*Вестник Сыктывкарского университета.*
*Сер.1.Вып.6.2008*

**УДК 519.677**

# Параллельное сжатие больших изображений

*А.В. Симаков*

В этой статье описан эффективный метод для параллельного сжатия больших изображений. Алгоритм сжатия базируется на вейвлетном преобразовании. Основной акцент в работе делается на исследование практических аспектов процесса сжатия. Описанный алгоритм реализован в виде программы на языке C, исходный текст которой свободно доступен для загрузки через Интернет.

## 1. Введение

В настоящее время вейвлетные методы находят широчайшее применение в самых разнообразных прикладных задачах. Одной из таких задач является сжатие изображений. К преимуществам вейвлетных методов можно отнести высокую степень сжатия, простой и изящный способ прогрессивной передачи, возможность заранее указывать размер архива с точностью до байта и многое другое. Однако одного лишь вейвлетного преобразования недостаточно для реализации полноценного программного комплекса для сжатия изображений. Помимо вейвлетного преобразования процесс сжатия включает в себя целый ряд дополнительных операций, без которых программа превратится всего лишь в «демонстрационный стенд». В этой же работе, напротив, акцент делается на всестороннее описание процесса сжатия, обсуждаются возможные трудности реализации и пути их разрешения.

Идеи, изложенные в этой работе, реализованы в виде программы EPSILON, исходный текст которой доступен для загрузки через Интернет[1]. Программа имеет простую и понятную модульную структуру,

[1]Исходный текст программы EPSILON: *http://epsilon-project.sourceforge.net*

поддерживает более 30 различных вейвлетных преобразований и имеет удобный механизм для добавления новых преобразований. Проект EPSILON нацелен на разработку открытой системы для параллельного и отказоустойчивого сжатия больших изображений. Программа распространяется на условиях открытой лицензии GNU GPL, таким образом каждый желающий может изучать и улучшать ее код.

## 2. Описание метода

**2.1. Исходное изображение** Будем считать, что на вход поступает изображение $I$ размера $W \times H$ пикселов, где $W$ – это ширина, а $H$ – высота изображения. Каждый пиксел изображения представлен тройкой чисел $RGB$, которые определяют интенсивность красного, зеленого и синего цвета соответственно. Допустимый диапазон значений равен [0..255]. Поскольку изображения в градациях серого являются частным случаем полноцветных изображений, рассматривать их отдельно мы не будем.

На выходе мы хотим получить сжатое представление изображения, имеющее заранее заданный размер и обладающее заданными характеристиками.

**2.2. Разбиение изображения на блоки** Поскольку изображение может оказаться очень большим, обрабатывать его необходимо поблочно. Размер и форма блока в значительной степени влияют на производительность и эффективность работы всей системы. Так увеличение размера блока положительно сказывается на соотношении размер-качество, уменьшает количество служебных данных, выводимых программой с каждым блоком, и уменьшает количество «швов» в изображении, заметных при очень сильном сжатии. Однако эта тенденция довольно быстро сходит на нет. Кроме того, с увеличением размера блока растет и время, необходимое на его обработку. С другой стороны, маленькие блоки быстрее обрабатываются и при более мелком разбиении повышается устойчивость закодированных данных к повреждениям, но, вместе с тем, растут и накладные расходы.

Форма блока также важна. Квадратные блоки, размер которых является степенью двойки, наиболее удобны для практической реализации. Особенно это ощущается в вейвлетном преобразовании и алгоритме кодирования коэффициентов. Программа EPSILON по умолчанию разбивает изображение на блоки $256 \times 256$ точек, в тоже время отставляя за пользователем возможность переопределения этого параметра. Данное значение выбрано экспериментально и в большинстве случаев дает хорошие результаты.

Обозначим размер блока символом $B$. Если ширина изображения $W$ либо высота изображения $H$ не кратны $B$, то изображение отражается относительно своих границ до необходимого размера. При отражении граничные значения дублируются. При этом отражение вначале производится по горизонтали, а затем по вертикали.

**2.3. Преобразование цветовых пространств** Наиболее употребительным цветовым пространством для полноцветных изображений является пространство $RGB$, в котором тремя параметрами являются интенсивности красного, зеленого и синего в данном цвете. При отображении на компьютере значения этих трех компонент чаще всего выбираются в интервале от 0 до 255 (то есть занимают 8 бит).

На практике, однако, пространство $RGB$ не очень удобно для сжатия изображений. Поскольку человеческий глаз очень чувствителен даже к малым изменениям яркости, удобно иметь цветовое пространство, в котором яркость, или иначе светимость ($Y$), является одним из трех компонентов. Простейший способ такого построения — это вычесть компоненту $Y$ из красной и синей компонент $RGB$ и использовать новое цветовое пространство $Y$, $Cb = B - Y$, $Cr = R - Y$. Последние две компоненты называются хроматическими (от греческого chroma — цвет, краска). Они выражают цвет в терминах присутствия или отсутствия синего ($Cb$) и красного ($Cr$) при данном значении светимости.

Основываясь на результатах многочисленных экспериментов, светимость определяется как взвешенная сумма красного, зеленого и синего с весами 77/256, 150/256 и 29/256, соответственно.

Цветовое пространство $YCbCr$ было разработано как часть рекомендации ITU-R BT.601 при выработке международного стандарта для цифрового видео. Компонента $Y$ имеет пределы от 16 до 235, а $Cb$ и $Cr$ изменяются от 16 до 240, причем 128 соответствует нулевым значениям параметров $R$, $G$, $B$.

Связь между пространством $RGB$ в интервале [16..235] и пространством $YCbCr$ устанавливается в виде простых линейных соотношений. Это преобразование пространств можно записать в следующем виде:

$$\begin{aligned} Y &= (77/256)R + (150/256)G + (29/256)B, \\ Cb &= -(44/256)R - (87/256)G + (131/256)B + 128, \\ Cr &= (131/256)R - (110/256)G + (21/256)B + 128. \end{aligned}$$

Обратное преобразование равно:

$$R = Y + 1.371(Cr - 128),$$
$$G = Y - 0.698(Cr - 128) - 0.336(Cb - 128),$$
$$B = Y + 1.732(Cb - 128).$$

Если перейти из пространства $YCbCr$ в пространство $RGB$, то значения компонентов будут лежать в интервале [16..235]. Из-за ошибок округления и искажений при сильном сжатии, значения компонентов могут попасть в области [0..15] и [236..255].

После преобразования цветового пространства все три компоненты сжимаются независимо друг от друга. Причем хроматические компоненты $Cb$ и $Cr$ можно сжимать с гораздо бóльшими потерями, чем компоненту $Y$. Так, на кодирование компоненты $Y$ программа EPSILON по умолчанию выделяет 90 % общего битового бюджета, в то время как на компоненты $Cb$ и $Cr$ выделяется всего по 5 %. Эти цифры получены экспериментальным путем и в большинстве случаев дают хорошие результаты. В случае необходимости, программа EPSILON позволяет пользователю самостоятельно распределить битовый бюджет по компонентам изображения.

**2.4. Субвыборка в цветовых каналах** Как уже было сказано, человеческий глаз очень чувствителен даже к малым изменениям яркости, в то время как чувствительность к хроматическим составляющим значительно слабее. Этот факт позволяет нам сжимать хроматические компоненты значительно сильнее яркостной не опасаясь при этом, что глаз заметит искажения. Во многих стандартах для сжатия цветных изображений и видео, цветовые каналы дополнительно прореживаются по вертикали и горизонтали в соответствии с различными схемами. Например, алгоритм JPEG из цветовых каналов $Cb$ и $Cr$ выбирает лишь каждый второй пиксел по горизонтали и вертикали. Таким образом, размер каждой из цветовых компонент уменьшается в четыре раза. При декодировании, каждый пиксел заменяется на четыре пиксела с такими же значениями.

Итак, субвыборка позволяет значительно сократить количество обрабатываемых данных, повышая тем самым общую степень сжатия и скорость работы программы. Стоит отметить, что даже такой «грубый» прием в большинстве случаев остается незаметным для человеческого восприятия.

Программа EPSILON может работать как с прореживанием цветовых каналов, так и без него — выбор остается за пользователем.

**2.5. Сдвиг уровня постоянного сигнала** Ясно, что чем больше по абсолютной величине будут исходные значения, тем больше получатся коэффициенты вейвлетного разложения. На этапе кодирования величина коэффициентов становится серьезным фактором: чем больше коэффициент, тем больше битов потребуется для его адекватной передачи.

Абсолютную величину исходных значений (пикселов) можно значительно уменьшить при помощи очень простого приема: сдвига уровня постоянного сигнала (DC level shift). Эта операция производится для каждого из каналов $Y$, $Cb$, $Cr$ следующим образом:

1. Вычислить среднее значение пикселов
2. Вычесть среднее значение из каждого пиксела
3. Передать среднее вместе с закодированными данными

Очевидно, что если исходные данные лежали в диапазоне $0 \ldots 255$, то теперь диапазон значений будет не шире чем $-128 \ldots 127$.

**2.6. Уменьшение граничных искажений** Как уже было упомянуто, исходное изображение обрабатывается квадратными блоками, размер которых равен $2^N$. Такой подход очень удобен, однако имеет один недостаток: при очень высоких степенях сжатия становятся заметными искажения на стыках блоков.

В своей работе [3] авторы предлагают весьма оригинальный и эффективный способ для уменьшения искажений на границах блоков. Название статьи — «Boundary artifact reduction using odd tile lengths and the low pass first convention (OTLPF)» — фактически определяет суть метода. Авторы предлагают брать блоки размер которых не четен (Odd Tile Length) и первый коэффициент разложения — низкочастотный (Low Pass First). Поскольку низкочастотные и высокочастотные коэффициенты чередуются, а общее их количество нечетно, то и последний коэффициент разложения также будет низкочастотным. Отметим, что если размер блока взять равным $2^N + 1$ то требуемые условия будут выполнены на любом уровне вейвлетного разложения.

В своей статье авторы убедительно доказывают, а затем и подтверждают экспериментально, что если по краям находятся низкочастотные коэффициенты то граничных искажений становится значительно меньше. Единственный недостаток заключается в том, что данный метод применим лишь с биортогональными фильтрами.

Программа EPSILON поддерживает два режима работы: обычный (размер блока равен $2^N$) и OTLPF (размер блока равен $2^N + 1$). Обычный режим можно использовать как с ортогональными так и с биортогональными фильтрами. Режим OTLPF будет работать только с биортогональными фильтрами.

Результаты собственных экспериментов с программой EPSILON подтверждают выводы авторов статьи. При очень высоких степенях сжатия применение метода OTLPF действительно уменьшает искажения на стыках блоков: об этом свидетельствует и визуальная оценка реконструированного изображения и показатель PSNR.

**2.7. Вейвлетное преобразование и банки фильтров** Центральную роль в описываемом алгоритме сжатия изображений играет вейвлетное проеобразование. Задача вейвлетного преобразования заключается в выделении низко– и высокочастостотных составляющих сигнала. Уже давно установлено, что низкочастотная составляющая более ценна для человеческого восприятия нежели высокочастотная. Таким образом, условно разделив информацию на «более важную» и «менее важную» можно закодировать первую с меньшими потерями чем вторую. В основном засчет этого и достигается сжатие.

Реализуется вейвлетное преобразование путем применения к сигналу каскада из низко– и высокочастотных фильтров. Первые выделяют низкочастотную и подавляют высокочастотную информацию, а вторые — наоборот.

Вообще, фильтром называется линейный оператор, определяемый с помощью набора коэффициентов $h(k)$. Этот оператор применяется к входному вектору $x$, в результате чего получается выходной вектор $y$:

$$y(n) = \sum_k h(k)x(n-k) = h \star x.$$

Таким образом, фильтрация сводится к свертке сигнала с ядром фильтра. Заметим также, что пределы суммирования зависят от выбора последовательностей $x$ и $h$.

Программа EPSILON поддерживает два вида вейвлетных преобразований: ортогональные и биортогональные. В обоих случаях используются по две пары фильтров. Прямое вейвлетное преобразование осуществляется с помощью низко– и высокочастотного фильтра анализа, а обратное — с помощью низко– и высокочастотного фильтра синтеза. Обозначим эти фильтры как $A_{Low}$, $A_{High}$, $S_{Low}$ и $S_{High}$ соответственно. В совокупности они образуют, так называемый, банк фильтров.

Итак, вейвлетное преобразование $\hat{X}[1..N]$ исходного сигнала $X[1..N]$ вычисляется по следующей формуле:

$$\hat{X} = (X \star A_{Low}) \downarrow 2 \cup (X \star A_{High}) \downarrow 2.$$

Символом $\star$ обозначена операция свертки, а символом $\downarrow 2$ — неполная выборка. Неполная выборка отбрасывает каждый второй элемент массива. Таким образом, после фильтрации мы будем иметь два массива по $N/2$ элеметов в каждом. Первый массив будет содержать низкочастотные коэффициенты вейвлетного разложения, а второй — высокочастотные. Операция $\cup$ объединяет эти два массива в искомый $\hat{X}[1..N]$.

Восстановм теперь сигнал $X[1..N]$ из его вейвлетного преобразования $\hat{X}[1..N]$. Для этого разобъем $\hat{X}$ на две части: $\hat{X}_1 = \hat{X}[1..N/2]$ и $\hat{X}_2 = \hat{X}[N/2+1..N]$. Дальнейшие вычисления производятся по следующей формуле:

$$X = (\hat{X}_1 \uparrow 2) \star S_{Low} + (\hat{X}_2 \uparrow 2) \star S_{High}.$$

Символом $\star$ обозначена операция свертки, а символом $\uparrow 2$ — нуль-восполнение. Нуль-восполнение подставляет нуль на место отброшенных ранее коэффициетов. Символ плюс в последней формуле следует понимать как поэлементное сложение двух массивов.

Описанные выше формулы вычисляют одноуровневое вейвлетное преобразование. Для получения многоуровневого вейвлетного разложения алгоритм необходимо применять рекурсивно к низкочастотным коэффициентам, как если бы они представляли исходный сигнал.

Данный алгоритм легко приспособить для вычисления двумерного многоуровневого вейвлетного преобразования. Для этого достаточно последовательно применять одномерный алгоритм сначала ко всем строкам изображения, а затем ко всем столбцам.

Подробное изложение взаимосвязи между вейвлетным преобразованием и банками фильтров можно найти в книге [5].

**2.8. Кодирование коэффициентов** После того как изображение пройдет через вейвлетное преобразование, полученные коэффициенты округляются до ближайших целых и кодируются специальным методом. В программе EPSILON для этих целей применяется алгоритм SPECK — *Set Partitioned Embedded bloCK coder* [8].

Как ни странно это звучит, но основная идея SPECK заключается не в том, чтобы непосредственно сжимать изображение, а в том, чтобы переупорядочить биты коэффициентов его вейвлетного разложения специальным образом. Уже давно установлено, что для человеческого

восприятия низкочастотные компоненты изображения (т.е. плавные переходы яркости и цвета) несут гораздо больше информации, чем высокочастотные (резкие границы, углы, прямые линии). По сути, на принципе выделения низко— и высокочастотной информации, с последующим подавлением последней, и построены практически все технологии сжатия изображений с потерями. Не исключение и SPECK.

Используя особенности структуры вейвлетных коэффициентов, алгоритм SPECK переупорядочивает их биты. При этом, первые биты будут нести наиболее важную информацию, в то время как последние — лишь незначительные, уточняющие детали. Такое упорядочение данных часто называют прогрессивным. При прогрессивной передаче изображения, декодер, получая очередные порции закодированных данных, может последовательно улучшать и уточнять полученное им изображение. При этом вначале декодером будут прорисованы основные цветовые и яркостные переходы, а уж затем второстепенные контуры и малозаметные детали.

Прогрессивное упорядочение, помимо всего прочего, позволяет точно задавать требуемую степень сжатия. Действительно, можно сохранить лишь требуемое количество первых битов закодированного изображения, а оставшийся «хвост» просто отбросить, так как он несет сравнительно мало информации. Отсюда, собственно, и главная идея: выбрать наиболее важную информацию и передать ее в первую очередь.

Ясно, что бóльшие по абсолютной величине коэффициенты должны передаваться в первую очередь, так как содержат больше информации. Такой же подход спаведлив и для передачи отдельных битов самих коэффициентов. Так как старшие биты несут больше информации, их необходимо передавать первыми. Это, так называемый, метод битовых плоскостей (bit-plane), который часто используется для прогрессивной передачи данных.

Следует отметить, что вышеупомянутые принципы прогрессивного кодирования не являются характерными только лишь для алгоритма SPECK. Аналогичным образом устроены и такие алгоритмы как EZW [6] или SPIHT [7], появившиеся еще раньше чем SPECK. Разница между алгоритмами этого семейства заключается прежде всего в способе группировки и переупорядочивания коэффициентов.

Детальное описание алгоритма SPECK дается в работе [8]. Кроме того, эту работу сопровождает статья «An Example of SPECK Coding» в которой самым подробным образом разобран конкретный пример кодирования небольшого (8x8 точек) фрагмента изображения.

**2.9. Мультиплексирование каналов** Как уже было сказано, цветное изображение представляется в виде трех компонент $Y$, $Cb$ и $Cr$, причем каждая из них кодируется независимо друг от друга. Другими словами, к определенному моменту времени мы будем располагать тремя независимыми закодированными битовыми потоками.

Если изображение распаковывается и отображается локально, то вполне приемлемо сохранить все три потока по очереди, отдельно друг от друга. Если изображение будет передаваться по сети, то сохранять потоки по отдельности не целесообразно. В противном случае пользователь вначале увидит как прорисовывается черно-белый вариант изображения, а затем будет наблюдать как изображение наполняется цветом.

Для того чтобы изображение при передаче прорисовывалось равномерно по всем каналам, соответствующие битовые потоки необходимо равномерно перемешать (мультиплексировать) в один поток. Разумеется, перемешивать их следует таким образом, чтобы декодер смог затем восстановить исходные данные.

Не смотря на то, что мультиплексировать следует три потока, алгоритму достаточно уметь объединять лишь два: его можно применять рекурсивно, рассматривая только что объединенный поток как новый.

**2.10. Маркеры блоков** Для обеспечения хорошей масштабируемости системы, изображение обрабатывается по блокам. В закодированных данных блочная структура поддерживается при помощи специальных маркеров. Фактически, маркеры — это просто нулевые байты, отделяющие блоки данных друг от друга. Разумеется вместо нулевого значения для маркера можно выбрать любое другое, но в программе EPSILON в качестве маркера выбран именно нулевой байт.

Маркеры позволяют декодировать несколько блоков одновременно, перемешивать блоки в произвольном порядке, значительно повысить устойчивость закодированных данных к повреждениям, а так же многое другое.

Применение маркеров требует определенной осторожности: если в закодированных данных встретится нулевой байт, декодер по ошибке может принять его за конец блока. Для того чтобы исключить подобную двусмысленность нулевые байты не являющиеся маркерами подвергаются экранированию — то есть замене на, возможно более длинную комбинацию ненулевых байтов. Алгоритм экранирования при этом должен обеспечить однозначность восстановления данных. Таким образом, экранирование (byte stuffing) исключает из закодированных данных все нулевые байты, обеспечивая тем самым однозначную интерпретацию маркеров.

Алгоритм экранирования, использованный в программе EPSILON называется COBS — Consistent Overhead Byte Stuffing [4]. Этот алгоритм гарантирует, что даже в худшем случае расширение данных составит не более 0.4%, что близко к теоретическому пределу в 0.07%. Интересно отметить, что у большинства алгоритмов экранирования этот показатель составляет 200%, то есть в худшем случае каждый байт экранируется двумя байтами. Кроме того, COBS свободен от патентных ограничений, элегантен и крайне прост в реализации. Подробное описание, примеры, результаты испытаний и теоретический анализ читатель может найти в работе [4].

### 2.11. Двухпроходный алгоритм

Поскольку блоки изображения кодируются независимо друг от друга, встает вопрос о распределении битового бюджета между ними (не путать с распределением битового бюджета между тремя цветовыми каналами одного блока). В самом простом случае битовый бюджет делится поровну на все блоки. Эта стратегия битового аллоцирования называется CBR — Constant Bit-Rate.

Если изображение имеет примерно одинаковую структуру, то CBR работает вполне приемлемо. Однако, в большинстве ситуаций изображение неоднородно: в нем есть как фрагменты с плавными переходами, так и фрагменты со множеством мелких деталей. Для краткости будем называть их «простыми» и «сложными» соответственно. Отметим также, что чем больше изображение, тем выше шансы, что оно будет неоднородно.

Практика показывает, что в случае неоднородных изображений алгоритм CBR работает не очень эффективно. Дело в том, что для адекватной передачи «простого» блока требуется меньше битов чем для передачи «сложного». Более того, зачастую «простые» блоки даже не выбирают весь выделенный для них битовый бюджет. При высоких степенях сжатия с использованием стратегии CBR в изображении становятся заметными стыки блоков, поскольку степень искажения у «сложных» и «простых» блоков не одинакова.

Для того, чтобы повысить качество кодирования больших неоднородных изображений применяется алгоритм VBR — Variable Bit-Rate. Суть этой стратегии битового аллоцирования очень проста: сэкономить битовый бюджет на «простых» блоках и использовать его для кодирования «сложных».

Ключевой вопрос в реализации VBR — метрика для определения «сложности» блока. В качестве такой метрики можно взять длину блока в битах, закодированного «почти без потерь». Термин «сжатие по-

чти без потерь» (nearly lossless compression) означает, что кодирование производится с минимально-возможными для данного алгоритма потерями. К примеру, фрагмент ровного синего неба скорее всего закодируется всего в несколько байт, в то время как на кодирование фрагмента изрезанной береговой линии уйдет гораздо больше битов.

Таким образом, алгоритм VBR, реализованный в программе EPSILON, является двухпроходным. На первом проходе все блоки изображения кодируются «почти без потерь». В полученном на этом этапе изображении длина закодированных блоков пропорциональна их сложности. Далее программа рассчитывает во сколько раз размер закодированного «почти без потерь» изображения превышает лимит, установленный пользователем. Допустим лимит превышен в $R$ раз. После этого программа пробегает по закодированному изображению и усекает каждый его блок в $R$ раз. Таким образом, на выходе получится изображение требуемого размера, причем длина закодированных блоков осталась пропорциональной их сложности.

Эксперименты показали, что VBR существенно повышает качество изображения по сравнению с CBR. Однако VBR не лишен и недостатков: поскольку алгоритм делает два прохода по изображению и требует больше работы по кодированию данных, скорость кодирования примерно в два раза меньше по сравнению с CBR. В любом случае программа EPSILON оставляет за пользователем выбор стратегии битового аллоцирования: CBR или VBR.

**2.12. Формат файла** Закодированные данные в конечном итоге сохраняются в файле. Формат файлов, создаваемых программной EPSILON весьма необычен. Фактически, никакого формата файла — в привычном его понимании — вообще не существует. В сжатом файле EPSILON нет ни одного заголовка (например, ширины или высоты изображения) общего для всех блоков. Вся необходимая служебная информация сохраняется в заголовках каждого блока. Подобный подход несколько избыточен, поскольку одна и та же информация сохраняется много раз, зато делает сжатый файл крайне устойчивым к повреждениям, а каждый блок — совершенно автономным.

Для простоты, все поля заголовков сохраняются не в бинарном виде, в виде ASCII текста. Это улучшает и упрощает переносимость программы на различные аппаратные архитектуры и позволяет, при необходимости, редактировать заголовки практически в ручную. Последняя возможность полезна при аварийном восстановлении сильно поврежденных данных.

Еще большую устойчивость к повреждениям обеспечивают марке-

ры. Они локализуют возможные повреждения и позволяют декодеру выполнить синхронизацию. Таким образом, если какой либо блок поврежден, декодер может локализовать повреждение и перейти к следующему блоку.

**2.13. Удаленный просмотр изображения по сети** Благодаря блочной структуре данных наблюдатель может удаленно (то есть по сети) просматривать интересующие его фрагменты изображения. Размер самого изображения при этом может на порядки превосходить размер выбранного фрагмента: допустим, это может быть район города на большой спутниковой фотографии. При этом загружаться будут только те данные, которые действительно необходимы, что в значительной степени снижает требования к скорости канала передачи данных.

Кроме того, благодаря прогрессивному алгоритму кодирования изображение не будет загружаться сверху вниз строка за строкой, а будет отображаться сразу целиком, создавая тем самым эффект постепенного проявления, при котором размытое изображение постепенно становится все более и более четким. Этот процесс можно остановить в любой момент времени, когда будет достигнуто необходимое визуальное качество изображения. Опыты показывают, что даже ничтожно малой части — тысячной доли от размера оригинала — вполне достаточно, для того чтобы составить о нем впечатление.

## 3. Параллельная обработка

Программа EPSILON изначально проектировалась как параллельная система. Распараллеливание наиболее удобно на уровне блоков — то есть небольших квадратных фрагментов изображения. Поскольку блоки никак не связаны друг с другом, обрабатывать их можно параллельно.

В программе EPSILON реализовано два подхода к параллельной обработке. Первый подход основан на применении потоков (POSIX threads). Этот метод будет хорошо работать на многопроцессорных компьютерах, а также на компьютерах с многоядерными процессорами. Второй подход основан на применении MPI (Message Passing Interface) — технологии которая очень широко используется на высокопроизводительных кластерах.

**3.1. Потоки стандарта POSIX** Потоки — это широко распространенная технология параллельного программирования. Распараллеливание достигается за счет того, что операционная система старается выполнять каждый поток многопоточного приложения на отдельном

процессоре. Если процессоров меньше чем потоков, то им приходится делить вычислительные ресурсы между собой.

Реализаций потоков достаточно много. Программа EPSILON использует потоки стандарта POSIX (POSIX threads или Pthreads). Выбор в основном обусловлен тем, что поддержка Pthreads имеется практически в любой современной UNIX-системе.

Многопоточная версия программы EPSILON предназначена для многопроцессорных компьютеров, а также для компьютеров с многоядерными процессорами. Необходимое количество потоков указывается пользователем в качестве параметра программы.

**3.2. Message Passing Interface** MPI является наиболее известной и распространенной технологией параллельного программирования. Стандарт MPI имеет большое количество разнообразных реализаций, в том числе и несколько открытых (LAM, MPICH).

Принцип работы MPI-программы достаточно прост. Каждый процесс в системе имеет уникальный номер (rank). Этот номер однозначно определяет процесс и позволяет ему обмениваться сообщениями с другими процессами. Поскольку процессы в MPI-программе, как правило, запущены на разных компьютерах — появляется возможность выполнять вычисления параллельно.

## 4. Результаты испытаний

**4.1. Тестовые изображения** В испытаниях программы EPSILON использовались два цветных изображения: цифровая фотография острова Санторини и большая спутниковая фотография пустыни Намиб. Характеристики изображений приведены в следующей таблице:

| Изображение | Ширина | Высота | Бит на пиксел | Размер |
|---|---|---|---|---|
| `SANTORINI` | 3072 | 2304 | 24 | 20 Мб |
| `NAMIB_DESERT` | 7904 | 8512 | 24 | 200 Мб |

**4.2. Производительность многопоточной версии** Испытания многопоточной версии программы EPSILON проводились на компьютере с несколькими процессорами (CPU — Central Processing Unit) следующей конфигурации:

| Тип процессора | Intel Xeon CPU 3.2 GHz |
|---|---|
| Количество CPU | 2 |
| Количество ядер | 4 |
| Оперативная память | 2 Gb |
| Жесткий диск | SCSI 200 Gb (RAID 5) |
| Операционная система | Linux Mandriva 2007 |

**Описание теста:** Сжатие изображения `SANTORINI` в 10 раз, размер блока 256x256 пикселов, фильтр Добеши 9/7, режим обработки блоков OTLPF, прореживание цветовых каналов. Декодирование сжатого изображения.

| Кол-во потоков | Кодирование (сек.) | Декодирование (сек.) |
|---|---|---|
| 1 | 11.035 | 11.200 |
| 2 | 5.819 | 6.123 |
| 3 | 5.542 | 5.532 |
| 4 | 4.866 | 4.964 |
| 5 | 6.058 | 5.086 |
| 6 | 6.134 | 5.463 |

**Комментарий:** В обоих случаях при увеличении количества потоков с одного до двух получаем почти двухкратное ускорение программы. При увеличении количества потоков сначала до трех, а затем и до четырех также наблюдается определенный рост производительности. Видно, что когда количество потоков превышает количество реальных процессоров, рост производительности замедляется. При дальнейшем увеличении количества потоков отмечаем снижение производительности. Данный факт вполне предсказуем и объясняется тем, что количество потоков превышает количество ядер.

**Описание теста:** Сжатие изображения `NAMIB_DESERT` в 20 раз, размер блока 256x256 пикселов, фильтр Добеши 9/7, режим обработки блоков OTLPF, прореживание цветовых каналов. Декодирование сжатого изображения.

| Кол-во потоков | Кодирование (сек.) | Декодирование (сек.) |
|---|---|---|
| 1 | 75.450 | 83.155 |
| 2 | 39.761 | 44.028 |
| 3 | 36.230 | 39.529 |
| 4 | 32.046 | 36.108 |

**Комментарий:** Наблюдаем схожую динамику производительности, что и в предыдущем случае. При переходе на два потока получаем практически двухкратное ускорение. При дальнейшем увеличении количества потоков так же наблюдается определенный рост производительности. Снижение темпов ускорения программы в данном случае объясняется тем, что количество потоков превышает количество реальных процессоров. Тем не менее, видно что многоядерные процессоры дают существенный прирост производительности.

### 4.3. Производительность кластерной версии

Кластерная версия программы EPSILON тестировалась на двух кластерах: MicroFlops (собственная сборка) и ANT (Научно-Исследовательский Вычислительный Центр при МГУ им. Ломоносова).

Конфигурация кластера MicroFlops:

| | Узел 0 | Узел 1 | Узел 2 | Узел 3 |
|---|---|---|---|---|
| Тип CPU | Athlon 2Ghz | Xeon 3Ghz | Xeon 3Ghz | Xeon 3.2Ghz |
| Кол-во CPU | 1 | 1 | 1 | 2 |
| Кол-во ядер | 1 | 2 | 2 | 4 |
| RAM | 512Mb | 2Gb | 2Gb | 2Gb |
| HDD | 150Gb | 70Gb | 70Gb | 200Gb |
| ОС | MDV2007 | MDV2007 | MDV2007 | MDV2007 |

Конфигурация кластера ANT:

| | |
|---|---|
| Узлов | 80 |
| Ядер | 160 |
| Конфигурация узла | 2xOpteron 2.2 ГГц, 4 Gb RAM |
| Коммуникационная сеть | Infiniband |
| Транспортная сеть | Gigabit Ethernet |
| Сервисная сеть | FastEthernet |
| Пиковая производительность | 704 GFlops |

**Описание теста:** Сжатие изображения `SANTORINI` в 10 раз, размер блока 256x256 пикселов, фильтр Добеши 9/7, режим обработки блоков OTLPF, прореживание цветовых каналов. Декодирование сжатого изображения. Кластер MicroFlops.

| Кол-во процессов | Кодирование (сек.) | Декодирование (сек.) |
|---|---|---|
| 2 | 12.561 | 12.704 |
| 3 | 6.431 | 6.497 |
| 4 | 4.397 | 4.424 |
| 5 | 3.290 | 3.440 |
| 6 | 3.155 | 3.269 |
| 7 | 3.010 | 3.161 |
| 8 | 2.767 | 2.984 |
| 9 | 2.595 | 2.807 |

**Комментарий:** Видно, что даже на обычной офисной сети FastEthernet программа демонстрирует хорошую масштабируемость. Замедление темпов роста производительности наблюдается лишь начиная со случая CPU=6, когда количество процессов начинает превышать количество реальных процессоров. Тем не менее рост производительности все равно на лицо. Это еще раз подтверждает преимущества многоядерных процессоров.

**Описание теста:** Сжатие изображения `SANTORINI` в 10 раз, размер блока 256x256 пикселов, фильтр Добеши 9/7, режим обработки блоков OTLPF, прореживание цветовых каналов. Декодирование сжатого изображения. Кластер ANT.

| Кол-во процессов | Кодирование (сек.) | Декодирование (сек.) |
|---|---|---|
| 2 | 10.365 | 8.531 |
| 3 | 4.594 | 4.264 |
| 4 | 3.258 | 2.972 |
| 5 | 2.501 | 2.171 |
| 6 | 2.008 | 1.784 |
| 7 | 1.698 | 1.559 |
| 8 | 1.563 | 1.417 |
| 9 | 1.399 | 1.297 |
| 10 | 1.251 | 1.199 |
| 11 | 1.148 | 1.160 |
| 12 | 1.091 | 1.140 |

**Описание теста:** Сжатие изображения `NAMIB_DESERT` в 20 раз, размер блока 256x256 пикселов, фильтр Добеши 9/7, режим обработки блоков OTLPF, прореживание цветовых каналов. Декодирование сжатого изображения. Кластер ANT.

| Кол-во процессов | Кодирование (сек.) | Декодирование (сек.) |
|---|---|---|
| 2 | 52.694 | 52.087 |
| 3 | 28.115 | 26.959 |
| 4 | 21.729 | 18.598 |
| 5 | 18.222 | 14.343 |
| 6 | 15.616 | 12.099 |
| 7 | 14.661 | 11.665 |
| 8 | 12.998 | 11.018 |
| 9 | 11.794 | 9.648 |
| 10 | 11.745 | 9.287 |
| 11 | 10.393 | 9.217 |
| 12 | 10.353 | 8.963 |

**Комментарий:** На кластере ANT программа EPSILON также демонстрирует хорошую масштабируемость. Некоторое замедление темпов роста производительности при большом количестве процессов объясняется недостаточной пропускной способностью сети, поверх которой работает сетевая файловая система (NFS).

## 5. Заключение

В данной работе были изложены принципы построения эффективной системы для параллельного сжатия изображений. Результаты испытаний на различных компьютерах и кластерах показывают хорошую масштабируемость программы.

Программа EPSILON распространяется в виде исходных текстов на условиях открытой лицензии GNU GPL. Таким образом, каждый желающий может свободно загрузить и изучить ее код [2].

Помимо своего прямого назначения — сжатия данных — программа EPSILON может использоваться и как платформа для исследования разнообразных вейвлетных преобразований. На момент написания этих строк в программе EPSILON заложена поддержка порядка 30 различных фильтров. Более того, имеется очень простой (практически не требующий навыков программирования) механизм для добавления и изучения новых фильтров.

### Литература

1. Желудев В.А, Певный А.Б. *Вейвлетное преобразование Баттерворта и его реализация при помощи рекурсивных фильтров* // Ж. вычисл. мат. и матем. физ. 2002. Т. 42. N 4. С. 571–582.

[2] Исходный текст программы EPSILON: *http://epsilon-project.sourceforge.net*

2. Ватолин Д, Ратушняк А, Смирнов М, Юкин В. *Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео* // Диалог-МИФИ, 2003 г

3. Wei J, Pickering M.R, Frater M.R, Arnold J.F, Boman J, Zeng W. *Boundary artifact reduction using odd tile lengths and the low pass first convention (OTLPF)* // Proc. of SPIE, Appl. of Digital Image Proc., July, 2001.

4. Cheshire S, Baker M. *Consistent overhead byte stuffing* // IEEE/ACM Transactions on Networking, vol. 7, N. 2, 159–172, 1999.

5. Strang G, Nguyen T. *Wavelets and Filter Banks* // Wellesley-Cambridge Press, Boston, 1996.

6. Shapiro J. *Embedded image coding using zerotrees of wavelet coefficients* // IEEE Transactions on Signal Processing. 1993. V. 41. N 12. P. 3445–3462.

7. Said A, Pearlman W. *A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees* // IEEE Trans. on Circuits and Systems for Video Technology. 1996. V. 6. P. 243–250.

8. Asad Islam, Pearlman W. *An embedded and efficient low-complexity hierarchical image coder* // in Proc. of SPIE V. 3653, Visual Comm. and Image Processing 99, San Jose, CA, Jan 1999, p. 294–305

9. Huffman D.A. *A method for the construction of minimum-redundancy codes* // Proc. Inst. Radio Engineers. 1952. V. 40. N 9. P. 1098–1101.

10. Воеводин В.В., Воеводин Вл.В. *Параллельные вычисления* // СПб. БХВ-Петербург, 2002.

11. Симаков А.В. *Передача звука с адаптацией к пропускной способности канала* // Сыктывкарский государственный университет, январь 2005. *http://citforum.ru/nets/articles/agress/*

12. Симаков А.В. *Прогрессивная передача изображений через Интернет* // Нелинейные проблемы механики и физики деформируемого твердого тела. СПбГУ, 2004. Вып. 8. С. 147–161.

13. Симаков А.В. *Сжатие изображений при помощи вейвлетных преобразований* // Вестник молодых ученых. Сер. Прикладная математика и механика. 2004. Вып. 4. С. 53–62.

14. Симаков А.В. *Код Хаффмана* // Сыктывкарский государственный университет, октябрь 2002. *http://www.entropyware.info*

15. International Telecommunication Union (ITU). *Information technology – Digital compression and coding of continuous-tone still images: Requirements and guidelines* // CCITT Rec. T.81 (1992 E) | ISO/IEC 10918-1 : 1993(E)

16. Стивенс Р. *UNIX: разработка сетевых приложений* // Спб: Питер, 2003.

**Summary**

In this paper we describe an efficient algorithm for parallel compression of huge images. The algorithm is based on wavelet transform. The article is focused on practical implementation and evaluation of this compression method. As a result, all described features are implemented in software library written in C language. This library and compression program is freely available through the Internet and distributed with full source code under the terms of Open Source GNU GPL license. So, everyone can download, build and evaluate the program.

*Сыктывкарский государственный университет* *Поступила 1.11.2007*

**Содержание**



**Contents**