An Extremely Compact JPEG Encoder for Adaptive Embedded Systems

Josef Schneider
Sri Parameswaran
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
{jschneider,sridevan}@cse.unsw.edu.au

Abstract—
JPEG Encoding is a commonly performed application that is also very
process and memory intensive, and not suited for low-power embedded
systems with narrow data buses and small amounts of memory. An
embedded system may also need to adapt its application in order to meet
varying system constraints such as power, energy, time or bandwidth.
We present here an extremely compact JPEG encoder that uses very
few system resources, and which is capable of dynamically changing its
Quality of Service (QoS) on the fly. The application was tested on a NIOS
II core, AVR, and PIC24 microcontrollers with excellent results.

I. INTRODUCTION

Compressing a digital image is often essential as it reduces the
image data size by at least an order of magnitude with very little loss
in quality. This comes at the expense of computational and memory
requirements as the Discrete Cosine Transform (DCT) employed in
JPEG encoding performs many multiplications and memory accesses.
As a result, executing JPEG encoding on a small embedded processor
with varying power, bandwidth or throughput constraints can be
problematic. Tailoring a system for the worst case constraints is
not a good solution as the processor will spend most of the time
underperforming.

To resolve this issue, Peddersen et al. [1] proposed a technique
called “application adaptation”. The application routinely monitors
its constraining factors and varies the QoS accordingly. Peddersen
was capable of meeting power targets in a number of applications
including JPEG encoding. The source code used was developed by
the Independent JPEG Group [2]. This software, though extremely
flexible, instantiates data memory in the order of Megabytes and
also requires a large amount of ROM. This makes it unsuitable for
very small embedded systems. Other available open source JPEG
encoders, such as the Embedded JPEG Codec Library [3], jpe [4]
and Jpegant [5], are not capable of varying the output quality in any
way.

It is also important to note that what is conventionally referred
to as JPEG quality is in fact the amount of compression performed.
This changes the size of the quantisation factors which has a direct
impact on the low-power Huffman encoding stage and the size file.
Independent of the compression chosen, typical implementations do
not change the process-intensive DCT stage which requires a fixed,
large amount of computation. For this reason, we focus here on
quality variation at the DCT level which can easily be combined
with the variation of compression.

The most suitable candidate for our purposes from available open-
source software is “jpeg-compressor” [6] as both the compression
and DCT levels are adaptable. One of its shortcomings is the use of
dynamically allocated memory, which is often undesirable in small
embedded systems. Also, the options for quality variation of a colour
image on the DCT level are limited, as it relies solely on differing
chroma subsampling ratios (i.e., three quality levels 1x1, 2x1, 2x2).

We present here an extremely compact JPEG encoder which is, to
the best of our knowledge, the first of its kind with the following
capabilities:

- very small footprint, requiring 20 to 27kB of ROM and
  a minimum of 5 to 9kB of RAM, depending on the processors
tested;
- it can easily adapt its QoS between frames; and
- the adaptation significantly alters the processing requirements of
  the DCT algorithm. This is done by:
  1) combining different luma and chroma subsampling ratios;
  2) switching between a fast yet inaccurate, and a slow accu-
  rate DCT algorithm; and

II. JPEG ENCODING APPLICATION

The stages of JPEG encoding can be seen in Fig.1. Variation
of the application is primarily achieved by selective downsampling
of YCbCr components (by a factor of 2 in both horizontal and
vertical directions) and changing the DCT algorithm used. In the
JPEG encoding process, an image is first converted from the RGB
into the YCbCr colour space (stage 1) before being sub-divided into
Minimum Coded Unit (MCU) blocks. In our JPEG encoder we only
use a 16x16 MCU block size. If a component is not downsampled
(Fig.2), the MCU of that component is divided into four 8x8 blocks.
Four 8x8 DCTs are then performed producing four 8x8 arrays of
DCT coefficients.

Quality variation of the 8x8 DCT was accomplished by switching
between a fast yet inaccurate, and a slow accurate algorithm. The
slow algorithm is characterised by using 12 multiplications per pass,
whereas the fast algorithm uses 5 multiplications. Note that 16 passes
are computed for each 8x8 block. Downsampling, the process of
reducing the DCT output of an MCU to one single 8x8 array, was
performed in two ways. Either the entire MCU was computed by
a 16x16 DCT algorithm (Fig.3) or the MCU was first averaged
to obtain an 8x8 array before executing an 8x8 DCT (Fig.4). The
former produces the better quality output, though it requires 28
multiplications per pass, and 24 passes.

The number of multiplications required for each process mentioned
can be seen in Table I. Though the computational requirements of an
algorithm are not solely defined by the multiplication count, this table
gives a rough idea of the processing requirements of the different

Fig. 1. Different Stages of JPEG encoding.

Fig. 2. Discrete Cosine Transform of an MCU block that is not downsampled.

3) performing downsampling by averaging, or directly
through a 16x16 DCT.

Additionally the embedded designer can easily vary the quality by
changing the amount of compression.

978-3-9815370-0/0/DATA13/ © 2013 EDAA
JPEG encoding format allows for different downsampling rates for the different components of the YCbCr colour space. The most common configuration is no downsampling of the luminance component (Y), while both chrominance components (Cb and Cr) are downsampled by a factor of 4; this is known as the 4:2:0 ratio. A 4:2:0 configuration eliminates features in both color components, by multiplexing them with previous color information. On the other hand, downsampling all of the components effectively lowers the image resolution by a factor of four, i.e., a 640x480 pixel image is transformed into a 320x240 pixel image with a 4:4:4 ratio. Combining the different luminance and chrominance downsampling ratios and conversion methods we created 9 quality levels which can be seen in Table II. By changing the quality level, the application can adapt its processing requirements.

### III. PERFORMANCE

An input image size of 640x480 pixels was used in our experiments. From Table III it can be seen that the output compressed file size mostly depends on the chosen YCbCr downsampling rates. The JPEG encoding application was implemented on three different embedded processors: an Atmel AVR ATmega1280 (8 bit), a Microchip PIC24FJ256GB110 (16 bit) and an Altera NIOS II softcore processor (32 bit) running only from on-chip memory with 8kB instruction and 8kB data caches. For fair comparison, the timing values were scaled to represent operation at 16MHz. The source code was compiled with the respective gcc compiler optimising for size (-Os). Memory usage estimates can be seen in Table IV while Fig.5 shows the timing at each quality level. A compression level of Q=90 was used at each run.

![Fig. 3. Downsampling through the use of a 16x16 DCT.](image)

![Fig. 4. Downsampling by averaging.](image)

![Fig. 5. Time it takes to encode a frame on the different processors.](image)

### REFERENCES