

Для современных универсальных процессоров с производительностью ~50 GFLOPS теоретическое время расчета алгоритма для указанного случая составит около 0,12 с. Однако реальная производительность из-за ограничений по быстродействию памяти (обрабатываемые матрицы имеют достаточно большой размер), неоптимального построения вычислений и других факторов оказывается существенно ниже. Так, при случая $n = 12$, $N_r = N_r^{\text{бпф}} = 128$, когда $O_1^{/} \approx 0{,}07$ GFLOP процессор Intel Core i7 (1,07-4,2 ГГц) с паспортной производительностью около 30 GFLOPS вычисляет данный алгоритм за 649 мс, то есть реальная производительность составляет всего 0,12 GFLOPS. Это практически исключает возможность реализации алгоритма в реальном масштабе времени и определяет необходимость поиска соответствующих аппаратно-программных решений. Эти решения должны быть основаны, преимущественно, на механизме параллельных вычислений [2].

Несмотря на то, что современные универсальные процессоры поддерживают механизм параллелизма, но он достаточно ограничен операционной системой. Существует так же другие реализации такие как MATLAB, но его инструментарий достаточно узок и, в основном, подходит для математического представления и анализа работы данного алгоритма. Одним из наиболее перспективных является использование графических процессоров, в частности, графических процессоров производства Nvidia с поддержкой технологии CUDA – архитектуры, предложенной для

осуществления вычислений [3, 4]. Архитектура позволяет разработчику по собственному усмотрению осуществлять доступ к ускорителям, организовав довольно сложный вычислительный процесс. Программная архитектура представлена (как CUDA SDK) с API, базирующейся на модифицированной версии языке Си. Использование графического процессора позволяет разгрузить центральный процессор, тем самым позволяя выполнять дополнительные действия.

**Особенности и примеры организации вычислений**

Основное отличие от традиционного программирования на центральном процессоре является разделение на типы доступа к памяти, основными из которых являются host и device. Функции хоста запускаются только с центрального процессора и выполнятся только на центральном процессоре. Функции устройства запускаются на графическом процессоре и вызываются на графическом процессоре. Функции со спецификатором global (глобальные) вызываются центральным процессором, однако выполнения блока происходит на графическом процессоре [3, 4].

В особенности работы с функциями global можно отнести то, что можно распределить потоки (threads) в блоки и запускать на обработку определенное количество блоков и потоков. Количество потоков, блоков и сетки ограничено физически.

```
dim3 blocksize, gridsize;
int threadnum = 1024;
blocksize = dim3(threadnum, 1, 1);
gridsize = dim3((int)Ns / threadnum, (int)Nr, 1);
```

Вызов функции, где используется блоки и потоки осуществляется следующим образом:

```
Main_Realisation <<< gridsize, blocksize >>> (список параметров);
KChH <<< blocksize, threadnum >>> (список входных параметров);
```

В тройных кавычках указывается размерность сетки и размер блока, либо же количество потоков и количество выполнений.

Массивы, с которыми ведется работа выделяются непосредственно на графическом процессоре, функциями хоста. Однако необходимо учитывать тип данных, который будет хранится в массиве – память выделяется в байтах. Очистка памяти производится аналогично со стандартным принципом C++.

```
cudaMallocManaged(&входной массив, Ns*Nr*sizeof(cufftComplex)));
```

Так как технология работы с графическим процессором иная нежели с центральным из-за механизма параллельных вычислений, то работа с циклами выглядит совершенно иначе. Если при работе с ЦП цикл прохода по массиву использует конструкции for (int I = 0; I < 10; i++) или while (I < 10), то в ГП цикл прохода по массиву выглядит так:

```
int m = blockIdx.x * blockDim.x + threadIdx.x;
int k = blockIdx.y * blockDim.y + threadIdx.y;
int index = m * Nr + k;

if ((k < Nr) && (m < Ns))
{
      // функциональный блок
}
```

Основное различие в подходах заключается в:

1. Использовании условного оператора if ();

2. Итератор представляет собой переменную, которая содержит информацию о потоке (индекс блока внутри сетки, размер одного блока в потоках, индекс потока внутри блока), который работает над функцией в данный момент.

Для расчета БПФ и ОБПФ использовалась дополнительная библиотека, специально разработанная компанией Nvidia для вычислений преобразований Фурье – CUFFT. Специфика ее

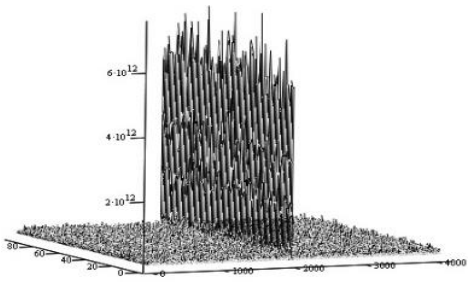
работы такова, что для вывоза функций необходимо иметь плоскость (plane), которая будет работать с одно-, двух- или трехмерными массивами и обработчик (handler). Если же необходимо обработать какую-то специфическую последовательность значений, то это можно реализовать с помощью cufftplanMany. К примеру, реализация плоскости, через которую рассчитывается БПФ по строкам.

```
int rank_1 = 1;          // размерность массива (одно-, двух-, трехмерный)
int n1[] = { Ns };       // длина массива
int istride1 = Nr,
    ostride1 = Nr;       // расстояние между 2 последовательными элементами
int idist1 = 1,
    odist1 = 1;          // расстояние между блоками выполнения
int inembed1[] = { 0 }; // длина массива ввода со смещением
int onembed1[] = { 0 }; // длина массива вывода со смещением
int batch1 = Nr;         // количество выполнений
cufftPlanMany(&handle_r, rank_1, n1,
              inembed1, istride1, idist1,
              onembed1, ostride1, odist1, CUFFT_C2C, batch1);
```

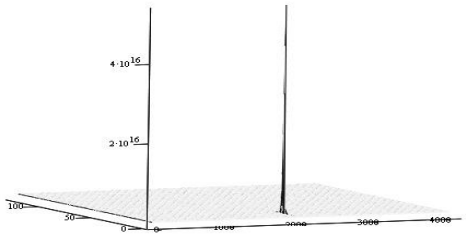
Сама по себе программа, реализующая алгоритм, линейна, то есть внутри нет условных операторов или циклов, выполняющихся на центральном процессоре.

**Оценка производительности**

На рис. 2. приведены результаты реализации работы алгоритма (для визуализации радиолокационного изображения использовался пакет *Mathcad*-15), иллюстрирующая правильность его реализации.



a) выход блока 3 – сжатые сигналы с компенсацией миграции дальности



b) выход блока 4 – результирующее РЛИ

**Рис. 2.** Радиолокационные изображения на различных этапах реализации алгоритма

В табл. 1 приведены результаты оценки быстродействия и объема используемой видеопамяти для двух видеокарт среднего уровня: GeForce 1050 (728 процессоров, 2 GB, GDDR5 на частоте 7 ГГц) и GeForce 1660ti (1024 процессоров, 6 GB, GDDDR5 на частоте 3 ГГц).

Рассматривались два варианта обработки, когда все исходные/рассчитываемые массивы хранились в памяти и когда исходные массивы (более не нужные) замещался в памяти рассчитываемым. Дополнительно исследовалось быстродействие в зависимости от числа программно задаваемых нитей (потоков).

Как следует из приведенных в табл. 1 результатов:

- время выполнения алгоритма на рассматриваемых видеокартах в 48…166 раз меньше времени выполнения на процессоре Intel Core i7;

- реальная производительность видеокарт для рассматриваемого алгоритма составляет 6,0…7,7 GFLOPS и 14,1…20,5 GFLOPS; производительность операций БПФ/ОБПФ в числе отсчетов входных сигналов в секунду примерно совпадает с данными, приведенными в [5]; аналогично [5] можно сделать вывод, что реальная производительность при реализации рассматриваемого алгоритма обработки радиолокационной информации ограничена пропускной способностью памяти, а не вычислительными возможностями устройства;

- при сокращении объем используемой памяти за счет замещения массивов производительность для видеокарты GeForce 1050 не изменяется, а для видеокарты GeForce 1660ti увеличивается до 30%, что можно объяснить более низкой частотой работы памяти для указанной видеокарты;
- производительность напрямую не зависит от числа программно задаваемых нитей (потоков);
- увеличение числа физических процессоров в видеокарте приводит к примерно кратному снижению времени вычислений, то есть алгоритм обладает высоким естественным параллелизмом, а компилятор обеспечивает эффективное распараллеливание вычислений;
- для рассматриваемого алгоритма проблема объема памяти является существенной и для определенных случаев может привести к ограничениям в реализации алгоритма обработки информации.

**Таблица 1.** Время выполнения алгоритма, реальная производительность и объем памяти для массивов данных при хранении всех / замещении массивов в памяти

| | Видеокарта GeForce 1050 | | | GeForce 1660ti | | |
|---|---|---|---|---|---|---|
| $N_s$ | *t*, мс | S, GFLOPS | Объем, МБ | *t*, мс | S, GFLOPS | Объем, МБ |
| 4096 | 12,6 / 13,4 | 6,4 / 6,0 | 492/ 471 | 5,7 / 3,9 | 14,1 / 20,5 | 1235 / 1214 |
| 8192 | 26,1 / 25,8 | 6,6 / 6,6 | 550 / 509 | 12,6 / 8,4 | 13,6 / 20,4 | 1293 / 1251 |
| 16384 | 53,1 / 55,1 | 6,8 / 6,6 | 665 / 583 | 25,1 / 20,4 | 14,4 / 17,8 | 1408 /1325 |
| 32768 | 109,5 / 108,9 | 7,0 / 7,1 | 895 / 731 | 54,4 / 44,5 | 14,1 / 17,2 | 1638 / 1473 |
| 65536 | 223,4 / 220,1 | 7,2 / 7,3 | 1365 / 1027 | 112,9 / 95,2 | 14,3 / 16,1 | 2098 / 1769 |
| 131072 | – / 441,5 | - / 7,7 | – / 1719 | 225,9 / 188,4 | 15,0 / 18,0 | 3020 / 2361 |
| 262144 | – / – | – / - | – / – | 453,5 / 365,1 | 15,6 / 19,4 | 4864 / 3549 |

Дальнейшие исследования по повышения производительности следует сосредоточить на использование page-locked памяти, развертки циклов, использовании атомарных операций.

### Список использованных источников

1. Козлов С. В., Ле Ван Кыонг. Экономичные алгоритмы длительного когерентного накопления отраженного сигнала при наличии миграции по дальности и ее производным. В настоящем сборнике.
2. Голутвин Р. И., Красилов А. А. Применение технологии CUDA для обработки радиолокационных данных // Информационные технологии и системы 2013 (ИТС 2013): материалы международной научной конференции, БГУИР, Минск, Беларусь, 23 октября 2013 г. – Information Technologies and Systems 2013 (ITS 2013) / редкол.: Л.Ю. Шилин [и др.]. - Минск: БГУИР, 2013. – С. 264–265.
3. Сандерс Дж., Кэндрот Э. Технология CUDA в примерах: введение в программирование графических процессоров. – М. Издательство ДМК-Пресс, 2011. – 230 с.
4. Параллельные вычисления на GPU. Архитектура и программная модель CUDA: Учеб. пособие / А. В. Боресков и др. – 2-е изд. – М.: Издательство Московского университета,2015. – 336 с.
5. Пантелеев А. Ю. Цифровая обработка сигналов на современных графических процессорах // Цифровая обработка сигналов, № 3, 2012. – С 68-75.