Декодирование изображений в формате JPEG на процессорах КОМДИВ

П. Б. Богданов, О. Ю. Сударева

Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", г. Москва, Россия

Аннотация. В работе оценивается возможность применения специализированного массивно-параллельного SIMD-сопроцессора вещественной и комплексной арифметики СР2, разработанного и поддерживаемого в некоторых решениях ФГУ ФНЦ НИИСИ РАН, в задачах сжатия цифровых изображений. На примере стандарта сжатия цифровых изображений JPEG проведен анализ алгоритма декодирования JPEG-изображений с целью его дальнейшей реализации на СР2.

Ключевые слова: КОМДИВ, СР2, JPEG, JFIF, ДКП, libjpeg-turbo.

DOI 10.14357/20718632190101

Введение

ФГУ ФНЦ НИИСИ РАН производит широкую линейку аппаратных средств для высокопроизводительных вычислительных систем, в том числе универсальные и специализированные процессоры. Специализированные микропроцессоры оснащаются математическим массивно-параллельным SIMD-сопроцессором цифровой обработки сигналов СР2, который предназначен для эффективных параллельных вычислений в задачах потоковой обработки сигналов — в частности, для вычисления быстрого преобразования Фурье (БПФ) [1, 2]. Вместе с тем, использование СР2 может дать существенный прирост производительности по сравнению с вычислениями на управляющем ядре (CPU) и на других вычислительных алгоритмах, связанных с массивно-параллельной обработкой больших массивов данных [3, 4]. Последняя версия сопроцессора СР2 реализована в микропроцессоре под кодовым названием КОМДИВ128-М (далее ВМ9).

Операция кодирования цифрового видео, звука или, в нашем случае, изображения для рядового пользователя чаще всего не является актуальной, так как зачастую является разовой и требует довольно больших как вычислительных ресурсов, так и ресурсов по хранению необработанных, «сырых» данных. В добавок к этому, обработкой и упаковкой сырых данных обычно занимаются сами производители цифрового контента. Рядовому пользователю ПК, планшета или смартфона как потребителю контента чаще всего требуется обратная операция — операция декодирования сжатых данных. Иными словами, из двух операций — кодирования и декодирования — вторая выполняется чаще и предъявляет более строгие требования к производительности.

Одним из наиболее распространённых форматов хранения и передачи изображений в сжатом виде является формат JPEG [5, 6]. Имеется множество исследований, посвящённых эффективной реализации декодирования JPEG на универсальных процессорах и системах других

архитектур — например, на графических ускорителях (GPU).

В данной работе рассматривается возможная реализация декодирования JPEG на процессоре ВМ9. Поскольку большинство стадий декодирования исполняются независимо для отдельных фрагментов изображения и могут быть легко распараллелены, представляется разумной реализация этих стадий на СР2. Такая реализация подразумевает низкоуровневую оптимизацию процедур с учётом особенностей архитектуры сопроцессора и эффективную реализацию обменов данными через канал DMA.

1. Реализация на основе библиотеки libjpeg-turbo

Одним из наиболее известных «кодеков» JPEG, то есть библиотек, предоставляющих как процедуры кодирования, так и декодирования, является библиотека с открытым исходным кодом libjpeg-turbo [7]. Библиотека используется во множестве приложений, включая популярные веб-браузеры и дистрибутивы ОС Linux. В данной библиотеке многие из вычислительных процедур, на которые подразделяется кодирование и декодирование JPEG, оптимизированы под SIMD-расширения известных универсальных процессоров — например, расширения SSE2 и AVX2 процессоров от Intel. Оптимизированные процедуры написаны на языках ассемблера соответствующих архитектур.

При реализации декодирования на КОМДИВ, как будет показано далее, не все вычислительные процедуры имеет смысл оптимизировать на СР2. Кроме того, на CPU должны выполняться все вспомогательные операции: обработка команд пользователя, чтение входного файла и т.п.. Поэтому для реализации библиотеки в качестве основы работы с форматом JPEG для КОМДИВ была выбрана libjpeg-turbo, в которой планируется заменить ряд фрагментов кода, выполняющих основные вычисления, вызовами разработанных оптимизированных процедур для СР2 (тот же приём применялся ранее, например, при реализации на КОМДИВ нескольких тестов из NAS Parallel Benchmarks [3, 4]). Вообще говоря, CPU процессора ВМ9 имеет, помимо сопроцессора СР2, векторное расширение CPV [8], и в перспективе возможно его использование для дополнительного повышения производительности библиотеки.

2. Обзор формата JPEG

Стандарт JPEG [5] поддерживает сжатие изображений с разным количеством каналов и определяет целый класс различных процедур сжатия. Для передачи изображений по сети Интернет — как цветных, состоящих из трёх каналов, так и одноканальных монохромных — наиболее широкое распространение получила базовая последовательная процедура сжатия. Для цветного RGB-изображения эта процедура подразделяется на несколько стадий [5, 9, 10]:

- преобразование из цветового пространства RGB в YCbCr (также в некоторых источниках называемое YUV): Y — яркость, Cb и Cr — два канала цветности;
- прореживание каналов Cb и Cr по одной из стандартных схем;
- разбиение каждого канала на блоки размера 8×8 (DU, Data Unit), группировка блоков трёх каналов в MCU (Minimal Coded Unit — минимальный фрагмент изображения, который может быть закодирован);
- для каждого блока каждого канала:
  - двумерное дискретное косинусное преобразование (ДКП);
  - квантование — поэлементное деление на заданную матрицу с округлением результатов до целых;
  - кодирование повторов в последовательности элементов блока в порядке «зиг-заг»;
  - энтропийное кодирование.

Декодирование представляет собой выполнение обратных операций в обратном порядке. Часть деталей представления сжатого изображения, которые существенны для процедуры декодирования, отражена в стандарте JFIF [6] — формата файлов с изображениями JPEG. Как правило, говоря о «файлах JPEG», подразумевают именно JPEG/JFIF. На исходное изображение накладываются ограничения: оно может иметь только 1 канал или 3 канала (RGB), и семплы (пиксели) каждого канала должны быть 8-битными. Ниже везде подразумевается, что изображение поступает на вход декодировщика в виде JFIF-файла.

Файл подразделяется на ряд сегментов, в которые в частности записаны различные параметры изображения и информация, необходимая для декодирования: размер исходного изображения, использованная схема прореживания, коэффициенты квантования, таблицы, использованные для энтропийного кодирования, и т.д. Начало каждого очередного сегмента файла отмечено специальным маркером. В отдельном сегменте располагается «скан» — само изображение в закодированном виде. В скане могут встречаться специальные рестарт-маркеры — они разделяют фрагменты скана, которые могут быть декодированы независимо.

Скан изображения представляет собой последовательность закодированных DU. В чёрно-белом изображении все DU соответствуют каналу Y, в цветном DU трёх каналов группируются в MCU и записываются в общую последовательность. Количество DU в одном MCU и порядок записи можно восстановить по информации в заголовке файла. В каждом DU выделяется DC-коэффициент — левый верхний в блоке размера 8×8 — и AC-коэффициенты — все остальные. Коэффициенты этих двух типов закодированы различными способами.

В данной работе рассматривается наиболее популярная базовая последовательная схема сжатия JPEG. Отличительной особенностью этой схемы является, в частности, использование кодов Хаффмана для энтропийного кодирования. Отметим, что в качестве альтернативы кодам Хаффмана в стандарте JPEG предлагается арифметическое кодирование, однако оно не поддерживается в полной мере многими приложениями, поскольку долгое время было защищено патентами. Арифметическое кодирование позволяет добиться более высокой степени сжатия, чем код Хаффмана (примерно на 10%), но процедуры кодирования и декодирования более сложны алгоритмически и требуют больше времени. В базовой схеме сжатия арифметическое кодирование не предусмотрено.

Опишем кратко вычисления, выполняемые на каждой из стадий декодирования.

2.1. Декодирование кода Хаффмана

Сущность кода Хаффмана состоит в том, что чем чаще некоторый символ встречается в последовательности, тем меньшим количеством бит он кодируется [11]. На вход процедуры декодирования поступает закодированная последовательность и таблица, которая использовалась при кодировании.

Что касается формата JPEG, здесь необходимо учитывать, что не вся последовательность бит в скане изображения представляет собой сообщение, закодированное кодом Хаффмана — подробнее в это будет описано в следующем разделе. Кроме того, при кодировании применяется дополнительная техника «заполнения байт» (byte stuffing), в результате которой в скане появляются дополнительные биты — при декодировании их следует пропускать.

Таблицы Хаффмана восстанавливаются перед началом декодирования по информации из соответствующих сегментов файла JFIF. В общем случае различаются таблицы для каналов яркости и цветности, а для каждого канала — для AC- и DC-коэффициентов.

2.2. Декодирование повторов

Рассмотрим, что происходит в процедуре кодирования на стадии «зиг-заг»-упорядочения элементов блока и кодирования повторов. На вход поступает DU — блок целых чисел размера 8×8, упорядоченный в памяти по строкам, который обладает следующим свойством: ненулевые элементы сконцентрированы в левом верхнем углу, и наибольшим по модулю, как правило, является DC-коэффициент. Сначала элементы блока записываются в вектор длины 64 в порядке «заг-заг», показанном на Рис. 1. В полученном векторе первым идёт DC-коэффициент, а затем все AC-коэффициенты, причём с какого-то момента все они равны 0.

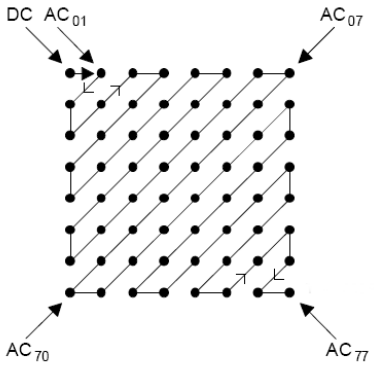

Рис 1. Порядок элементов блока «зиг-заг»

DC-коэффициент заменяется на разность между ним и DC-коэффициентом предыдущего вектора (разности вычисляются для каждого из каналов Y, Cb, Cr по отдельности); для первого вектора предыдущий DC-коэффициент считается нулевым. Далее, перед DC-коэффициентом вставляется специальный символ — количество бит для его представления. Перед каждым ненулевым AC-коэффициентом серия предшествующий нулей (в т.ч. пустая) заменяется на специальный символ другого вида — пару [количество нулей в серии, количество бит для представления ненулевого числа]. Серия нулей в конце вектора заменяется на символ «EOB».

Специальные символы перед DC- и AC- коэффициентами независимо кодируются двумя различными кодами Хаффмана. В выходной последовательности чередуются эти коды и сами коэффициенты, специальным образом представленные в двоичном виде.

Теперь рассмотрим процедуру декодирования. Для одного блока необходимо выполнить следующие действия.

1. Декодировать по таблице Хаффмана для спец. символов перед DC-коэффициентами один символ, начиная с нулевого бита последовательности. Результатом будет символ $[n_0]$ — количество следующих бит в последовательности, представляющих DC-коэффициент.

2. Восстановить значение DC-коэффициента по следующим n0 битам, записать его в выходную последовательность.

3. Декодировать с текущей позиции (после бит, представляющих DC-коэффициент) один символ по таблице Хаффмана для специальных символов перед AC-коэффициентами. Результатом будет символ $[k_1, n_1]$.

4. Записать в выходную последовательность k_1 нулей.

5. Восстановить значение первого ненулевого AC-коэффициента по следующим n_1 битам, записать его в выходную последовательность.

6. Снова декодировать один символ, перед вторым AC-коэффициентом...

7. И так далее, пока во входной последовательности результатом декодирования очередного символа не будет EOB (в этом случае выходная последовательность дополняется нулями до длины 64), либо пока в выходной последовательности не будет 64 элемента.

8. Полученная последовательность из 64 элементов записывается в матрицу размера 8×8 в порядке «зиг-заг». На этом декодирование одного DU закончено.

Существенной особенностью данной процедуры является то невозможность установить, где в последовательности находится начало очередного кода спец. символа или коэффициента, до тех пор, пока не декодированы все предыдущие символы и коэффициенты. Различные символы и коэффициенты кодируются разным, не известным заранее количеством бит. В частности, невозможно установить границы отдельных DU в скане изображения.

2.3. Деквантование

Операция квантования при кодировании изображения представляет собой поэлементное деление каждого DU на заданную таблицу коэффициентов, т.е. также матрицу порядка 8×8, с округлением результатов до ближайших целых. Таблицы, при помощи которых было закодировано изображение, записываются в соответствующий сегмент файла. В общем случае различаются таблицы квантования для каналов яркости и цветности.

Перед началом декодирования все необходимые таблицы считываются из файла. Деквантование каждого DU состоит в том, что блок, полученный после декодирования повторов, поэлементно домножается на соответствующую таблицу.

2.4. Обратное ДКП

Дискретное косинусное преобразование (ДКП) — процедура, родственная дискретному преобразованию Фурье (ДПФ): входной сигнал представляется как сумма заданного набора синусоид с некоторыми коэффициентами [12]. Обратное ДКП восстанавливает входной сигнал по набору коэффициентов. Общие формулы для вычисления прямого и обратного одномерного ДКП вектора длины N следующие:

$$\boldsymbol{F[k]} = \sqrt{\frac{2}{N}}\boldsymbol{c(k)} \sum_{\substack{\boldsymbol{j=0} \\ \boldsymbol{k=\overline{0,N-1}}}}^{\boldsymbol{N-1}} \boldsymbol{f[j]cos}\frac{\boldsymbol{(2j+1)k\pi}}{\boldsymbol{2N}}, \quad (1)$$

$$\boldsymbol{f[j]} = \sqrt{\frac{\boldsymbol{2}}{\boldsymbol{N}}} \sum_{\boldsymbol{k=0}}^{\boldsymbol{N-1}} \boldsymbol{c(k)F[k]cos}\frac{\boldsymbol{(2j+1)k\pi}}{\boldsymbol{2N}},$$
$$\boldsymbol{j} = \overline{\boldsymbol{0, N-1}},$$

где $c(k) = 1/\sqrt{2}$ при $k = 0$ и 1 при $k \neq 0$. Преобразование переводит вещественные вектора в вещественные.

В процедуре сжатия выполняется двумерное ДКП для каждого DU, т.е. ДКП порядка 8×8. Формула для обратного двумерного преобразования следующая:

$$f[j,k] = \\ = \frac{2}{N}\sum_{p=0}^{7}\sum_{q=0}^{7} c(p)c(q)F[p,q]cos\frac{(2j+1)p\pi}{16}cos\frac{(2k+1)q\pi}{16}, \\ j,k = \overline{0,7} \qquad (2)$$

Нетрудно увидеть, что данное двумерное преобразование может выполняться, как и ДПФ, путём последовательного вычисления набора одномерных обратных ДКП длины 8 по столбцам, а затем по строкам.

В процедуре декодирования после вычисления обратного ДКП результаты округляются до ближайшего целого.

2.5. Интерполяция

Интерполяция — процедура, обратная прореживанию, которое состоит в том, что несколько соседних значений заменяются на одно. В процедуре сжатия прореживание применяется к каналам Cb и Cr, но не Y, поскольку человеческий глаз менее чувствителен к потере качества в цветности, чем в яркости. Существует целый набор различных схем прореживания и специальная нотация для их обозначения [13]. Как правило, для прореживания каналов Cb и Cr используется одна и та же схема.

Наиболее распространёнными являются схемы прореживания 4:2:0 и 4:2:2, проиллюстрированные на Рис. 2. В первом случае каждые четыре соседних значения, образующих блок 2×2, заменяются на одно, во втором случае — два соседних по горизонтали.

В результате, например, для схемы 4:2:0 получается, что после прореживания каждой паре блоков размера 8×8 каналов Cb и Cr соответствует фрагмент изображения 16×16 пикселей, а значит и блок канала Y размера 16×16, который разбивается на 4 блока 8×8. Этот набор и составляет один MCU: 4 DU (блока) Y, один DU Cb, один DU Cr. Аналогично, для схемы 4:2:2 один MCU — это 2 DU Y, один DU Cb и один DU Cr.

При декодировании на вход процедуре интерполяции поступают последовательности DU, соответствующие каналам Cb и Cr. Информация о том, какая именно из схем прореживания к ним применялась, записана в заголовке файла. В зависимости от реализации, интерполяция может вычисляться по-разному, с применением различных фильтров. Например, в библиотеке libjpeg-turbo реализовано два вида фильтров: «прямоугольный» фильтр, который копирует в выходные пиксели значения ближайших входных, и «треугольный», который осуществляет линейную интерполяцию по значениям в двух соседних входных пикселях. Эти два варианта интерполяции для схемы 4:2:2 и фрагмента одной строки значений одного блока представлены на Рис. 3 (C означает входные значения — Cb или Cr).

2.6. Преобразование в цветовое пространство RGB

В результате интерполяции получается набор блоков размера 8×8, в котором каждому фрагменту изображения из 8×8 пикселей соответствует 3 блока: каналы Y, Cb и Cr. Соответственно, на каждый пиксель приходится по три значения. Для представления изображения на

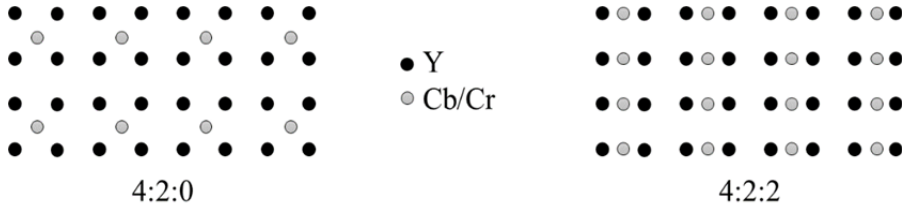

Рис. 2. Схемы прореживания 4:2:0 и 4:2:2

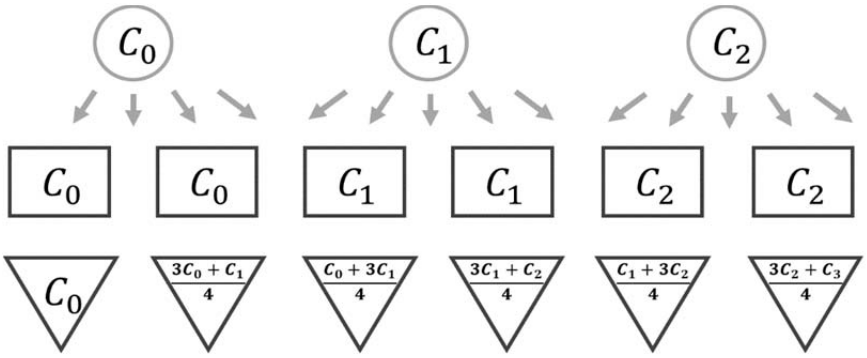

Рис. 3. Два способа интерполяции значений для схемы 4:2:2

экране используется цветовое пространство RGB — также по 3 значения на пиксель. Для каждого пикселя эти значения вычисляются по следующим формулам:

$$\begin{cases} R = Y + 1{,}40200(Cr - 128) \\ G = Y - 0{,}34414(Cb - 128) - 0{,}71414(Cr - 128). \\ B = Y + 1{,}72200(Cb - 128) \end{cases} \quad (3)$$

Здесь и входные, и выходные значения являются 8-битными целыми: результаты вычислений округляются. В формулах из значений Cb и Cr вычитается 128, т.к. в процедуру кодирования обычно, в том числе в libjpeg-turbo, добавляется сдвиг на 128, что позволяет уменьшить объём сжатого изображения.

На этой стадии необходимо учесть, что исходное изображение размера $M \times N$ могло не делиться на целое количество блоков 8×8, а также на целое количество MCU — это означает, что перед кодированием оно было дополнено нужным количеством строк и/или столбцов пикселей. Значения N и M записаны в файл — соответственно, те части блоков из последнего столбца и последней строки, которые выходят за границы реального изображения, необходимо отбросить.

На выходе процедуры получается последовательность целых чисел вида: R0, G0, B0, R1, G1, B1, R2, G2, B2, ... — здесь тройки соседних значений соответствуют трём каналам одного пикселя. Этот массив соответствует декодированному изображению, в котором пиксели упорядочены по строкам. Для отображения на экране к этим значениям могут применяться дополнительные преобразования, такие как масштабирование, приведение к цветовому пространству, поддерживаемому монитором, и т.п.

3. Оптимизация отдельных стадий декодирования на CP2

3.1. Декодирование кода Хаффмана и декодирование повторов

Как отмечалось в разделе 2.2, процедура декодирования кода Хаффмана, которая в случае JPEG сочетается с декодированием повторов, является существенно последовательной. Тем не менее, попытки всё же распараллелить декодирование на различных архитектурах предпринимались неоднократно. В [14] предлагается распараллелить декодирование, пользуясь следующим свойством кодов Хаффмана: если начать декодировать последовательность с произвольного бита, то сначала какое-то количество символов будет декодировано неправильно, но с некоторого момента произойдёт «синхронизация», и дальнейшая последовательность будет декодирована верно. Декодирование параллельно на двух процессорах можно организовать следующим образом: первый процессор начинает декодирование с начала последовательности, а второй тем временем — с произвольного места k_1. При этом запоминаются начала всех найденных кодовых слов. Первый процессор декодирует последовательность до позиции k_1 и дальше, до тех пор, пока начало очередного кодового слова не совпадёт с началом кодового слова, найденным ранее вторым процессором. Совпадение означает, что далее последовательность уже верно декодирована вторым процессором, поэтому первый процессор может переключиться на декодирование с новой позиции k_2. Второй процессор декодирует до k_2 и далее, до тех пор, пока снова не будет совпадения начала кодового слова

с тем, что уже декодировано первым процессором, и так далее.

Описанный подход можно распространить на произвольное количество процессоров или процессорных ядер и таким образом организовать параллельное декодирование на многоядерных CPU. Впоследствии данный подход был запатентован для декодирования на GPU ([15]). Однако если говорить о реализации на СР2, препятствием здесь является то, что вычислительные секции SIMD-сопроцессора СР2 обладают меньшей независимостью, чем вычислительные элементы GPU или, тем более, ядра CPU. Все секции СР2 исполняют параллельно одну и ту же последовательность инструкций. В описанной процедуре же каждый процессор декодирует своё, не известное заранее, количество кодовых слов, прежде чем происходит синхронизация.

В [16] запатентован другой способ параллельного декодирования кодов с переменной длиной строки в целом — для этого предлагается при кодировании дополнить последовательность специальными маркерами так, чтобы вычислительная нагрузка на декодирование различных фрагментов между маркерами была приблизительно равной. При этом необходимость описанной выше синхронизации отпадает. Стандартом JPEG действительно предусмотрены такие рестарт-маркеры, и некоторые кодировщики включают их в скан изображения, но при декодировании полагаться на наличие маркеров во входном файле нельзя.

В [17] предложен ещё один метод параллельного декодирования на GPU. Вычислительные ресурсы GPU позволяют сделать избыточное декодирование: входная последовательность декодируется с каждого бита — на выходе получается последовательность, в которой лишь некоторые значения являются «настоящими». Далее при помощи специальной процедуры фильтрации в этой последовательности находятся символы конца блоков (EOB), которые могут быть границами реальных блоков. Зная границы блоков, можно произвести окончательное декодирование последовательности. Данная процедура опирается на то, что в конце каждого блока изображения присутствует символ EOB — в реальности это не всегда так, и может потребоваться дополнительная логика. По итогам реализации оказывается, что процедура фильтрации блоков на GPU занимает больше времени, чем декодирование на CPU. Если говорить о реализации на СР2, она, скорее всего, окажется ещё менее эффективной, т.к. процедура фильтрации задействует большое количество условных переходов, для которых архитектура СР2 не приспособлена.

Помимо описанных трудностей, необходимо иметь в виду, что даже если бы начала блоков в закодированной последовательности были заранее известны (например, отмечены маркерами), либо удалось их вычислить, это не означает, что они были бы выравнены. Для пересылки же данных на СР2 требуется выравнивание адресов на 16 байт (потенциально 8 байт, или 32 бита). Кроме того, в любом случае, не совпадают длины кодовых слов в различных блоках, поэтому потребуется чтение по разным адресам в разных секциях. Ранее, при реализации на СР2 процедуры умножения разреженной матрицы на вектор, уже одно это обстоятельство привело к тому, что количество тактов на это раздельное чтение в несколько раз превысило количество тактов на сами вычисления. Наконец, сама процедура чтения кодового слова — последовательности бит произвольной длины — и поиска его в таблице в одной секции СР2 представляется сложной для реализации, т.к. на СР2 доступно только условное выполнение одной арифметической команды и цикла n раз, а сравнение можно выполнить только для 64-разрядных регистров.

Также нужно учесть, что скан JPEG-изображения представляет собой не просто последовательность, закодированную кодом Хаффмана, а чередование кодовых слов и значений, причём чередуются также блоки разных каналов и типы коэффициентов, для которых используются разные таблицы (раздел 2.2) — всё это означает усложнение процедуры декодирования. Кроме того, объём памяти СР2 может оказаться недостаточным для хранения всех таблиц.

Из всего сказанного можно сделать вывод, что перенос декодирования кодов Хаффмана и повторов на СР2 скажется на итоговой производительности процедуры отрицательно. Отметим, что в библиотеке libjpeg-turbo этот этап

декодирования также не оптимизирован на SIMD-расширениях процессоров. Более того, за исключением энтропийного декодирования, все этапы кодирования и декодирования имеют SIMD-реализации.

3.2. Деквантование

Поэлементное умножение каждого блока на заданную матрицу — простейшая операция линейной алгебры, в которой вычисления выполняются независимо для различных блоков и различных элементов одного блока. Её эффективная реализация на SIMD-архитектуре CP2 не представляет трудности. Ряд похожих процедур был ранее успешно оптимизирован на CP2 в рамках библиотеки цифровой обработки сигналов (БЦОС).

Блоки изображения могут обрабатываться по 4, параллельно в четырёх секциях CP2. При этом блоки, относящиеся к одному каналу изображения, умножаются на одну и ту же матрицу, т.е. всего требуется 3 матрицы размера 8×8. С учётом дальнейших вычислений (разд. 3.5), обработку удобно организовать следующим образом: в каждой секции за один приём обрабатывать все блоки каналов Y, Cb и Cr, относящиеся к одному или нескольким MCU. При этом в памяти каждой из секций должны постоянно храниться все три матрицы коэффициентов.

Для эффективного использования канала DMA вычисления на CP2 необходимо выполнять с двойной буферизацией: в одной половине памяти CP2 в цикле обрабатывается набор («страница») блоков, в то время как из другой половины выгружается страница результатов и загружается следующая страница входных данных. Этот приём позволяет избежать простоев CP2 и DMA и является стандартным при разработке оптимизированных процедур под гибридные системы [1, 3].

Можно подсчитать, что в ядре на обработку 4 блоков параллельно в 4 секциях будет приходиться порядка 48 тактов CP2, при этом больше тактов потребуется на загрузки и выгрузки из регистров, чем на само умножение. Отсюда, на существующих частотах процессорного ядра и памяти BM9 время обработки набора блоков на CP2 будет меньше, чем время пересылки набора входных и выходных данных на CP2 и обратно, т.е. производительность процедуры деквантования будет определяться пропускной способностью канала DMA. Эта ситуация имеет место на большинстве процедур линейной алгебры из БЦОС.

Существенно сократить накладные расходы на пересылки по DMA можно, если принять во внимание, что в алгоритме декодирования JPEG к каждому блоку сразу после деквантования применяется операция обратного ДКП, независимо от других блоков. Две эти процедуры могут быть объединены в одну «композитную» функцию: к странице блоков, загруженной в локальную память, применяется деквантование, затем обратное ДКП. Это позволит сэкономить две пересылки всего объёма данных по DMA. Можно также сэкономить инструкции в самом ядре на CP2, домножая входные элементы на соответствующие коэффициенты непосредственно перед ДКП, без промежуточной выгрузки из регистров. Отметим, что похожая оптимизация сделана и в библиотеке libjpeg-turbo, как показано ниже.

3.3. Обратное ДКП

Процедура прямого/обратного ДКП является наиболее вычислительно нагруженной в кодировании/декодировании JPEG. Стандарт не предписывает использование конкретного алгоритма ДКП. К настоящему моменту разработано множество алгоритмов быстрого вычисления — в частности, конкретно для двумерного ДКП порядка 8×8 в составе процедур кодирования/декодирования JPEG.

В работе [12], где вводится понятие самого ДКП, предлагается и алгоритм для его вычисления через БПФ: ДКП длины N сводится к БПФ длины $2N$ с рядом вспомогательных операций. Таким образом, ДКП может быть вычислено за $O(NlogN)$ операций.

В [18] показано, что ДКП длины N можно свести и к БПФ длины N, при этом общее количество операций уменьшается примерно в 2 раза по сравнению с предыдущим алгоритмом. Выводится также алгоритм вычисления двумерного ДКП, непосредственно, а не как набора одномерных по столбцам и строкам — он обеспечивает дополнительную экономию в количестве операций.

Говоря о количестве операций, необходимо учитывать, что это количество включает как операции сложения, так и умножения. Алгоритмы на основе БПФ задействуют комплексную арифметику и связаны с выполнением большого количество операций умножения. Вместе с тем, на традиционных архитектурах CPU инструкция умножения является более «дорогой», чем инструкция сложения, в смысле количества выполняемых инструкций за такт процессора. В [19] был предложен алгоритм вычисления ДКП, не использующий комплексную арифметику и требующий меньшего количества операций умножения.

В дальнейшем было разработано большое количество алгоритмов, позволяющих сэкономить на операциях умножения. Одним из наиболее эффективных и часто используемых в настоящее время является алгоритм LLM [20] (названный так по первым буквам фамилий его авторов), который вычисляет прямое ДКП длины 8 за 11 операций умножения и 29 операций сложения. Здесь количество операций умножения совпадает с теоретической нижней оценкой для ДКП ([21]).

Другой популярный алгоритм — AAN [22], описанный также, например, в [23]. Этот алгоритм использует больше умножений (13), но 8 из них выполняются на начальном этапе, над входными элементами, и могут быть объединены с предыдущей операцией деквантования — благодаря этому деквантование совместно с обратным ДКП в сумме требует меньше операций умножения, чем при использовании алгоритма LLM.

Алгоритм AAN реализован в libjpeg-turbo для вычисления ДКП с использованием вещественной арифметики. Данная библиотека предоставляет также две реализации ДКП с использованием целочисленной арифметики: одна из них реализует модифицированную версию алгоритма LLM, другая, более эффективная, но менее точная — версию AAN.

Рассмотрим теперь варианты оптимизированной реализации на CP2. Поскольку на CP2 отсутствуют команды целочисленного умножения, необходимые при вычислении ДКП по любому из алгоритмов, имеет смысл рассматривать только вычисления с вещественными числами. Архитектура CP2 позволяет на каждом такте выполнять команды сложения и умножения комплексных чисел либо пар вещественных чисел. Кроме того, имеется команда «бабочка Фурье», которая также может выполняться на каждом такте. Поэтому, вообще говоря, с точки зрения CP2 нет причин экономить команды умножения или избегать комплексной арифметики.

Возможная реализация на CP2 была проанализирована для двух алгоритмов: алгоритма на основе БПФ из [18] и алгоритма AAN, который оперирует вещественными числами. В обоих случаях рассматривалось двумерное обратное ДКП порядка 8×8. Преобразование не зависит от того, к какому из каналов относится блок, и применяется в цикле ко всем блокам одной страницы, параллельно в четырёх секциях. Рассмотрим подробно вычисление обратного ДКП для одного блока в одной секции по алгоритму AAN.

Алгоритм AAN вычисляет двумерное ДКП, прямое или обратное, как набор одномерных по строкам и столбцам, т.е. в общей сложности 16 одномерных ДКП длины 8. Схема вычисления одномерного обратного ДКП длины 8 представлена на Рис. 4.

На данной схеме можно выделить 5 типов операций, которые могут быть выполнены за одну арифметическую команду CP2 (Рис. 5):

1) Сложение — вычисление по формуле $O = I_0 + I_1$.

Команда `psadd` выполняет сложение. Важно, что эта команда, как и все последующие, работает с парами вещественных чисел в регистрах. Поскольку обратное ДКП требуется вычислять единообразно для наборов векторов (8 столбцов и затем 8 строк), вектора можно обрабатывать парами.

Команда сложения используется в алгоритме 1 раз.

2) Вычитание — вычисление по формуле $O = I_0 - I_1$. Вычитание выполняется командой `pssub`. Таких команд в алгоритме 4.

3) Одновременное вычисление суммы и разности одних и тех же пар чисел. Для этой операции на CP2 есть специальная команда `psaddsub`. Всего в алгоритме требуется 11 таких команд.

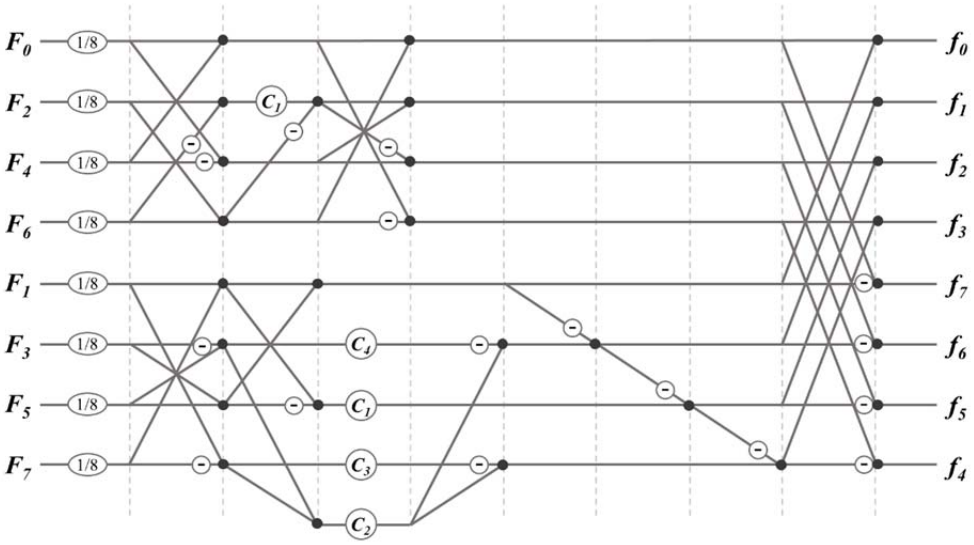

Рис. 4. Вычисление обратного ДКП длины 8 по алгоритму AAN

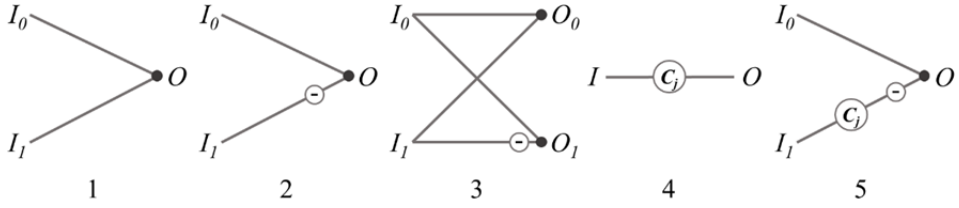

Рис. 5. Виды операций в алгоритме AAN

4) Умножение на коэффициент: `O = cjI`. Умножить пару чисел на один и тот же коэффициент можно, например, с помощью команды `psmul` (коэффициент должен располагаться и в старшем, и в младшем полуслове соответствующего регистра). В алгоритме используется 3 таких команды, не считая 8 умножений на 1/8, которые объединяются с деквантованием и не учитываются.

5) Умножение на коэффициент с вычитанием: $O = I_0 - c_j I_1$. Оно выполняется командой `psmsub`. Таких команд потребуется две.

Всего в операциях 4, 5 используется 4 коэффициента, которые могут постоянно храниться в регистрах СР2: $c_0 = 1{,}414213562$, $c_1 = 1{,}847759065$, $c_2 = 1{,}082392200$, $c_3 = 2{,}613125930$.

Общее количество арифметических команд на вычисление одномерного обратного ДКП пары векторов составляет 4 + 1 + 11 + 3 + 2 = 21. Всего на ДКП всех столбцов и строк потребуется $21 \cdot 4 \cdot 2 = 108$ команд.

Подсчитаем теперь команды на конвейере обменов с регистрами. Двумерное ДКП подразделяется на 2 этапа: одномерное ДКП по столбцам и по строкам. На каждом этапе требу ется в сумме одна загрузка 64 входных значений в регистры и одна выгрузка в локальную память — они требуют по 16 команд `ld/sd`. Отметим, что на схеме на Рис. 4 входные и выходные элементы располагаются в специальной последовательности — её нужно учитывать при загрузке и выгрузке элементов из регистров. Кроме того, на втором этапе после чтения из локальной памяти в одном регистре будут находиться соседние элементы одной строки. Чтобы обрабатывать строки парами, нужно, чтобы в одном регистре находились элементы соседних строк с одинаковыми индексами — для этого потребуется перегруппировка с использованием команды `psprmsgn`. На каждую пару векторов длины 8 нужно 4 таких команды, всего — 16. Перед выгрузкой результатов в системную память потребуется обратная перегруппировка.

Итого, общее количество команд обменов с регистрами составляет $4 \cdot 16 + 2 \cdot 16 = 84$. Эти команды могут объединяться с арифметическими в VLIW-инструкциях, и при условии конвейеризации все они могут быть скрыты. Таким образом, минимальное необходимое количество инструкций на весь алгоритм AAN — 108.

Аналогичный анализ, проведённый для алгоритма из [18], показал, что несмотря на специализированные возможности для БПФ, вычисление двумерного обратного ДКП одного блока по этому алгоритму потребует по меньшей мере 296 тактов СР2. К преимуществам алгоритма AAN можно отнести также бóльшую локальность: одновременно происходят вычисления только с парой векторов длины 8, соответственно, не требуется держать в регистрах все 64 элемента блока — это даёт дополнительную свободу для конвейеризации без накладных расходов на загрузку и выгрузку промежуточных данных. Кроме того, меньше количество дополнительных коэффициентов, которые используются в ходе вычислений.

В дальнейшем планируется реализация на СР2 алгоритма AAN.

3.4. Интерполяция

Интерполяция с применением того или иного фильтра, в том числе двух рассмотренных в разделе 2.5, представляет собой вычисление линейных комбинаций или копирование соседних значений блоков. Блоки размера 8×8 обрабатываются независимо друг от друга, при этом значение каждого пикселя может быть вычислено с использованием нескольких арифметических команд и небольшого количества регистров. Такого рода вычисления могут быть эффективно конвейеризованы и распараллелены на СР2.

Процедуры интерполяции различаются в зависимости от выбранного фильтра и схемы прореживания, использованной при сжатии, — для каждого случая потребуется разработка отдельной процедуры на СР2.

Чтобы сэкономить на пересылках по DMA, интерполяцию можно объединить в одну процедуру с предшествующим деквантованием и обратным ДКП. При определении размера страницы, т.е. количества блоков, обрабатываемых на СР2 за один приём, необходимо будет учесть, что после интерполяции количество блоков 8×8 в памяти СР2 увеличится.

3.5. Преобразование в цветовое пространство RGB

Преобразование значений из пространства YCbCr в RGB представляет собой вычисление трёх линейных комбинаций, независимо для различных троек значений. Эта процедура также может быть реализована на СР2 с высокой эффективностью.

Чтобы объединить это преобразование с предыдущими, необходимо организовать загрузку данных на СР2 так, чтобы все блоки, относящиеся к одному MCU, одновременно находились в одной секции. После вычисления деквантования, обратного ДКП, а также интерполяции для блоков Cb и Cr, в каждой секции окажутся тройки значений (Y, Cb, Cr), соответствующие отдельным пикселям — по ним и будут вычисляться новые тройки (R, G, B).

Итак, при декодировании JPEG вычисления на СР2 можно организовать так, что в сумме потребуется всего две пересылки данных: загрузка входных коэффициентов, после декодирования кода Хаффмана и повторов, и выгрузка выходных значений пикселей. Все этапы декодирования от деквантования до преобразования цветового пространства будут реализованы как последовательность вычислительных ядер на СР2, работающих с данными в локальной памяти, без обращения к DMA. Производительность такой процедуры будет определяться не пропускной способностью DMA, а производительностью разработанных ядер.

3.6. Формат входных и выходных данных

Выше при рассмотрении процедур на СР2 неявно подразумевалось, что все вычисления выполняются над 32-битными вещественными числами. Однако в процедуре декодирования JPEG коэффициенты на входе (после декодирования кода Хаффмана и повторов) и значения пикселей на выходе являются целыми. В библиотеке libjpeg-turbo целые числа на входе не менее чем 16-битные, а на выходе — 8-битные (какие именно типы будут использованы, зависит от конкретной архитектуры). Кроме того, в ходе вычислений дополнительно применяется округление: например, если числа на выходе ДКП не попадают в диапазон [0..255], они принудительно заменяются на 0 или 255. В libjpeg-

turbo все эти преобразования делаются путём применения маски и взятия значений из заранее заготовленной таблицы.

Что касается СР2, здесь поддерживается только 32-разрядная целочисленная арифметика и в весьма ограниченном объёме: в частности, отсутствует команда умножения, необходимая на всех стадиях декодирования JPEG. Иными словами, для вычисления на СР2 потребуется преобразование входных данных в 32-битный вещественный формат, а выходных — обратно в целые. На промежуточных этапах также может потребоваться преобразование форматов, чтобы осуществить округление.

Для описанных манипуляций могут быть использованы несколько команд СР2. Команда `pwtops` переводит пару 32-битных целых в пару 32-битных вещественных, а `pstopw` — обратно, с округлением. Команда `and` предназначена для побитового умножения — например, для наложения маски. Проверить, не выходит ли значение за заданный диапазон, можно при помощи команды `C.cond.fmt`. Кроме того, имеется команда `unpck16ws2ps`, которая преобразует четыре 16-битных целых числа в 32-битные вещественные. Для того чтобы собрать в непрерывную область памяти выходные 8-битные значения, можно воспользоваться командами `join8`, `join16` и `psprmsgn`.

Преобразование типов должно быть интегрировано в общую процедуру и выполняться после загрузки данных на СР2 и выполнения определённых этапов вычислений.

4. Заключение

Проведённое исследование продемонстрировало, что процессоры линейки КОМДИВ потенциально могут быть использованы для эффективного декодирования изображений в формате JPEG, а значит, могут успешно применяться в соответствующих приложениях.

Ключевой процедурой, которая требует оптимизации на СР2, является ДКП — планируется реализация этой процедуры с использованием алгоритма AAN. В дальнейшем планируется оптимизация также процедур деквантования, интерполяции и преобразования цветового пространства. Разработанные процедуры могут быть встроены в инфраструктуру библиотеки libjpeg-turbo.

Заключительным этапом исследования должно стать всестороннее тестирование оптимизированной библиотеки работы с форматом JPEG: сравнение производительности вычислений на КОМДИВ с использованием на СР2 и без, а также на других современных процессорах — в том числе на различных реальных задачах.

Литература

1. Сударева О.Ю. Эффективная реализация алгоритмов быстрого преобразования Фурье и свёртки на микропроцессоре КОМДИВ128-РИО. М.: НИИСИ РАН, 2014. 266 с.
2. Райко Г.О., Павловский Ю.А., Мельканович В.С. Технология программирования многопроцессорной обработки гидроакустических сигналов на вычислительных устройствах семейства «КОМДИВ». Гидроакустика. Вып. 20 (2). СПб.: ОАО «Концерн "Океанприбор"», 2014. 118 с.
3. Богданов П.Б., Сударева О.Ю. Применение отечественных специализированных процессоров семейства КОМДИВ в научных расчётах // Информационные технологии и вычислительные системы, 2016. №3. С. 45–65.
4. Богданов П.Б., Сударева О.Ю. Производительность процессоров КОМДИВ на ряде типовых расчётных задач // Информационные технологии и вычислительные системы, 2017. №4. С. 104–111.
5. International Standard ISO/IEC 10918-1:1993(E). CCITT Rec. T.81 (1992 E). Information technology — Digital compression and coding of continuous-tone still images — Requirements and Guidelines.
6. International Standard ISO/IEC 10918-5:2012(E). Rec. ITU-T T.871 (05/2011). Information technology — Digital compression and coding of continuous-tone still images — JPEG File Interchange Format (JFIF).
7. libjpeg-turbo — домашняя страница. Размещена по адресу: https://libjpeg-turbo.org/ (дата обращения 23 декабря 2018).
8. Зубковский П.С. Описание векторного сопроцессора процессора К64-М, версия 2.7. М.: НИИСИ РАН. 2013.
9. Wallace, G.K. The JPEG Still Image Compression Standard. 1992. IEEE Transactions on Consumer Electronics. 38(1):18–34. doi: 10.1109/30.125072.
10. Milleson, J. 2014. Partial Image Decoding On The GPU For Mobile Web Browsers. Master's Thesis. Chalmers University of Technology, University of Gothenburg. 53 p.
11. Huffman, D.A. 1952. A method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE. 40(9):1098–1101. doi: 10.1109/JRPROC.1952.273898.
12. Ahmed, N., T. Natarajan, and K.R. Rao. 1974. Discrete Cosine Transform. IEEE Transactions on Computers. C-23(1):90–93. doi: 10.1109/T-C.1974.223784.

13. Kerr, D.A. 2012. Chrominance Subsampling in Digital Images. Issue 3. Available at: http://dougkerr.net/Pumpkin/articles/Subsampling.pdf (accessed December 23, 2018).
14. Klein, S.T., and Y. Wiseman. 2003. Parallel Huffman Decoding with Applications to JPEG Files. The Computer Journal. 46(5):487–497.
15. Plumadore, K. 2018. GPU Parallel Huffman Decoding. U.S. Patent No. 9906239. Available at: http://www.freepatentsonline.com/9906239.html (accessed December 23, 2018).
16. Singh, S.P., A. Bhasin, and K. Saha. 2011. Parallelization of Variable Length Decoding. U.S. Patent No. 2011/0150351. Available at: https://patents.google.com/patent/US20110150351 (accessed December 23, 2018).
17. Chieppe, P. 2017. JPEG decoding using «end of block» markers to concurrently partition channels on a GPU. Australian National University, COMP4560. Available at: http://courses.cecs.anu.edu.au/courses/CSPROJECTS/17S1/Reports/Patrick_Chieppe_Report.pdf (accessed December 23, 2018).
18. Makhoul, J. 1980. A Fast Cosine Transform in One and Two Dimensions. IEEE Transactions on Acoustics, Speech and Signal Processing. 28(1):27–34. doi: 10.1109/12.895848.
19. Chen, W.-H., C. Smith, and S. Fralick. 1977. A Fast Computational Algorithm for the Discrete Cosine Transform. IEEE Transactions on Communications. 25(9):1004–1009. doi: 10.1109/TC.1982.1676108.
20. Loeffler, C., A. Ligtenberg, and G.S. Moschytz. 1989. Practical Fast 1-D DCT Algorithms with 11 Multiplications. Proceedings of the International Conference on Acoustics, Speech, and Signal Processing. 2:988–991. doi: 10.1109/ICASSP.1989.266596.
21. Duhamel, P., and H. H'Mida. 1987, New 2n DCT Algorithms Suitable for VLSI Implementation. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. 12:1805–1809. doi: 10.1109/ICASSP.1987.1169491.
22. Arai, Y., T. Agui, and M. Nakajima. 1987. A fast DCT-SQ scheme for images. Transactions of the IEICE. E-71(11):1095–1097.
23. Popović, M., and T. Stojić. 1998. The Fast Computation of DCT in JPEG Algorithm. 9th European Signal Processing Conference (EUSIPCO 1998). 1–4.

Богданов Павел Борисович. Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", г. Москва, Россия. Младший научный сотрудник. Количество печатных работ: 20. Область научных интересов: высокопроизводительные вычисления, OpenCL, SpMV, низкоуровневая оптимизация. E-mail: pbogdanov@list.ru

Сударева Ольга Юрьевна. Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", г. Москва, Россия. Младший научный сотрудник. Количество печатных работ: 7 (в т.ч. 1 монография). Область научных интересов: высокопроизводительные вычисления, GPGPU, OpenCL, цифровая обработка сигналов, БПФ, MG, SpMV, низкоуровневая оптимизация.
E-mail: olsudareva@gmail.com

JPEG image decoding on the KOMDIV microprocessors

P. B. Bogdanov, O. J. Sudareva

Federal Research Center “Institute of System Research” of Russian Academy of Sciences, Moscow, Russia

Abstract. In this paper we consider possible application of special-purpose massively-parallel SIMD-coprocessor CP2 for digital image compression. The CP2 coprocessor is designed by and used in several products of ISR RAS, and it is capable of computations with real and complex numbers. We take the JPEG digital image compression standard as an example and analyse the JPEG decompression algorithm for further implementation on CP2.

Keywords: KOMDIV, CP2, JPEG, JFIF, DCT, libjpeg-turbo.

DOI 10.14357/20718632190101

References

1. Sudareva, O.J. 2014. Effektivnaya realizatsiya algoritmov bystrogo preobrazovaniya Furje I svertki na microprocessore KOMDIV128-RIO [The effective implementation of the Fast Fourier Transform and convolution algorithms for the KOMDIV128-RIO microprocessor]. Moscow: ISR RAS. 266 p.

2. Rajko, G.O., J.A. Pavlovskij, and V.S. Melkanovich. 2014. Techologiya programmirovaniya mnogoprocessornoy obrabotki signalov na vychislitelnyh ustroistvah semeistva “KOMDIV” [Technology of programming of multiprocessor processing of hydroacoustic signals on “KOMDIV” computer set]. Gidroakustika [Hydroacoustics]. Issue 20(2). SPb.: The “Oceanpribor” concern. 118 p.
3. Bogdanov, P.B., and O.J. Sudareva. 2016. Primenenie otechestvennyh specializirovannyh processorov semeystva KOMDIV v nauchnyh raschetah [The applicability of Russian special-purpose KOMDIV microprocessor series for scientific computations]. Informatsionnye technologii i vychislitelnye sistemy [Information Technologies and Computational Systems]. 3:45–65.
4. Bogdanov, P.B., and O.J. Sudareva. 2017. Proizvoditelnost processorov KOMDIV na ryade tipovyh raschetnyh zadach [The KOMDIV microprocessors performance on a number of typical computational problems]. Informatsionnye technologii i vychislitelnye sistemy [Information Technologies and Computational Systems]. 4:104–111.
5. International Standard ISO/IEC 10918-1:1993(E). CCITT Rec. T.81 (1992 E). Information technology — Digital compression and coding of continuous-tone still images — Requirements and Guidelines.
6. International Standard ISO/IEC 10918-5:2012(E). Rec. ITU-T T.871 (05/2011). Information technology — Digital compression and coding of continuous-tone still images — JPEG File Interchange Format (JFIF).
7. libjpeg-turbo — homepage. Available at: https://libjpeg-turbo.org/ (accessed December 23, 2018).
8. Zubkovsky, P.S. 2013. Opisanie vektornogo soprocessora processora K64-M, versiya 2.7 [The specification of vector coprocessor of the K64-M procession, version 2.7]. M.: ISR RAS.
9. Wallace, G.K. The JPEG Still Image Compression Standard. 1992. IEEE Transactions on Consumer Electronics. 38(1):18–34. doi: 10.1109/30.125072.
10. Milleson, J. 2014. Partial Image Decoding On The GPU For Mobile Web Browsers. Master’s Thesis. Chalmers University of Technology, University of Gothenburg. 53 p.
11. Huffman, D.A. 1952. A method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE. 40(9):1098–1101. doi: 10.1109/JRPROC.1952.273898.
12. Ahmed, N., T. Natarajan, and K.R. Rao. 1974. Discrete Cosine Transform. IEEE Transactions on Computers. C-23(1):90–93. doi: 10.1109/T-C.1974.223784.
13. Kerr, D.A. 2012. Chrominance Subsampling in Digital Images. Issue 3. Available at: http://dougkerr.net/Pumpkin/articles/Subsampling.pdf (accessed December 23, 2018).
14. Klein, S.T., and Y. Wiseman. 2003. Parallel Huffman Decoding with Applications to JPEG Files. The Computer Journal. 46(5):487–497.
15. Plumadore, K. 2018. GPU Parallel Huffman Decoding. U.S. Patent No. 9906239. Available at: http://www.freepatentsonline.com/9906239.html (accessed December 23, 2018).
16. Singh, S.P., A. Bhasin, and K. Saha. 2011. Parallelization of Variable Length Decoding. U.S. Patent No. 2011/0150351. Available at: https://patents.google.com/patent/US20110150351 (accessed December 23, 2018).
17. Chieppe, P. 2017. JPEG decoding using «end of block» markers to concurrently partition channels on a GPU. Australian National University, COMP4560. Available at: http://courses.cecs.anu.edu.au/courses/CSPROJECTS/17S1/Reports/Patrick_Chieppe_Report.pdf (accessed December 23, 2018).
18. Makhoul, J. 1980. A Fast Cosine Transform in One and Two Dimensions. IEEE Transactions on Acoustics, Speech and Signal Processing. 28(1):27–34. doi: 10.1109/12.895848.
19. Chen, W.-H., C. Smith, and S. Fralick. 1977. A Fast Computational Algorithm for the Discrete Cosine Transform. IEEE Transactions on Communications. 25(9):1004–1009. doi: 10.1109/TC.1982.1676108.
20. Loeffler, C., A. Ligtenberg, and G.S. Moschytz. 1989. Practical Fast 1-D DCT Algorithms with 11 Multiplications. Proceedings of the International Conference on Acoustics, Speech, and Signal Processing. 2:988–991. doi: 10.1109/ICASSP.1989.266596.
21. Duhamel, P., and H. H'Mida. 1987, New 2n DCT Algorithms Suitable for VLSI Implementation. Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing. 12:1805–1809. doi: 10.1109/ICASSP.1987.1169491.
22. Arai, Y., T. Agui, and M. Nakajima. 1987. A fast DCT-SQ scheme for images. Transactions of the IEICE. E-71(11):1095–1097.
23. Popović, M., and T. Stojić. 1998. The Fast Computation of DCT in JPEG Algorithm. 9th European Signal Processing Conference (EUSIPCO 1998). 1–4.

Bogdanov P.B. Federal Research Center “Institute of System Research” of Russian Academy of Sciences, 36 building 1 Nakhimovsky prosp., Moscow, Russia, e-mail: pbogdanov@list.ru

Sudareva O.J. Federal Research Center “Institute of System Research” of Russian Academy of Sciences, 36 building 1 Nakhimovsky prosp. Moscow, Russia, e-mail: olsudareva@gmail.com