Intel Integrated Performance Primitives for Intel Architecture

Intel® Integrated Performance Primitives Version of the IJG JPEG Library

Version 2.4

October, 2010

Table of Contents

Intel® Integrated Performance Primitives

Optimization Notice

The Intel® Integrated Performance Primitives (Intel® IPP) library contains functions that are more highly optimized for Intel microprocessors than for other microprocessors. While the functions in the Intel® IPP library offer optimizations for both Intel and Intel-compatible microprocessors, depending on your code and other factors, you will likely get extra performance on Intel microprocessors.

While the paragraph above describes the basic optimization approach for the Intel® IPP library as a whole, the library may or may not be optimized to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

Intel recommends that you evaluate other library products to determine which best meets your requirements.

This document provides supporting information on how to use the code sample that implements a modification of the Independent JPEG Group (IJG) JPEG library by means of using embedded Intel® Integrated Performance Primitives (Intel® IPP).

The Intel IPP software is a new generation of the Intel® Performance Libraries, comprising a broad range of functions for basic software functionality and including, among many others, the JPEG coding functions subset.

Intel IPP was developed as a low-level basic software layer that hides the differences of diverse hardware platforms, presents uniform API, and provides for developing high performance applications on different platforms. Intel IPP can be used on different hardware platforms including IA-32, Itanium® -based systems and Intel XScale® microarchitecture, and in several operating systems, specifically 32-bit and 64-bit Windows*, Windows CE*, Linux* and ARM* Linux. This diversity explains why Intel IPP can help to develop portable software.

Significant performance gains (without optimization efforts) can be achieved by means of using calls to the Intel IPP low-level software in customer application programs. This paper also presents the performance test results that show advantage of the Intel IPP code use over the original code.

Additional information on this software as well as other Intel® software performance products is available at http://www.intel.com/software/products/.

To give feedback or report any problems with installation or use of this software, please contact via Intel® Premier Support at http://premier.intel.com. For registration information, please check http://support.intel.com/support/performancetools/libraries/ipp.

Using Intel IPP to Modify the IJG Library

The Independent JPEG Group (IJG) library (see http://www.ijg.org) is well known among multiple JPEG libraries and codecs that have been created since the first JPEG standard was published.

New technologies implemented in the last generation Intel® microprocessors with Streaming SIMD instructions provide a good opportunity for the application programmers to increase performance of the JPEG codecs.

To take full advantage of the new Intel architecture and thus achieve a performance gain, it is more effective to use embedded Intel IPP functions rather then to write programs straight in assembler for the new instruction set.

In the code sample that implements the modified version of the IJG library, several parts of this library were substituted by the Intel IPP JPEG primitives without loss of functionality. Performance of the modified codec was tested in comparison with the original IJG version 6B.

Optimization Notice

The Intel® Integrated Performance Primitives (Intel® IPP) library contains functions that are more highly optimized for Intel microprocessors than for other microprocessors. While the functions in the Intel® IPP library offer optimizations for both Intel and Intel-compatible microprocessors, depending on your code and other factors, you will likely get extra performance on Intel microprocessors.

While the paragraph above describes the basic optimization approach for the Intel® IPP library as a whole, the library may or may not be optimized to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

Intel recommends that you evaluate other library products to determine which best meets your requirements.

Structure of the JPEG Codec

The below diagram illustrates the block structure of the modified JPEG codec implemented on the basis of the IJG library. Every component contains the names of Intel IPP functions that have substituted the original IJG code.

Color conversion routines:

ippiRGBToY_JPEG_8u_C3C1R

ippiRGBToYCbCr_JPEG_8u_C3P3R

ippiCMYKToYCCK_JPEG_8u_C4P4R

ippiYCbCrToRGB_JPEG_8u_P3C3R

ippiYCCKToCMYK_JPEG_8u_P4C4R

Subsampling routines:

ippiSampleDownRowH2V1_Box_JPEG_8u_C1

ippiSampleDownRowH2V2_Box_JPEG_8u_C1

ippiSampleUpRowH2V1_Triangle_JPEG_8u_C1

ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1

DC level shift, DCT and Quantization routines:

ippiQuantFwdTableInit_JPEG_8u16u

ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R

ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R

Entropy coding:

ippiEncodeHuffman8x8_JPEG_16s1u_C1

ippiDecodeHuffman8x8_JPEG_1u16s_C1

Intel IPP Embedding

This section gives a description of the altered files used to embed Intel IPP functions into the modified IJG library.

In IPP IJG sample calls to IPP functions are enabled by default. You have an option to disable calls to IPP functions at compile time. To do that, change following lines in the file `jconfig.h`:

lines	from	to
44	//#undef USE_IPP	#undef USE_IPP
45	#define USE_IPP	//#define USE_IPP

You also can selectively exclude groups of IPP functions, like sub-sampling, color-conversion or huffman by undefining appropriate macros in `jconfig.h` file.

Table 1 provides a summary list of modified files that are part of the code sample, together with names and descriptions of modified functions contained in respective files.

Table 1. Modified Files and Functions with Embedded Intel IPP

File Name	Names of Modified Functions	Comment

`jccolor.c`	`rgb_ycc_convert_intellib,`
	`rgb_gray_convert_intellib,`
	`cmyk_ycck_convert_intellib,`
	`jinit_color_converter`
		The file contains modified color conversion functions of the JPEG encoder.

`jdcolor.c`	`ycc_rgb_convert_intellib,`
	`ycck_cmyk_convert_intellib,`
	`jinit_color_deconverter`
		The file contains modified color conversion functions of the JPEG decoder.

`jcdctmgr.c`	`forward_DCT_intellib,`
	`jinit_forward_dct,`
	`start_pass_fdctmgr`
		The file contains the modified forward DCT function.
		In order to create quantization tables applicable in Intel IPP functions, an array with raw quantization coefficients has been added to the source code of the `start_pass_fdctmgr` function. This array contains 64 coefficient values of the `Ipp8u` type[1)] .
		The code segment to transform raw quantization tables of the IJG format to tables of the Intel IPP format, as well as the code for fast generation of the quantization tables from the raw tables have also been added.
		A temporary buffer used in the forward DCT procedure ensures that the same computational path is chosen even if the step between adjacent rows is varied in the IJG library.

`jddctmgr.c`	`start_pass`	The modified `start_pass` function allows to use modified Intel IPP version of the inverse DCT function.
`jidctint.c`	`jpeg_idct_islow_intellib`	The file contains the modified version of the original inverse DCT function `jpeg_idct_islow` . A temporary buffer used in the inverse DCT procedure ensures that the same computational path is chosen even if the step between adjacent rows is varied in the IJG library.

`jcsample.c`	`h2v1_downsample_intellib,`
	`h2v2_downsample_intellib,`
	`jinit_downsampler`
		Optimized version of the original functions `h2v1_downsample` and `h2v2_downsample`.

`jdsample.c`	`h2v1_fancy_upsample_intellib,` `h2v2_fancy_upsample_intellib,`	Optimized version of the original functions `h2v1_fancy_upsample` and `h2v2_fancy_upsample`.

	jinit_upsampler	
jchuff.h		Contains the definition of c_derived_tbl with changed data type. The change in data type is required for the modification of Huffman encoder functions.
jchuff.c		Contains the definition of savable_state with changed data type and also the changed macro ASSIGN_STATE . The change in data type is required for the modification of Huffman encoder functions. Modified functions contained in jchuff.c that use calls to Intel IPP are introduced below:
	start_pass_huff	initializes the Huffman encoder.
	jpeg_make_c_derived_tbl_intellib	computes the Huffman tables
	dump_buffer_intellib	dumps the inner JPEG buffer into the file or output buffer according to the jpeg_destination_mgr module realization
	flush_bits_intellib	flushes the inner bit buffer
	encode_one_block_intellib	performs Huffman encoding of an 8x8 block of quantized DCT coefficients
	emit_restart_intellib	flushes the Huffman codec when the restart interval is encoded
	encode_mcu_huff_intellib	processes one MCU and calls the above described functions
	finish_pass_huff	flushes the Huffman codec in the end of each scan
	htest_one_block_intellib	collects the Huffman symbols statistics that will be used later for generating the optimal Huffman tables
	encode_mcu_gather_intellib	processes one MCU
	jpeg_gen_optimal_table_intellib	generates optimal Huffman tables using the previously collected Huffman symbols statistics
	finish_pass_gather	flushes the Huffman codec in the end of the pass
jdhuff.h		Contains the definition of d_derived_tbl with changed data type. The change in data type is required for the modification of Huffman decoder functions.
jdhuff.c		Contains the definition of savable_state with changed data type and also the changed macro ASSIGN_STATE . The change in data type is required for the modification of Huffman decoder functions. Modified functions contained in jdhuff.c that use calls to Intel IPP are introduced below:
	start_pass_huff_decoder	initializes the Huffman decoder and creates tables
	jpeg_make_d_derived_tbl_intellib	computes the Huffman tables

	process_restart	flushes the Huffman codec when the restart interval is decoded
	decode_mcu_intellib	decodes data for one MCU
	jinit_huff_decoder	initializes the Huffman decoder
jdatasrc.c	fill_input_buffer_intellib, jpeg_stdio_src	Contains functions for preliminary sampling of decoder data in compliance with Intel IPP functions requirements.
jdatadst.c	empty_output_buffer_intellib, jpeg_stdio_dest	Contains functions that output encoded data into the file in compliance with Intel IPP functions requirements.

1) For the definition of data types used in Intel IPP functions, refer to the Intel IPP Reference Manual included with the Intel IPP package.

Accuracy of the Inverse DCT Transform

The `ippiDCT8x8Inv_16s_C1` function that implements the inverse DCT transform in Intel IPP was tested using the special IEEE-1180 compliance test which can be downloaded from ftp://ftp.mpegtv.com/pub/mpeg/mssg/ieee1180.tar.gz .

The test was run on the Pentium® 4 processor-based system described in the Test System Specification section. The test output results are given below. These results indicate that the actual accuracy of the `ippiDCT8x8Inv_16s_C1` function meets the IEEE-1180 standard requirements.

```
IEEE test conditions: -L = -256, +H = 255, sign = 1, #iters = 10000

Peak absolute values of errors:
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
Worst peak error = 1  (meets spec limit 1)
Mean square errors:
   0.0126   0.0115   0.0134   0.0121   0.0116   0.0126   0.0125   0.0130
   0.0142   0.0157   0.0128   0.0134   0.0124   0.0145   0.0125   0.0138
   0.0140   0.0134   0.0127   0.0130   0.0132   0.0152   0.0138   0.0125
   0.0110   0.0139   0.0118   0.0130   0.0118   0.0145   0.0115   0.0121
   0.0117   0.0134   0.0126   0.0110   0.0112   0.0112   0.0124   0.0149
   0.0148   0.0129   0.0138   0.0137   0.0134   0.0126   0.0139   0.0126
   0.0136   0.0136   0.0130   0.0141   0.0129   0.0136   0.0146   0.0136
   0.0134   0.0130   0.0110   0.0127   0.0131   0.0133   0.0123   0.0130

Worst pmse = 0.015700  (meets spec limit 0.06)
Overall mse = 0.013014  (meets spec limit 0.02)
Mean errors:
   0.0006  -0.0003   0.0012  -0.0007   0.0002   0.0006  -0.0009   0.0006
  -0.0022  -0.0013   0.0004  -0.0004   0.0022  -0.0003  -0.0005  -0.0008
  -0.0012  -0.0018   0.0007  -0.0010   0.0002   0.0010   0.0002  -0.0001
  -0.0004  -0.0005   0.0000   0.0006   0.0010  -0.0025   0.0001   0.0011
  -0.0015   0.0006   0.0010  -0.0002  -0.0002  -0.0002   0.0000  -0.0011
   0.0022  -0.0005  -0.0002  -0.0017  -0.0008   0.0010   0.0007  -0.0004
  -0.0006  -0.0020  -0.0012   0.0015  -0.0001  -0.0014   0.0002  -0.0006
   0.0008   0.0002   0.0006  -0.0007   0.0011  -0.0009   0.0021   0.0018

Worst mean error = 0.002500  (meets spec limit 0.015)
Overall mean error = -0.000073  (meets spec limit 0.0015)
0 elements of IDCT(0) were not zero

IEEE test conditions: -L = -5, +H = 5, sign = 1, #iters = 10000
```

```
Peak absolute values of errors:
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
Worst peak error = 1  (meets spec limit 1)
Mean square errors:
   0.0128   0.0146   0.0132   0.0128   0.0115   0.0114   0.0122   0.0140
   0.0138   0.0128   0.0136   0.0124   0.0140   0.0143   0.0146   0.0141
   0.0135   0.0148   0.0141   0.0126   0.0121   0.0152   0.0132   0.0136
   0.0148   0.0103   0.0112   0.0135   0.0123   0.0120   0.0119   0.0117
   0.0116   0.0125   0.0146   0.0141   0.0101   0.0139   0.0115   0.0122
   0.0139   0.0136   0.0136   0.0133   0.0125   0.0133   0.0140   0.0134
   0.0150   0.0124   0.0146   0.0140   0.0118   0.0141   0.0129   0.0141
   0.0120   0.0118   0.0115   0.0126   0.0137   0.0130   0.0128   0.0109
Worst pmse = 0.015200  (meets spec limit 0.06)
Overall mse = 0.013034  (meets spec limit 0.02)
Mean errors:
  -0.0004   0.0024   0.0000   0.0014   0.0007  -0.0002   0.0024  -0.0002
   0.0002   0.0000   0.0002  -0.0012   0.0034   0.0017   0.0000   0.0009
  -0.0009  -0.0006   0.0017   0.0014  -0.0019  -0.0016  -0.0008   0.0012
  -0.0018  -0.0001  -0.0010   0.0013  -0.0013   0.0022   0.0015  -0.0013
  -0.0002   0.0001   0.0008  -0.0021  -0.0011  -0.0017   0.0011  -0.0002
  -0.0001   0.0000  -0.0004   0.0001  -0.0021   0.0017   0.0012   0.0010
   0.0010   0.0004  -0.0006   0.0004  -0.0006  -0.0023  -0.0003  -0.0019
   0.0012  -0.0010   0.0001  -0.0002   0.0001   0.0000   0.0014  -0.0001
Worst mean error = 0.003400  (meets spec limit 0.015)
Overall mean error = 0.000078  (meets spec limit 0.0015)
0 elements of IDCT(0) were not zero

IEEE test conditions: -L = -300, +H = 300, sign = 1, #iters = 10000

Peak absolute values of errors:
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
Worst peak error = 1  (meets spec limit 1)
```

```
Mean square errors:
   0.0101   0.0111   0.0115   0.0116   0.0120   0.0090   0.0118   0.0122
   0.0084   0.0111   0.0124   0.0112   0.0101   0.0103   0.0117   0.0112
   0.0106   0.0092   0.0131   0.0101   0.0131   0.0093   0.0125   0.0116
   0.0094   0.0098   0.0120   0.0109   0.0123   0.0114   0.0105   0.0094
   0.0111   0.0112   0.0092   0.0117   0.0102   0.0100   0.0106   0.0109
   0.0111   0.0108   0.0120   0.0113   0.0117   0.0112   0.0102   0.0106
   0.0112   0.0108   0.0102   0.0127   0.0112   0.0117   0.0117   0.0117
   0.0113   0.0100   0.0099   0.0105   0.0121   0.0102   0.0105   0.0112
Worst pmse = 0.013100  (meets spec limit 0.06)
Overall mse = 0.010963  (meets spec limit 0.02)
Mean errors:
   0.0003  -0.0003  -0.0007   0.0018   0.0010  -0.0006   0.0002   0.0012
   0.0002   0.0003   0.0018  -0.0018   0.0003  -0.0019   0.0013  -0.0012
  -0.0004  -0.0006  -0.0005  -0.0001   0.0003  -0.0003   0.0011   0.0010
   0.0008  -0.0008   0.0002  -0.0009   0.0005  -0.0004   0.0005  -0.0004
  -0.0011  -0.0014   0.0020  -0.0001  -0.0004  -0.0006  -0.0002  -0.0003
   0.0021   0.0006   0.0000  -0.0019   0.0003  -0.0008  -0.0012  -0.0006
  -0.0012  -0.0008  -0.0006   0.0003  -0.0006  -0.0003   0.0019  -0.0007
  -0.0009  -0.0010  -0.0009  -0.0001  -0.0005   0.0010   0.0019   0.0016
Worst mean error = 0.002100  (meets spec limit 0.015)
Overall mean error = -0.000041  (meets spec limit 0.0015)
0 elements of IDCT(0) were not zero

IEEE test conditions: -L = -256, +H = 255, sign = -1, #iters = 10000

Peak absolute values of errors:
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
Worst peak error = 1  (meets spec limit 1)
Mean square errors:
   0.0125   0.0127   0.0122   0.0120   0.0129   0.0119   0.0120   0.0142
   0.0126   0.0151   0.0115   0.0154   0.0134   0.0151   0.0138   0.0136
   0.0146   0.0147   0.0130   0.0142   0.0132   0.0141   0.0132   0.0110
   0.0126   0.0122   0.0141   0.0136   0.0124   0.0141   0.0119   0.0109
   0.0110   0.0145   0.0125   0.0122   0.0133   0.0125   0.0104   0.0142
   0.0123   0.0132   0.0130   0.0127   0.0140   0.0139   0.0149   0.0133
   0.0130   0.0131   0.0127   0.0148   0.0124   0.0114   0.0142   0.0141
   0.0125   0.0127   0.0121   0.0131   0.0115   0.0109   0.0145   0.0133
Worst pmse = 0.015400  (meets spec limit 0.06)
Overall mse = 0.013045  (meets spec limit 0.02)
```

```
Mean errors:
   0.0009  -0.0007  -0.0010  -0.0006  -0.0009  -0.0005   0.0000  -0.0002
  -0.0008   0.0019  -0.0013   0.0002  -0.0014   0.0001   0.0008   0.0000
   0.0008   0.0013   0.0016   0.0000   0.0026  -0.0003   0.0002   0.0004
   0.0018  -0.0026   0.0007   0.0006  -0.0002   0.0001   0.0009  -0.0011
   0.0028  -0.0005  -0.0003  -0.0006  -0.0003   0.0011   0.0006  -0.0002
  -0.0021   0.0010  -0.0004   0.0005   0.0002   0.0019   0.0003   0.0001
  -0.0004   0.0003   0.0017  -0.0002   0.0004   0.0018  -0.0010   0.0011
   0.0003   0.0001  -0.0005   0.0001  -0.0005   0.0005   0.0011  -0.0007
Worst mean error = 0.002800  (meets spec limit 0.015)
Overall mean error = 0.000180  (meets spec limit 0.0015)
0 elements of IDCT(0) were not zero

IEEE test conditions: -L = -5, +H = 5, sign = -1, #iters = 10000

Peak absolute values of errors:
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
Worst peak error = 1  (meets spec limit 1)
Mean square errors:
   0.0127   0.0131   0.0132   0.0121   0.0132   0.0116   0.0113   0.0109
   0.0135   0.0146   0.0147   0.0142   0.0145   0.0110   0.0158   0.0143
   0.0121   0.0135   0.0124   0.0124   0.0124   0.0138   0.0116   0.0129
   0.0140   0.0113   0.0121   0.0134   0.0138   0.0114   0.0111   0.0111
   0.0117   0.0119   0.0151   0.0123   0.0124   0.0134   0.0132   0.0117
   0.0133   0.0135   0.0141   0.0122   0.0130   0.0132   0.0124   0.0143
   0.0142   0.0131   0.0138   0.0136   0.0147   0.0123   0.0127   0.0137
   0.0120   0.0128   0.0118   0.0138   0.0151   0.0128   0.0129   0.0121
Worst pmse = 0.015800  (meets spec limit 0.06)
Overall mse = 0.012955  (meets spec limit 0.02)
Mean errors:
  -0.0009  -0.0007   0.0018  -0.0017   0.0000   0.0010  -0.0003   0.0005
  -0.0015   0.0018  -0.0007   0.0000   0.0003  -0.0002   0.0016  -0.0009
   0.0003  -0.0011  -0.0008   0.0004  -0.0004  -0.0020  -0.0006  -0.0009
   0.0018  -0.0021   0.0013  -0.0012   0.0008  -0.0002  -0.0013  -0.0005
  -0.0011  -0.0013   0.0021   0.0011  -0.0014   0.0006  -0.0004  -0.0003
  -0.0013   0.0001  -0.0001  -0.0004   0.0002   0.0014   0.0000  -0.0003
   0.0010  -0.0005   0.0012  -0.0014   0.0007  -0.0015  -0.0005  -0.0009
  -0.0010   0.0002  -0.0014   0.0010   0.0007   0.0008   0.0007   0.0003
Worst mean error = 0.002100  (meets spec limit 0.015)
Overall mean error = -0.000127  (meets spec limit 0.0015)
0 elements of IDCT(0) were not zero

IEEE test conditions: -L = -300, +H = 300, sign = -1, #iters = 10000
```

```
Peak absolute values of errors:
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1
Worst peak error = 1  (meets spec limit 1)
Mean square errors:
   0.0108   0.0114   0.0117   0.0119   0.0095   0.0089   0.0117   0.0117
   0.0107   0.0116   0.0102   0.0111   0.0102   0.0118   0.0117   0.0112
   0.0113   0.0090   0.0136   0.0109   0.0151   0.0118   0.0106   0.0115
   0.0098   0.0097   0.0122   0.0114   0.0126   0.0102   0.0101   0.0104
   0.0088   0.0107   0.0099   0.0124   0.0110   0.0095   0.0116   0.0102
   0.0095   0.0106   0.0129   0.0119   0.0108   0.0101   0.0105   0.0119
   0.0127   0.0119   0.0104   0.0092   0.0113   0.0100   0.0105   0.0093
   0.0102   0.0121   0.0097   0.0113   0.0121   0.0096   0.0110   0.0116
Worst pmse = 0.015100  (meets spec limit 0.06)
Overall mse = 0.010961  (meets spec limit 0.02)
Mean errors:
  -0.0008  -0.0002   0.0013  -0.0005  -0.0007   0.0013   0.0003  -0.0017
  -0.0001  -0.0016  -0.0006  -0.0003  -0.0020  -0.0004  -0.0031   0.0002
   0.0003   0.0004  -0.0004  -0.0011   0.0003  -0.0002   0.0000  -0.0005
   0.0006   0.0019  -0.0010  -0.0014   0.0000   0.0010  -0.0005   0.0008
   0.0002  -0.0013  -0.0017   0.0024   0.0002   0.0009  -0.0006  -0.0006
  -0.0013   0.0002  -0.0017   0.0027  -0.0004  -0.0001  -0.0005  -0.0001
   0.0019   0.0025  -0.0002  -0.0010  -0.0005   0.0006  -0.0005  -0.0001
   0.0004   0.0009   0.0001   0.0001   0.0005  -0.0030  -0.0016  -0.0016
Worst mean error = 0.003100  (meets spec limit 0.015)
Overall mean error = -0.000186  (meets spec limit 0.0015)
0 elements of IDCT(0) were not zero
```

Accuracy of the JPEG Codec

The test application **`ijg_timing.exe`** performs an accuracy test of the entire JPEG codec. Run the "**`ijg_timing –t <BMP_NAME>`**" command to estimate the accuracy of the codec. The test encodes a source BMP file and then decodes the encoded data. The result and the original data are compared. The maximum difference (norm C), relative square error (norm RL2) and some other norm values that can be computed for the version that uses Intel IPP primitives are presented in the **Table 2** below. These results have been obtained for the Lenna's test image.

Table 2. Accuracy Results of the JPEG Codec

Norm	IJG			IJG+ Intel IPP		
	channel 1	channel 2	channel 3	channel 1	channel 2	channel 3
C	4	3	4	4	3	4
L1	161458	85055	133094	157883	82401	130166
RL1	0.0058	0.0033	0.0028	0.0057	0.0032	0.0028
L2	218690	92275	165682	210217	88695	159480
RL2	0.0082	0.0053	0.0043	0.0081	0.0052	0.0042

Performance Results

Optimization Notice

The Intel® Integrated Performance Primitives (Intel® IPP) library contains functions that are more highly optimized for Intel microprocessors than for other microprocessors. While the functions in the Intel® IPP library offer optimizations for both Intel and Intel-compatible microprocessors, depending on your code and other factors, you will likely get extra performance on Intel microprocessors.

While the paragraph above describes the basic optimization approach for the Intel® IPP library as a whole, the library may or may not be optimized to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include Intel® Streaming SIMD Extensions 2 (Intel® SSE2), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), and Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3) instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors.

Intel recommends that you evaluate other library products to determine which best meets your requirements.

Performance numbers obtained for the original version of IJG library and for the modified Intel IPP version are given in the **Table 3**.
The performance measurements were done on the Intel® Pentium® M processor-based system detailed in the Test System Specification section.

Images of size 640x480 pixels with 1, 3 and 4 channels were used for the tests; the pixels had random values.

The resulting figures (given in the cpu-cycles per pixel) indicate that the use of Intel IPP code increases performance on given processors at least 1.5-2.0 times for both the encoder and decoder.

Table 3. Performance Data

JPEG parameters			Pentium® M processor (1.7 GHz)		
Codec	**Channels**	**Sampling**	**IJG**	**IJG+ Intel IPP**	**speedup**
encoder	1	444	276.2	72.8	3.8
	3	444	772.7	203.6	3.8
	3	422	477.9	144.5	3.3
	3	411	377.9	113.5	3.3
	4	444	1028.7	279.9	3.7
	4	422	843.8	221.6	3.8
	4	411	901.4	199.6	4.5
decoder	1	444	164.1	50.4	3.3
	3	444	267.1	122.2	2.2
	3	422	191.3	90.6	2.1
	3	411	162.9	75.4	2.2
	4	444	383.6	174.1	2.2
	4	422	289.1	137.2	2.1
	4	411	261.9	121.5	2.

Test System Specification

Configuration of the base machine used for testing is given in the table below:

Table 4. System Configurations

Processor Type	Intel Pentium M processor
Stepping	Model 9 Step 5
CPU clock frequency	1700MHz
Bios version	02/25/04
L2 Cache size	1024Kb
Main memory size	1024Mb
Operating system	Windows* XP

* * *

The code sample described in this document fully implements the IJG JPEG library functionality and demonstrates advantages that can be achieved by using Intel® Integrated Performance Primitives in application programs running on latest generation Intel® processors.