The Basics of Performance Libraries for Embedded Systems

ABSTRACT: With each successive generation of processors, the number of developers who want to program in assembly language is diminishing. Further, every generation has required a new round of assemblers, compilers, and a new set of developer expertise. Tools vendors recognized a need to provide developers with software tools to support every processor release with a variety of functionality, a storehouse of optimized code for developers to draw upon. Thus performance libraries targeting several application domains have been developed. This article discusses the basics of performance libraries, and how different areas of computer science benefit.

Performance libraries are constructed by first identifying common functions used in different application areas and then producing optimized routines to perform the required functionality. These optimized routines are built from lower level primitive operations ordered in such a way as to maximize performance on a given hardware platform.

These routines can be classified into domain specific groupings. These groupings pertinent to discussing Intel IPP are as follows:

- Matrix and vector mathematics
- Signal processing and digital filtering
- Cryptography
- Image compression and decompression
- Video encode and decode
- Image processing and object recognition
- String processing

The next sections summarize each of these application groupings.

Matrix and Vector Mathematics

Matrix and vector mathematics are the building blocks for higher level library functionality and are also employed in all manner of scientific and engineering computation. Thus, many performance libraries provide basic linear algebra capabilities offering optimized functions such as those for dot products, matrix multiplication, inverse, and normalization. BLAS1 is the prototypical matrix and vector mathematics library.
Signal Processing and Digital Filtering

Signal processing is the analysis of, typically, time series data measurements of some physical events. These events can be sound, images, or other sampled representation of a physical occurrence. The types of processing on these signals include filtering, analysis, and transformations. Applications for signal processing include audio content creation and speech recognition.

Digital filters take a series of inputs, manipulate them in some way without destroying all of the original information, and produce a series of outputs. This “some way” generally preserves most characteristics of the original data but removes others. Often it involves noise removal or information extraction. The inputs are often samples from some sensor or medium, such as audio from a microphone or CD player, or samples of a carrier wave from an antenna; however, regardless of the source and meaning, the time at which the samples are filtered is expressed as numerical entries in an array.

Filters are classified based upon a number of different properties:

- **Linear versus nonlinear.** A linear filter is one that meets the following criteria:
  - Adding two inputs then filtering produces the same result as filtering and then adding the two outputs:
    \[ F(x_1 + x_2) = F(x_1) + F(x_2). \]
  - Multiplying the input by a constant then filtering produces the same result as filtering then multiplying the output by a constant:
    \[ F(ax_1) = aF(x_1). \]

- **Time-invariant versus time-varying.** Time-invariant filters are defined as systems for which shifting the input in time shifts the output by the same amount without otherwise affecting the output. Linearity and time-invariance simplifies both study and implementation of filters. The core Intel IPP filter operations are time-invariant and linear, because linear filters can be expressed as an array, and time-invariant filters are those for which that array doesn’t change in the life of the filter. Frequency-based filters are inherently time-invariant.

- **Causal versus noncausal.** A causal function is one for which the output at time \( n \) only refers to inputs from time \( n \) or earlier.

- **Real-time versus batch mode.** If the software system is taking input signals and producing outputs for immediate consumption, it might be
thought of as a real-time system. By contrast, systems that have the entire signal available at once and filter it are termed batch-mode.

Fourier Transforms

The Fourier transform converts a signal indexed by time into a signal indexed by frequency. The inverse Fourier transform converts this signal back into the original time-based signal. The same information is contained in both the time-based and frequency-based versions of the signal, but the information is arranged differently.

The Fourier transform calculates the set of sinusoids that best represent the original signal. For this reason the sinusoid is called the basis function of the Fourier transform. A related transform, the Discrete Cosine Transform (DCT), also uses sinusoids, but transforms can and do use square waves, triangle waves, and so on as their basis functions. The use of the sinusoid by the Fourier transform matches a typical physical or electrical oscillation and is therefore appropriate to break down frequency in audio and other physical systems.

The version of this transform that is relevant to software-based filtering is the Discrete Fourier Transform (DFT). The DFT is very similar to the Fourier series, in that it breaks a signal into a series of frequencies.

Appealing characteristics of the Fourier transform with respect to digital filtering include:

- **Linearity.** The Fourier transform is a linear function. One effect of this function is that the time-domain signal can be interpreted as the sum of several independent frequency components. If each single spike transforms to a sine wave, then several spikes transform to the sum of the corresponding sine waves.

- **Reciprocity.** The Fourier transform is reversible. It is possible to reconstruct the original signal from the information in the Fourier coefficients. If it weren’t possible, the DFT wouldn’t be useful for filtering, only for analysis.

Classifying Filters

Filters are often classified as “pass” or “stop,” meaning that they either attenuate a frequency range or attenuate all but that range. This is generally modified with “low,” “high,” or “band,” indicating that the range is the low end of the frequency spectrum, the high end, or a contiguous range of frequencies in the middle. For example, a high-pass filter tries to remove all but a specified range of high frequencies, while a band-stop filter tries to
eliminate a range of frequencies. A band-stop filter is often referred to as a “notch” filter, particularly when the band in question is narrow.

*Time-Domain and Two-Dimensional Filters*

Frequency-domain filtering is conceptually simple and intuitive. Time-domain filtering, on the other hand, can be mathematically simple but generally has less-than-intuitive results. Frequency domain filtering typically consists of simple multiplications to enhance or attenuate certain frequencies. In the time domain, that simple element-by-element multiplication with a frequency spectrum becomes a convolution with a signal. The filter that was designed to modify the frequencies becomes an almost random sequence of filter coefficients.

The same mathematical operations that produce filters in one-dimensional signals can be applied to two-dimensional images, generally by a simple expansion of the formula. Image filters can be designed and analyzed using the Fourier transform, and filters can be executed in the frequency domain or using convolution.

**Cryptography**

Cryptography is the processing of data to conceal any of the information it contains. Algorithms for encoding data are usually complicated and almost always time-consuming. While the cryptographic domain includes some primitive components, the domain has about a dozen popular algorithms implemented completely. The functions can be divided into four groups: symmetric encryption algorithms, public key encryption algorithms, hashing functions, and primitives.

The security of cryptography schemes are generally based on the expense of attempting randomly to decrypt balanced with the practicality of the time to correctly encrypt and decrypt. For this reason, large integers are the core of many cryptographic schemes. Big numbers in this case are generally more than 64 bits long and less than 4,096.

Public-key cryptography allows an entity that wishes to receive secure transmissions to create a system for encrypting data that can only be read by that entity. Such an entity creates a pair of keys based on large primes. The public key is published or provided to any other entity that wishes to send a secure transmission. The private key and primes are kept secret and are used to decode any transmissions encoded with the public key.
The result is an easy and secure transmission method that anyone with the public key can use. Further, if some authority certifies the identity of the owner of the public key, then this method doubles as part of an identity certification scheme, since only the person certified can read the encrypted messages.

Categories of cryptographic algorithms and specific algorithms are listed as follows:

- **Symmetric cryptography:**
  - Data Encryption Standard (DES) and Triple Data Encryption Standard (TDES)
  - Rijndael, Blowfish, and Twofish block ciphers

- **Hash and data authentication algorithm (DAA) functions:**
  - MD5, HMAC-MD5
  - SHA1, SHA256/384/512, HMAC-SHA1, HMACSHA256/384/512
  - DAADES, DAATDES
  - DAARijdael, DAABlowfish, DAATwofish

- **Public key cryptography:**
  - Infrastructure functions such as pseudorandom number generation (PNRG) and prime number generation
  - Digital Signature Algorithm (DSA)

**Image Compression and Decompression**

Image and video encoders and decoders, in software called *codecs*, are intended to compress their media for storage or transmission. Raw images are quite large and raw digital video is unreasonably large, absorbing disk or network capacity very quickly. Moreover, processor speed is sufficient that working with these media uncompressed, except for capture and display, is completely unnecessary and inefficient. It is faster to read compressed video from disk and decompress it than it would be to read uncompressed video.

Most compression is based on taking advantage of redundancy and predictability in data to reduce the number of bytes necessary to represent it. Two common techniques are run-length coding, which converts runs of data into run-lengths and values, and variable-length coding, which converts data of fixed bit lengths into variable bit lengths according to popularity. Huffman coding and arithmetic coding are examples of variable-length coding.

Another source of compression is perceptibility. For some kinds of data, such as text and binary executables, compression must be lossless. A compression method that sometimes changed an “a” to an “A” would not be acceptable. Standalone Huffman coding is exactly reversible. However, it is possible to compress media information in a way that is not exactly
reversible but is virtually undetectable. Such methods are called *lossy*, meaning that the output is not guaranteed to be exactly the same as the input. However, in many cases the loss can be imperceptible or have acceptable visual effect. Just as with audio coding, the compression algorithm transforms the data into spaces in which information can be removed while minimizing the perceptible impact to the media.

Most media compression is done using transform-based coding methods. Such methods convert the position-based information into frequency-based or position/frequency-based information. The compression benefit is that important information becomes concentrated in fewer values. Then the coder can represent the more-important information with more bits and the less-important information with fewer bits. The perception model dictates the importance of information, but generally higher-frequency information is considered less important.

Figure 1 shows the framework of a transform-based encoding and decoding scheme.

![Figure 1: Simple Diagram of Transform-Based Image Coding](image.png)

*JPEG*

JPEG is a widely used standard for image compression. The term is an acronym that stands for the Joint Photographic Experts Group, the body that designed the specification. JPEG is a powerful and effective compression
technique that has been around for over a decade. It has gained popularity because of its effectiveness, because it is an international standard, and because a baseline JPEG codec can be built without using proprietary algorithms.

The Independent JPEG Group (IJG) is a small association that wrote a reference implementation of the JPEG standard. This implementation, often called the IJG library or merely IJG, is widely used as a JPEG codec.

*Data Compression Principles*

This section summarizes key data compression algorithms. The descriptions are brief and many details are neglected, so these descriptions should be treated as a simple illustration of main ideas of the algorithms. Key descriptions of data compression techniques are as follows:

- **Huffman coding.** Huffman coding is a common example of a variable-length code. Whereas most (uncompressed) schemes encode characters using a fixed number of bits, often 8 or 16, the idea of Huffman coding is to use variable-length bit codes to represent symbols. The algorithm uses the shortest codes for the most frequently-used characters and often much longer codes for the rare symbols. This conversion from fixed-length codes to variable-length can reduce the length of the whole input string significantly.

- **Burrows-Wheeler Transform (BWT).** Variable-length coding is limited by the statistics of the data. The more uneven the frequencies of the symbols, the better the compression. It is possible to change the statistics of data by performing some kind of transform to make the data further compressible. The Burrows-Wheeler transform and other block-sorting transforms are designed to change the statistics or statistical homogeneity of the input data with this goal in mind.

  BWT rearranges the symbols of the input data in order to obtain an output data block containing long series of equal symbols, creating likely statistical inhomogeneity in the block.

- **Move-to-Front (MTF) Transform.** The goal of the MTF is to turn a series of equal symbols into a series of zeros. To achieve this, the MTF transform changes the statistics of the input string to contain series of equal symbols. This makes the string more compressible.

  The principal idea of the MTF transform is very simple. On every step the transform writes to the output stream a value that’s equal to the index of the current symbol of string in the alphabet, then reorders the
alphabet so that that symbol is first. It does this by shifting the symbols between the first symbol and the current symbol’s original position.

- **Run-Length Encoding (RLE).** RLE reduces the length of input data vectors that contain runs of zeros, or more generally runs of any number. RLE replaces a series of values with a pair of values: the value of the members of the series and the number of values in the series minus one. When encoding a run of five zeros, for example, the pair of values would be [0, 4].

- **Lempel-Ziv-77 (LZ77).** The LZ77 algorithm belongs to the family of dictionary-based algorithms. Such algorithms assume the existence of the dictionary filled with indexed strings. They achieve compression by replacing input substrings with the index of a matching string in the dictionary. The LZ77 algorithm uses a sliding dictionary that is a window of perhaps 32 KB into the input stream itself. The encoding algorithm searches the window for substrings also in the look-ahead buffer, which is much smaller, then advances the window and look-ahead buffer across the input stream. If it finds a match, the algorithm encodes the second occurrence of the string using the offset from the beginning of the first occurrence of the string and the length of the string that matches.

- **Interval Transform.** The Interval transform changes the statistics of input data by replacing the input string with sequences of intervals between matching symbols. The encoder produces a sequence of such intervals for each symbol in the alphabet. In producing these sequences, the algorithm iterates through the alphabet, marking the distances between each pair of symbols and then removing every occurrence of that symbol.

**Video Encode and Decode**

Video encoding and decoding is similar to image encoding and decoding except that video requires compression and decompression of successive frames. Video typically requires more storage than an image, but the amount typically does not scale linearly with the number of frames encoded and decoded. Video compression and decompression takes advantage of the relatively low entropy between successive frames. Two widely used standards are MPEG-2 and H.264.
MPEG-2

MPEG-2 is intended for high-quality, high-bandwidth video. It is most prominent because it is used for DVD and HDTV video compression. Computationally, good encoding is expensive but can be done in real-time by current processors. Decoding an MPEG-2 stream is relatively easy and can be done by almost any current processor or, obviously, by commercial DVD players.

MPEG-2 is a complicated format with many options. It includes seven profiles dictating aspect ratios and feature sets, four levels specifying resolution, bit rate, and frame rate, and three frame types. The bit stream code is complex and requires several tables. However, at its core are computationally complex but conceptually clear compression and decompression elements.

MPEG-2 components are very similar to those in JPEG. MPEG-2 is DCT based, and uses Huffman coding on the quantized DCT coefficients. However, the bit stream format is completely different, as are all the tables. Unlike JPEG, MPEG-2 also has a restricted, though very large, set of frame rates and sizes. But the biggest difference is the exploitation of redundancy between frames.

There are three types of frames in MPEG: I (intra) frames, P (predicted) frames, and B (bidirectional) frames. There are several consequences of frame type, but the defining characteristic is how prediction is done. Intra frames do not refer to other frames, making them suitable as key frames. They are, essentially, self-contained compressed images. By contrast, P frames are predicted by using the previous P or I frame, and B frames are predicted using the previous and next P or I frame. Individual blocks in these frames may be intra or non-intra, however.

MPEG is organized around a hierarchy of blocks, macroblocks, slices, and frames. Blocks are 8 pixels high by 8 pixels wide in a single channel. Macroblocks are a collection of blocks 16 pixels high by 16 pixels wide and contain all three channels. Depending on subsampling, a macroblock contains 6, 8, or 12 blocks. For example, a YCbCr 4:2:0 macroblock has four Y blocks, one Cb and one Cr.

The key to the effectiveness of video coding is using earlier and sometimes later frames to predict a value for each pixel. Image compression can only use a block elsewhere in the image as a base value for each pixel, but video compression can aspire to use an image of the same object. Instead of compressing pixels, which have high entropy, the video compression can
compress the differences between similar pixels, which have much lower entropy.

Objects and even backgrounds in video are not reliably stationary, however. In order to make these references to other video frames truly effective, the codec needs to account for motion between the frames. This is accomplished with motion estimation and compensation. Along with the video data, each block also has motion vectors that indicate how much that frame has moved relative to a reference image. Before taking the difference between current and reference frame, the codec shifts the reference frame by that amount. Calculating the motion vectors is called *motion estimation* and accommodating this motion is called *motion compensation*.

**H.264**

The two series of video codec nomenclature H.26x and MPEG-x overlap. MPEG-2 is named H.262 in the H.26x scheme. Likewise, another popular codec, H.264, is a subset of MPEG-4 also known as MPEG-4 Advanced Video Coding (AVC). Its intent, like that of all of MPEG-4, was to produce video compression of acceptable quality and very low bit-rate—around half of its predecessors MPEG-2 and H.263.

**Image Processing and Object Recognition**

Geometric transformations constitute a large and important segment of image processing operations. Any function that changes the size, shape, or orientation of the image or order of the pixels can be grouped under this broad classification. The math employed by these transform operations uses two coordinate systems, the source image coordinate system and the destination. Both systems have an origin (0,0) that is defined by the data pointer. The two coordinate systems are related by the geometric transform.

In most cases, the location in the source from which the data is to be drawn, indicated as \((x', y')\), does not lie exactly on a source pixel. Some form of interpolation would then be used to calculate the value. The nearest neighbor method chooses the pixel that is closest to \((x', y')\). Linear interpolation takes a weighted average of the four surrounding pixels. Cubic interpolation fits a second-order curve to the data to calculate the \((x', y')\) value. Super-sampling interpolation averages over a wider range of pixels and is suitable for resizing images to a much smaller size, such as when creating a thumbnail image.

Other common transformations are summarized as follows:
- **Resize.** Resizing functions change an image from one size to another. The pixels in the first image are either stretched by duplication or interpolation, or they are compressed by dropping or interpolation. A single resize operation can stretch the image in one direction and compress it in the other.

- **Rotation.** Turn an image around the origin, around a designated point, or around the center.

- **Affine Transform.** The affine transform is a general two-dimensional transform that preserves parallel lines and is general enough to shear, resize, or shift an image.

- **Perspective Transform.** The perspective transform is a general three-dimensional transform. Properly applied, this transform can represent a projection of an image onto a plane of arbitrary orientation.

- **Remap.** The remap function is a completely general geometric transform. It takes a destination-to-source map the same size as the destination image. Each pixel has a corresponding floating-point (x,y) coordinate pair. The operation calculates the value at that location according to the interpolation mode and sets the destination pixel to that value. The remap function is most useful in morphing or exciting video effects, for which the other geometric transforms are not flexible enough.

Image processing functions are grouped into the following categories:

- Statistics: norm, mean, median, standard deviation, histograms
- Analysis functions and filters: erode and dilate, blur, Laplace, Sobel, distance transform, pyramid
- Feature detection: edge, corner, template matching
- Motion detection and understanding: motion templates

Many of these functions are closely associated with the Open Source Computer Vision Library (OSCVL), currently available online (Intel 2003b). The histories of this library and the Intel IPP computer vision domain are intertwined, and the OSCVL uses Intel IPP as its optimization layer.

The next subsections summarize key image processing functionality.

**Edge Detection**

Perhaps the most important low-level vision task is detection of edges in the image. Edges are visual discontinuities of brightness, color, or both. They are usually detected by an automated operation on a small region of pixels. Interpreted correctly, they convey higher-level scene information, particularly the boundaries of objects in the scene.
Multi-Resolution Analysis

When trying to find an object in a scene, size is as important a characteristic as shape or color. Even if you know exactly what shape or color an object is, you need to know how many pixels wide and tall it is. One easy way of performing an analysis without knowing this size is to perform the search on multiple resolutions of an image. Such a set of resolutions of an image is often called an image pyramid.

Template Matching

In computer vision, a template is a canonical representation of an object used for finding the object in a scene. There are many ways to match the template, such as taking the pixel-by-pixel normalized sum of the squared difference between template and image.

String Processing

Processing of character strings is common to many applications and is embodied by several standard library functions across a number of operating systems. Accelerated versions of these functions exist for various platforms taking advantage of specialized hardware capabilities such as SIMD instruction sets. Typical optimized string functions include functions to perform string copy, search, string length, insertion, removal, compare, uppercase, lowercase, and concatenation.

For more information about performance libraries and embedded system architecture, please refer to the book Break Away with Intel® Atom™ Processors: A Guide to Architecture Migration by Lori Matassa and Max Domeika.

About the Authors

Lori Matassa is a Sr. Staff Platform Software Architect in Intel’s Embedded and Communications Division and holds a BS in Information Technology. She has over 25 years experience as an embedded software engineer developing software for platforms including mainframe and midrange computer system peripherals, as well as security, storage, and embedded communication devices. In recent years at Intel she has contributed to driver hardening standards for Carrier Grade Linux, and has led the software enablement of multi-core adoption and architecture migration for embedded and communication applications. Lori is a key contributor to Intel’s Embedded
Design Center, with numerous whitepapers, blogs, and industry
ccontributions on a variety of topics critical to embedded migration.

Max Domeika is an embedded software technologist in the Developer
Products Division at Intel, creating tools targeting the Intel architecture
market. Over the past 14 years, Max has held several positions at Intel in
compiler development, which include project lead for the C++ front end and
developer on the optimizer and IA-32 code generator. Max currently
provides embedded tools consulting for customers migrating to Intel
architecture. In addition, he sets strategy and product plans for future
embedded tools. Max earned a BS in Computer Science from the University
of Puget Sound, an MS in Computer Science from Clemson University, and a
MS in Management in Science & Technology from Oregon Graduate Institute.
Max is the author of Software Development for Embedded Multi-core
Systems from Elsevier. In 2008, Max was awarded an Intel Achievement
Award for innovative compiler technology that aids in architecture
migrations.

Copyright © 2010 Intel Corporation. All rights reserved.

This article is based on material found in book Break Away with Intel®
Atom™ Processors: A Guide to Architecture Migration by Lori Matassa and
Max Domeika. Visit the Intel Press web site to learn more about this book:
http://www.intel.com/intelpress/sum_ms2a.htm

No part of this publication may be reproduced, stored in a retrieval
system or transmitted in any form or by any means, electronic,
mechanical, photocopying, recording, scanning or otherwise, except as
permitted under Sections 107 or 108 of the 1976 United States
Copyright Act, without either the prior written permission of the
Publisher, or authorization through payment of the appropriate per-
copy fee to the Copyright Clearance Center, 222 Rosewood Drive,
Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests
to the Publisher for permission should be addressed to the Publisher,
Intel Press, Intel Corporation, 2111 NE 25 Avenue, JF3-330, Hillsboro,
OR 97124-5961. E-mail: intelpress@intel.com.