On Runlength-based Approaches for Achieving High-speed Compression of Map Images

Øyvind Ryan
Department of Informatics, University of Oslo, Norway
Email: oyvindry@ifi.uio.no

Abstract
Many low bit-depth images, like map images, are more compactly expressed in terms of runlength-representations. Some image formats can utilize such representations to speed up image compression. It will be shown that both image standards basing compression on context formation, like JPEG2000, and standards performing compression scanline by scanline, like PNG, can utilize runlength-based optimisations. GIF is used as an example where runlength-based optimisations already have been established.

1. Introduction
Many low bit-depth images, like maps, are more compactly expressed in terms of runlength-representations. The TIFF standard [4] is specified so that it can exploit such representations when compressing image data. In [10], it was shown that a simple runlength-based technique can be applied to speed up compression of images to GIF. As it turns out, other image standards can also exploit runlength-based optimisations when it comes to speeding up compression. The main discovery of this paper is that standards like PNG and JPEG2000 also can benefit from runlength-based optimisations. These constitute two of the most widely used standards today, with JPEG2000 being the emerging standard.

Runlength-represented image data can arise from various sources. The source in this paper will be a number of bi-level TIFF files. The Raster IMaging Framework for image processing, or RIM, is used in our experiments. RIM contains support for TIFF, and decompresses bi-level TIFF to an internal runlength-representation. RIM then merges the representations for all the bi-level files to a single (multi colour) runlength-representation. RIM also supports compression to different image formats, like GIF, while support for PNG and JPEG2000 has been implemented particularly for this paper. In addition, different implementations for each of these formats were written, and will be compared here: One suited for runlength-based processing, another suited for raster-based processing.

This paper is organized as follows: Section 2 formalizes the use of runlength-representations for images. Section 3 presents adaptations of runlength-based methods for speeding up compression with the PNG image format. Section 4 presents similar results for JPEG2000.

2. Runlength-representations of images
A runlength-representation of the part of a bi-level image with upper left coordinates \((x, y)\) and lower right coordinates \((x + \text{wdt}, y + \text{lth})\) can be as follows:
- Each line is represented as a strictly increasing sequence of numbers. The numbers are the points where background/foreground changes occur, measured relative to the left side \((x)\) of the image.
- A background/foreground change at \(\text{wdt}\) is added as a terminator for the runlength-representation.

To elaborate, if \(x = 0\), \(\text{wdt} = 100\) and the pixels between 20 and 80 is foreground with the rest being background, runlength-representation of the line would be \([20, 80, 100]\). It is seen that only 3 values are needed to represent 100 pixels in this case, making the runlength-representation superior in cases with few foreground/background changes.

A runlength-representation may work well only up to a certain level of image detail. The term used to quantify image detail in this paper is \(\text{accumulated runs per line, or ARL}\). For a layered image, this is measured by calculating the total number of runs (background AND foreground) for each layer separately, adding these numbers, and finally taking a line average. It is not taken into account that runs from one layer may shadow for runs in other layers. Consequently, ARL is not the same as the actual number of runs in the layered image, it is somewhat higher. Intuitively, a raster-based representation would work best at high ARL values. Map data normally have low ARL values.

The TIFF test images which will be used are tiled, with tile dimensions \(512 \times 512\). They show different parts of Norway, and each image has 19 layers. Runlength-based and raster-based processing are performed for each tile to generate enough tests and ARL values. Compressed file sizes and performance (in terms of clock cycles) will be measured. Focus in the performance measurements is on the time it takes to transcode the TIFF input layers to another output format. The term megacycle \((= 10^6\text{ clock cycles})\) is used to measure performance for a single image transcoding.
operation.

3. Runlength-based PNG compression

The compression algorithm most commonly used in PNG datastreams is deflate [5], which is an LZ77 derivative [12]. Deflate uses a search buffer of size up to 32768 bytes, where previous samples are stored for the purpose of matching the next part of the datastream. Long matches result in better compression, since fewer symbols are used to encode the datastream. In this respect, deflate is suited for runlength-based optimisations, since matching within a search buffer can be performed with less memory access when a compact representation in terms of runlengths is used; If the next data segment to encode starts with a run of a given colour, and one has a reference to a previous run of the same colour in the search buffer, matching pixel by pixel can be avoided. In this case, one knows that the minimum length of the two runs match, and fewer memory accesses are needed in the matching process.

Two ways of performing PNG encoding have been implemented for this paper. One uses raster-based matching, the other uses runlength-based matching, i.e. matching based on runlength-represented data as sketched in section 2. Measurements for these two approaches will be compared.

Raster-based matching. This is based on libpng [9], libpng’s default way of encoding is used, using a search buffer of 32768 pixels. Matching is performed pixel by pixel, resulting in much memory access. Three consecutive pixel values are hashed for quick access to matches in the search buffer starting with these three pixels [5]. The matches for the three pixels are chained together, with the last pixel combinations first in the hash chain. Using longer hash chains means bigger potential for matches, and thus higher compression. The implementation used here limits hash chain lengths to 4096, which is libpng’s own configuration when high compression is preferred over high performance. Long hash chains degrade performance due to extensive matching.

Runlength-based matching. Runlength-based matching as described here uses a smaller search buffer. Matching is performed runlength by runlength, leading to less matching operations. A runlength is chained with previous runs of the same colour, in the same way libpng chains combinations of 3 pixels. This reduces the need for memory, since a raster-based search buffer is not needed. Compression for runlength-based matching is degraded since a smaller search buffer is used.

Figure 1 compares the two PNG encoding strategies. It is seen that the difference in performance is substantial, while the difference in compressed file sizes is minimal. For runlength-based matching, the search buffer is chosen so that it always contains pixels from the previous line. This means that the search buffer size depends on the image width. Since there is a high correlation between the current and previous line, one is lead to believe that many long matches will be picked up if the search buffer includes the previous line. Figure 1 shows the cumulative distribution of distances for matches up to 8 lines back. When raster-based matching is used, jumps are seen with distances an image width apart. The first jump is largest, indicating highest correlation with the previous line. With runlength-based matching only the first jump can be recognized, since the search buffer only includes pixels from the previous line. Since so many matches occur at the current or previous line, it is reasonable to assume that the two strategies are close when it comes to compression. It is likely that runlength-based matching can achieve both better performance and compression than raster-based matching if a longer search buffer is used.

The deflate compression algorithm offers some functionality for improving compression further, for instance dynamic Huffman codes for adapting the Huffman codes used to the image statistics. This paper is primarily interested in real-time image processing, and dynamic Huffman tables represent the additional real-time overhead of calculating image statistics. Dynamic Huffman tables are thus not used with either runlength-based or raster-based matching in this paper. PNG can also do more to improve compression, for instance by additionally applying filters to each line. In many cases, this improves compression since inter-line dependencies are picked up in the compression process. Filtering is not used in this paper, since preliminary measurements showed that it had little impact on compression on the types of images used here.

4. Runlength-based JPEG2000 compression

The computation-intensive tasks of JPEG2000 are embedded block coding with optimised truncation (EBCOT) Tier-1, and the discrete wavelet transform (DWT). Lossless JPEG2000 compression will only be used here. The wavelet used by lossless JPEG2000 was designed with continuous-tone imagery in mind. The (optional) discrete wavelet transform will not be used here since our images are far from continuous-tone.

JPEG2000 [3] groups samples in blocks, and applies arithmetic coding to them. Block samples are traversed in many passes, so that they need to be buffered. The best one can do to reduce memory usage is to buffer only a horizontal stripe of blocks at a time when processing the samples.

The arithmetic coding JPEG2000 applies is context-based. There has been extensive research into context-based approaches for improving compression. One recent development is that of multilayer context tree modeling [7]. While such techniques are useful for improving compression, the
penalty can be high when it comes to performance. In real-time image processing applications, a tradeoff between performance and compression must be done. This can be achieved by fixing a particular template to be used for context formation, like JPEG2000 does. JBIG [1] and JBIG2 [2] also apply context-based coding, although the number of contexts differ from that of JPEG2000.

It is harder to adapt runlength-based methods to context-based coding, than was shown to be the case for PNG. A substantial part of the complexity in EBCOT Tier-1 lies in partitioning bit-planes into three different passes: Significance propagation, magnitude refinement and cleanup. For each sample, pass membership is decided, and a context is formed. Methods have been proposed for simplifying this process. [8] proposes Sample Skipping (SS), Group-Of-Column Skipping (GOCS) and pass skipping as speedup methods: By storing information in previous passes, the number of pass membership checks is reduced. [6] suggests a different method, where the number of passes is minimized by buffering contexts and pass memberships for all passes in a few scans.

Methods such as the ones mentioned analyse samples in real-time to detect repeating patterns of pass membership, and they do not take advantage of an already existing data representation, such as a runlength-representation. A JPEG2000 skipping method based on runlength-representations will be sketched, and its performance enhancements will be measured. The method is similar to GOCS in the sense that a group of columns can be skipped altogether in many passes.

In figure 2(a), part of a stripe of samples is shown in conjunction with the line above and below. All samples with black colour are equal, other samples have any other values. Let us call a part of a stripe like this a stripe-constant area. With a runlength-representation, stripe-constant areas are quickly detected in areas with little variation, as often is the case for map images. For stripe-constant areas, pass membership and contexts can be determined many at a time, thereby offering possibilities for skipping techniques.

To see this, the following can be deduced from the JPEG2000 standard for the samples in a stripe-constant area:

1) If the bit-plane to be scanned comes before the first bit-plane where the samples become significant, all samples, except possibly the first and last stripe column, are coded in run-mode in the cleanup pass,

2) in the first bit-plane where the samples are magnitude refinement coded, all samples will use the same context (which [11] calls $\kappa_{mag} = 16$),

3) in all lower bit-planes, all samples are magnitude refinement coded with the same context (which [11] calls $\kappa_{mag} = 17$).

A runlength-based method detecting stripe-constant areas and exploiting the points listed above was implemented. The extra overhead with this method comes from the need for comparing the values on six consecutive runlength-represented lines. For images where only small stripe-constant areas are detected (i.e. images with much detail), it is to expect that no performance enhancements are obtained.

The runlength-based approach will be compared with a raster-based approach. The raster-based approach is implemented using the state broadcasting technique described in [11] chapter 17.1.2, to perform context formation and pass membership detection. State broadcasting reduces the need...
Fig. 4. Performance for JPEG2000 with the runlength-based and raster-based approaches

for memory access, by for instance skipping certain stripe columns entirely in many passes. To simplify comparing the raster-based and the runlength-based approaches, the runlength-based approach will also use state broadcasting outside stripe-constant areas. Also, in stripe-constant areas, state broadcasting will be used outside the bit-planes listed above (where context formation and pass membership simplified).

Figure 4 shows the comparison between the raster-based and runlength-based approach. There is a slight improvement in performance only. It is fair to assume that this is due to stripe-constant area simplifications. Figure 3(b) shows the percentage of samples having stripe-constant area membership. Numbers for all test images are shown, plotted against ARL to see the connection with image detail. As expected, lower ARL values give more stripe-constant areas, and higher ARL values give a smaller performance advantage for the runlength-based approach. For the tests, palette-based JPEG2000 was used with blocks of size $32 \times 32$. Only 4 magnitude bit-planes are needed for all samples, due to the limited number of colours in the images. It is expected that the difference between the two compared approaches is higher when more bit-planes are involved.

Since image formats like JBIG and JBIG2 also use context templates consisting of neighbourhood pixels, they should also be able to apply a similar runlength-based approach to improve performance. This is not investigated further in this paper.

5. Conclusion

It was shown that image compression can be speeded up by using runlength-based methods, at least for common image formats like PNG and JPEG2000. For PNG, image compression could be speeded up many times (at least when one is willing to sacrifice some compression), while for JPEG2000, speeded was obtained in a lesser scale. The results applied for images which can be compactly represented in terms of runlength-representations, like map images. The runlength-based optimisations presented here are constructed to be as simple as possible. It may be possible to produce runlength-based optimisations which outperforms numbers presented here, at the cost of simplicity.

Results in this paper were obtained with an Intel Pentium M processor with 1600MHz clock speed, L2 cache size of 1MB and 512 MB RAM. All tests were run under Windows XP, and all programs were compiled with Microsoft Visual C++ .NET 7.1.

Acknowledgement

The work in this paper is partially based on the RIM library from Faster Imaging AS [http://www.fasterimaging.com]. The post.doc project carried out by Dr. Øyvind Ryan at the University of Oslo has enhanced this implementation, and added algorithms for improved performance and scalability with regards to server applications and memory consumption.

This project has been sponsored by the Norwegian Research Council, project nr. 160130/V30.

REFERENCES