WHITE PAPER

Joseph Coombs,
***Applications engineering,***
*Texas Instruments*

Rahul Prabhu,
***Applications engineering,***
*Texas Instruments*

TEXAS INSTRUMENTS

# OpenCV on TI's DSP+ARM® platforms: Mitigating the challenges of porting OpenCV to embedded platforms

## Abstract

In today's advancing market, the growing performance and decreasing price of embedded processors are opening many doors for developers to design highly sophisticated solutions for different end applications. The complexities of these systems can create bottlenecks for developers in the form of longer development times, more complicated development environments and issues with application stability and quality. Developers can address these problems using sophisticated software packages such as OpenCV, but migrating this software to embedded platforms poses its own set of challenges. This paper will review how to mitigate some of these issues, including C++ implementation, memory constraints, floating-point support and opportunities to maximize performance using vendor-optimized libraries and integrated accelerators or co-processors. Finally, we will introduce a new effort by Texas Instruments (TI) to optimize vision systems by running OpenCV on the C6000™ digital signal processor (DSP) architecture. Benchmarks will show the advantage of using the DSP by comparing the performance of a DSP+ARM® system-on-chip (SoC) processor against an ARM-only device.

## Introduction

OpenCV is a free and open-source computer vision library that offers a broad range of functionality under the permissive Berkeley Software Distribution (BSD) license. The library itself is written in C++ and is also usable through C or Python language applications. Thousands of developers use OpenCV to power their own specialized applications, making it the most widely-used library of its kind. The OpenCV project is under active development, with regular updates to eliminate bugs and add new functionality. The mainline development effort targets the x86 architecture and supports acceleration via Intel's proprietary Integrated Performance Primitives (IPP) library. A recent release also added support for graphics processing unit (GPU) acceleration using NVIDIA's Compute Unified Device Architecture (CUDA) standard.

OpenCV's greatest asset is the sheer breadth of algorithms included in its standard distribution. Figure 1 on page 2 shows an incomplete list of some of the key function categories included in OpenCV. These range from low-level image filtering and transformation to sophisticated feature analysis and machine learning functionality. A complete listing of every function and use case is beyond the scope of this article, but we will consider the unique requirements of developers in the embedded vision space. For these developers, OpenCV represents an attractively comprehensive toolbox of useful, well-tested algorithms that can serve as building blocks for their own specialized applications. The question then becomes whether or not OpenCV can be used directly in their embedded systems.

Despite its original development focus for use with PC workstations, OpenCV can also be a useful tool for embedded development. There are vendor-specific libraries that offer OpenCV-like capabilities on various embedded systems, but few can match OpenCV's ubiquity in the computer vision field or the sheer breadth of its included algorithms. It should come as no surprise that OpenCV has already been ported to the ARM® architecture, a popular CPU choice for embedded processors. It's certainly possible to cross-compile the OpenCV source code as-is and use the result with embedded devices, but memory constraints and other architectural considerations may pose a problem. This white paper will examine some of the specific obstacles that must be overcome for OpenCV to achieve acceptable performance on

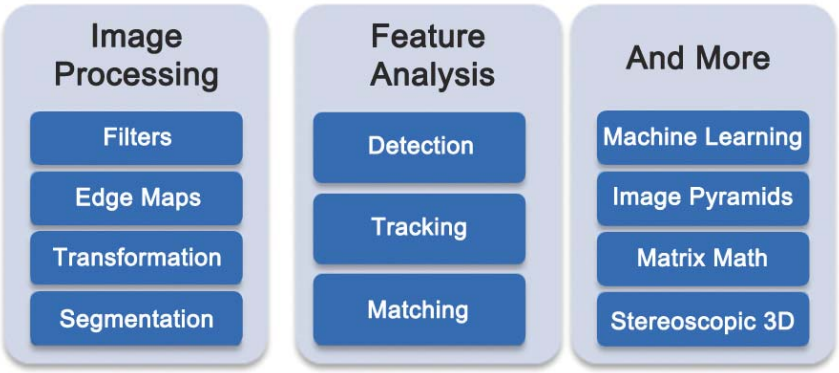


*Figure 1. Partial overview of the OpenCV library.*

an embedded platform. Finally, the paper will describe a new effort by Texas Instruments (TI) to bring OpenCV to its C6000™ digital signal processor (DSP) architecture. Performance benchmarks will compare TI's DSP+ARM® system-on-chip (SoC) processor against the standard ARM-only approach.

## Changing requirements of embedded vision applications

The continued growth of embedded vision applications places contradictory demands on embedded developers. Increasingly sophisticated vision algorithms require more memory and processing power, but price and deployment constraints require embedded devices that cost less money and consume less power. Embedded hardware and software expand in complexity while development cycles accelerate and contract. The following applications are representative of the current state and future direction of the overall embedded vision space.

Let's start with industrial vision applications. One common industrial vision task is assembly line inspection, which detects, classifies and sorts objects to maximize manufacturing speed and quality. These vision algorithms are often run on costly computer workstations; migrating to an embedded DSP is one obvious way to save on price and power consumption. Even applications that are already implemented with embedded systems can be improved by condensing discrete logic into the DSP. For example, many industrial vision systems share the basic shape illustrated by Figure 2. The image signal processor (ISP) is a field programmable gate array (FPGA) that performs time-critical pre-processing on incoming data before it reaches the DSP. This FPGA becomes more expensive and consumes more power proportional to its workload. One way to maximize the efficiency of the overall embedded system is to integrate as much pre-processing as possible into the DSP. The challenge then becomes keeping up with rapid improvements in the physical system.

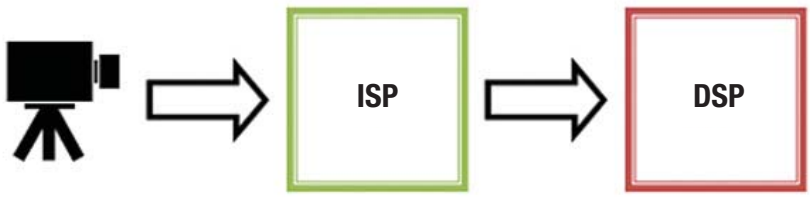


*Figure 2. Typical embedded vision system, including camera, pre-processing FPGA and DSP.*

Next-generation systems must process more data in less time to accommodate improved camera resolution and frame rate as well as faster assembly line speeds.

Video surveillance applications provide another perspective on the evolution of embedded vision. Traditional surveillance systems are less concerned with vision analytics than they are with simply encoding and recording video data. However, as vision algorithms improve, video surveillance will incorporate more automated monitoring and analysis of this recorded data. Examples range from motion and camera tamper detection to people counting and license plate reading. These algorithms enable so-called metadata streaming, or creating automated logs of detected activity to accompany streamed and recorded video data. As vision algorithms become more capable and reliable, video surveillance systems will become more automated and sophisticated. This presents a particular challenge to embedded video surveillance systems, since cutting-edge algorithms that are developed on PCs may require considerable rework and optimization to run efficiently on an embedded device. Consequently, many embedded video surveillance applications are limited to the simpler encode-and-record paradigm.

One last example application from the broad category of embedded vision is automotive vision. Unlike the previously discussed application spaces, automotive vision is almost exclusively the domain of embedded processors. Many automotive vision systems can be reduced to a block diagram similar to Figure 2, essentially consisting of a camera, a pre-processing FPGA and a DSP to apply intensive vision algorithms. Reliability is the key concern in applications such as lane departure warning, steering assistance and proximity detection. The vision algorithms used in automotive vision are under constant, active development using high-level PC software, but running the final application on a PC is simply not an option. The transition from PC to DSP is a critical step in the development of automotive vision applications. Writing and rewriting algorithms to achieve acceptable real-time performance is a major development focus. This only gets more difficult as embedded systems become more sophisticated, incorporating multiple camera inputs and multiple processing cores.

Efficient DSP software plays a critical role in all embedded vision applications. The prospect of using high-level software like OpenCV to facilitate rapid algorithm development is appealing, but optimizing that software for a new platform is a critical sticking point. Conversely, achieving acceptable performance with un-optimized DSP software is simply unrealistic. In the next section of this article, we consider the key challenges associated with porting and optimizing sophisticated PC software — particularly the OpenCV library — to run on an embedded device.

## *Challenges of porting OpenCV to embedded devices*

Since OpenCV is open source and written entirely in C/C++, the library has been cross compiled and ported as-is to a variety of platforms. However, simply rebuilding the library for an embedded platform may not yield the real-time performance demanded in that space. At the same time, rewriting and manually optimizing the entire OpenCV library for a new architecture represents an enormous amount of work. Device-appropriate optimizing compilers are critical to navigate between these opposing challenges. The ubiquitous GNU Compiler Collection (GCC) has been used to successfully port OpenCV to ARM platforms, but GCC is not available

on more specialized DSP architectures. These devices typically rely on proprietary compilers that are not as full-featured or standards-compliant as GCC. These compilers may have a strong focus on the C language and be less capable at optimizing C++ code. The current version of OpenCV relies heavily on C++ Standard Template Library (STL) containers as well as GCC and C99 extensions, which are not well-supported on certain embedded compilers. For these reasons, it may be necessary to revert to OpenCV version 1.1 or earlier — which are written almost entirely in C — when targeting a specialized embedded platform. The OpenCV source code includes many low-level optimizations for x86 processors that are not applicable to ARM® or DSP platforms. These optimizations can be replaced with vendor-provided support libraries or intrinsic functions that make explicit use of architecture-specific single instruction, multiple data (SIMD) commands to speed up code execution. OpenCV application programming interfaces (APIs) often allow data to be provided in multiple formats, which can complicate the task of optimizing these functions for a new target device. Limiting these functions to a single data type or splitting them into single-type variants can allow the compiler to generate simpler, more efficient code. Similarly, inlining small, frequently-used internal functions can provide a performance lift to high-level vision functions.

The word "optimization" for embedded platforms often means endlessly poring over low-level architectural minutiae to write and tweak device-specific assembly language code. Fortunately, as embedded processors have grown in complexity, so too have embedded development tools become more powerful and user-friendly. Most vendors in the embedded industry provide optimized libraries that have been hand tuned to provide the best performance on the device for low-level math, image and vision functionality. Coupling the OpenCV library with these libraries can accelerate high-level OpenCV APIs. TI is one of the few companies that provides vision and imaging libraries that can replace a portion of the code for an OpenCV function or, in some cases, the entire function itself. Similarly, optimized math and signal processing libraries can also provide a significant boost to maximize the potential of OpenCV functions on embedded devices. Using these optimized libraries underneath the OpenCV APIs can maximize performance by utilizing architecture-specific capabilities while maintaining the standard interface of the high-level software. In other words, these low-level libraries can accelerate OpenCV functions without breaking pre-existing application code that is written to use standard OpenCV APIs.

Another challenge often faced when using OpenCV functions in an embedded processor environment deals with the lack of native support for floating-point math. This poses a significant problem for OpenCV since it includes a number of specialized image processing functions that rely heavily on floating-point computation. OpenCV supports a wide range of image data types, including fixed- and floating-point representations. Many OpenCV image processing functions never use floating-point math, or use it only when the image data consists of floating-point values. However, some specialized functions that work with Eigen values, feature spaces, image transformation and image statistics always use floating-point math regardless of the original image data type. These intensive algorithms require native floating-point support to achieve real-time performance in an embedded application. Figure 3 on the following page compares the performance of several OpenCV functions that rely on floating-point processing across multiple embedded targets. The ARM9™

processor used lacks native floating-point support, while the ARM Cortex™-A8 processor includes NEON support for floating-point math and delivers a twofold increase in performance. Also included is TI's floating-point C674x DSP, which is highly optimized for intensive computation and delivers an even greater boost to performance. These benchmarks emphasize the need for native floating-point support when running certain OpenCV algorithms.

| Function Name | ARM9™ (ms) | ARM Cortex-A8 (ms) | C674x DSP (ms) |
|---|---|---|---|
| cvCornerEigenValsandVecs | 4746 | 2655 | 402 |
| cvGoodFeaturestoTrack | 2040 | 1234 | 268 |
| cvWarpAffine | 82 | 37 | 17 |
| cvOpticalFlowPyrLK | 9560 | 5240 | 344 |
| cvMulSpecturm | 104 | 69 | 11 |
| cvHaarDetectObject | 17500 | 8217 | 1180 |

***Figure 3. Performance benchmark for OpenCV functions with floating-point math. Image size 320×240; all cores operated at 300 MHz; ARM9 and C674x DSP cores tested using TI's OMAP-L138 C6-Integra™ DSP+ARM processor; ARM Cortex-A8 core tested using TI's DM3730 DaVinci™ digital media processor.***

Porting and running OpenCV on embedded systems also presents a more general set of design challenges. In addition to the processor architecture, there may also be memory restrictions and special requirements for deterministic, real-time operation. Multicore devices are also becoming more common in the embedded space, and utilizing these cores efficiently to maximize performance brings its own challenges. Embedded multicore devices may consist of homogeneous cores, such as dual-ARM devices, or they may integrate an ARM with a heterogeneous core such as a DSP or GPU. SoC devices also integrate peripherals and accelerators to reduce overall system complexity by simplifying board design and layout considerations. Many OpenCV functions can benefit greatly from utilizing these specialized processing cores and vector or floating-point accelerators. An algorithm that is highly parallelizable may be a good fit for an integrated GPU. Vision and image-processing algorithms that are not easily parallelized but still require intensive floating-point computation may be better suited for a DSP core. Low-level preprocessing functions like color space conversions, noise reduction and statistical computation tend to be well suited to single-purpose hardware like an FPGA or application-specific integrated circuit (ASIC). Embedded devices that allow developers to effectively split their application, including OpenCV, among the best-suited heterogeneous components can deliver superior performance.

Effectively using and sharing device memory is one of the primary challenges in embedded development. When both random-access memory (RAM) and read-only memory (ROM) are in short supply, applications must make judicious use of these resources. Many modern day applications require a full operating system (OS) with its own sizeable footprint, which makes managing device memory even more critical. An embedded vision application using OpenCV needs reasonably large memory with sufficient bandwidth and access time

to accommodate work buffers and program data for several interrelated tasks: data acquisition, processing, and storage or output of results. Moreover, OpenCV functions that operate on multi-dimensional data such as a feature space rather than the standard two- or three-dimensional image or video spaces can consume even larger blocks of memory. OpenCV developers on embedded devices must consider suitable tradeoffs between memory utilization and the full feature set of OpenCV. For example, some OpenCV APIs operate on a "memory storage" unit that is initially allocated with a fixed size and later expanded as necessary to prevent overflow as its contents grow. Developers can avoid unnecessary allocation calls and memory fragmentation by creating the initial memory storage with enough space to handle the worst-case scenario. Other tradeoffs can be made that impose limits on OpenCV APIs in order to achieve better performance without compromising computational accuracy. For example, nested image regions in OpenCV are represented as sets of components known as contours and holes. Each contour may be contained within a hole and may itself contain one or more holes, and the reverse is true for each hole. Figure 4 illustrates this relationship. OpenCV supports multiple formats to store and traverse these regions, including branched representations that require developers to write complicated routines to plot or process the overall image. Developers can achieve better performance by creating a single-branch structure that can be traversed using a simple loop. Finally, OpenCV applications may suffer from memory leaks caused by sloppy handling of large data buffers. These leaks could waste hundreds of megabytes of highly valuable RAM and could eventually crash the entire application. Memory leaks commonly arise when allocating memory and then changing the pointer itself (thereby precluding the use of "free" APIs), forgetting to free storage space after processing is complete, or carelessly changing or translating pointers inside complex data structures. Memory leaks are problematic in any system, but the consequences are particularly dire in the embedded space.

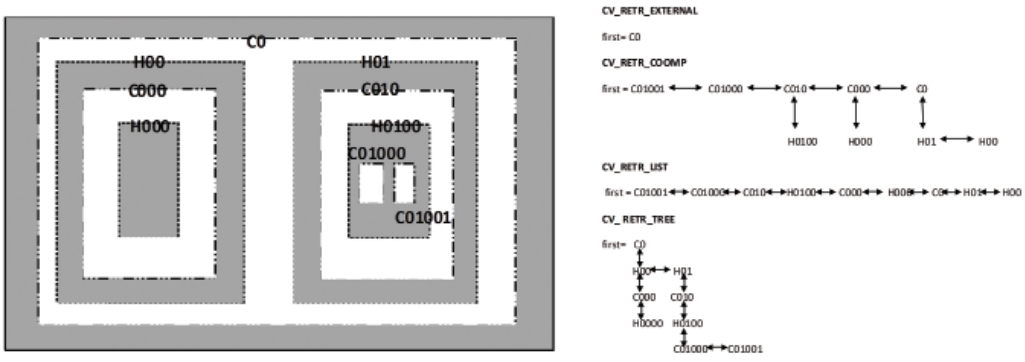


*Figure 4. Test image with contour/hole regions and tree structures supported by OpenCV.*

Multicore embedded processors provide increased performance by increasing the raw processing power available to applications, but significant challenges face embedded developers who want to use that power to accelerate OpenCV. The primary challenge when migrating to the multicore paradigm is properly partitioning the overall program and coordinating the various bits and pieces as they run independently. The simplest case is a system that consists of two separate processing units, such as two ARM cores, or an ARM and DSP. In this case, the problem is often approached as writing a normal, single-core application and then "offloading"

parts of that application to the other core. An important criterion for offloading a task from one core to the other is the inter-processor communication (IPC) overhead. Offloading a task is appropriate only if the time spent sending and receiving IPC messages does not exceed the time saved by splitting the processing load. In a multicore scenario, applications need to be multi-threaded to enable the utilization of multiple processor cores to complete a task. Multi-threaded applications need special handling to correctly coordinate their tasks and improve efficiency. However, the performance increase offered by parallelization in most vision algorithms is limited because much of the application must be executed serially. Cache coherency, address translation and endianness translation between multiple processors are some of the issues that a developer may encounter when designing a multicore application.

Certain data types in OpenCV pose a significant challenge to heterogeneous multicore systems. OpenCV defines several data types for its input/output (I/O) and processing operations that typically utilize a header/data format. Figure 5 shows a dynamic structure used by OpenCV that stores data as a simple linked list. Each list node consists of some data and pointers, or links, to neighboring list nodes. These links can be problematic when sharing lists between separate processing cores that do not share the same memory management unit (MMU). In order to share this data between the cores, pointers used by one core must be translated so that they can be understood by the other core. This address translation must then be reversed when data returns from the second core to the first. Cache coherence between the two cores is also an issue when data is passed back and forth. Additionally, internal OpenCV allocation APIs may need to be modified to ensure that data is placed in sections of memory that are equally accessible by both cores.

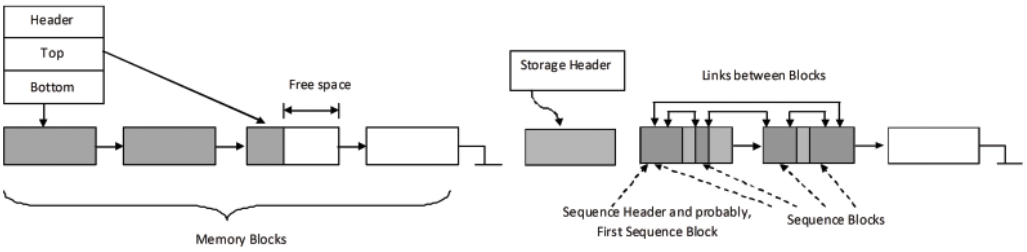


*Figure 5. Memory storage organization in OpenCV.*

In addition, OpenCV pre-allocates a memory storage in which the dynamic data structure is formed and further allocates memory if the link list outgrows the pre-allocated memory. Delegation of such a task from a master core to a slave core creates the added complication of feeding the newly allocated memory information back to the memory space of the master core. Compiler-based parallelism offered by OpenMP and application interface based task offloading offered by OpenCL are currently being evaluated for OpenCV implementation on multiple cores.

Multicore SoCs often feature heterogeneous processors that access shared external memory simultaneously. For this reason, developers using OpenCV in SoC applications must consider memory bandwidth in addition to memory capacity. Application performance depends on how quickly and efficiently memory is accessed. Simply adding more memory to a system won't always help. Direct Memory Access (DMA) adds

additional channels through which the processing cores can access external memory, which allows designers to increase bandwidth and reduce contention between cores. Through the use of enhanced DMA units, the processor does not have to directly control repetitive, low-level memory access. Figure 6 shows the performance improvement gained by using DMA to accelerate external memory access in three common image-processing algorithms. The test image is divided into slices and moved from external memory to internal RAM by DMA, processed and then copied out again by DMA. The performance using this method is much improved over processing the same image in-place in external memory.

| Function | Slice-based processing with DMA (ms) | In-place processing with cache (ms) |
|---|---|---|
| Gaussian filtering | 6.1 | 7.7 |
| Luma extraction | 3.0 | 10.9 |
| Canny edge detection | 41.4 | 79.1 |

***Figure 6. Performance benchmarks for three image-processing algorithms with and without DMA on TI's OMAP3530 DaVinci™ digital media processor at 720 MHz.***

Given the challenges inherent in bringing OpenCV to embedded devices, it is worth investigating other computer vision offerings that already exist in the embedded space. The next section of this article examines TI-provided alternatives to OpenCV. These packages are smaller than OpenCV, but they show the performance that is possible on embedded devices with highly optimized software and a deep understanding of the underlying architecture.

## *TI's other vision offerings in the embedded space*

Separate from OpenCV, TI provides optimized libraries to help developers achieve real-time performance with vision and image-processing applications on TI's embedded devices. The proprietary Vision Library (VLIB) and open source Image Library (IMGLIB) are separate collections of algorithms that are optimized for TI's C64x+™ DSPs. IMGLIB is distributed with full source code, a combination of optimized C and assembly that can be modified and rebuilt for newer DSP architectures, including C674x and C66x, to take advantage of all available architectural resources. TI also provides example application code to setup dual-buffered DMA transfers, which can speed up the image and vision kernels by 4 to 10 times compared to operating on data in external memory. These libraries are designed to convert most floating-point processing into fixed-point approximations in order to utilize SIMD extensions available in the C64x+ instruction set.

Despite the availability of these proprietary vision software offerings, OpenCV has the benefit of broad industry familiarity. Additionally, OpenCV boasts a development community actively contributing fixes and enhancements to the library, which continually improves and expands its capabilities and feature set. OpenCV has already been ported to several general-purpose processors (GPPs), including ARM®, but obtaining real-time performance often requires additional assistance from dedicated accelerators or co-processors on embedded devices. In the embedded space, DSP+ARM SoC processors and other multicore devices with high-performance, floating-point DSPs or hardware accelerators are excellent platforms to accelerate

OpenCV processing. Vision developers can utilize each core as appropriate to maximize the overall performance of their embedded system. Properly balancing processing and I/O tasks between cores can allow embedded developers to obtain real-time vision performance using OpenCV. The next section describes one effort to port and optimize OpenCV for TI's DSP+ARM® SoC processors.

## DSP acceleration of OpenCV on TI's C6-Integra™ DSP+ARM processors

TI's C6-Integra DSP+ARM processors are an attractive target for porting of OpenCV to the embedded space due to their processing capability, high level of integration and power requirements. These processors allow application developers to exploit the strengths of two embedded processor cores. The ARM runs Linux and acts as a GPP, managing I/O transactions such as video input and output and a USB-based user interface. Meanwhile, the floating-point DSP acting as a processing engine enables real-time performance for OpenCV functions. Properly utilizing the power of the DSP core presents two major challenges: coordinating basic communication between heterogeneous processing cores, and passing large data buffers from one memory space to the other. TI provides software solutions for both of these problems.

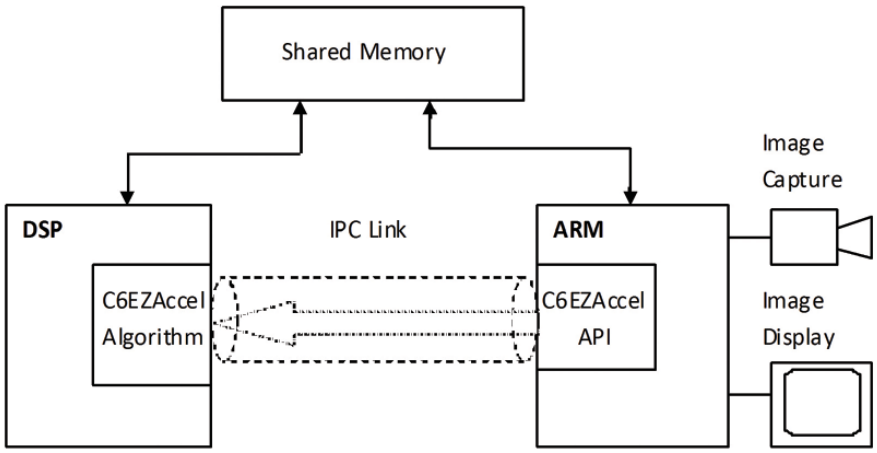


*Figure 7. High-level view of a C6-Integra DSP+ARM application using the C6EZAccel framework.*

C6EZAccel is a software development tool from TI that provides ARM-side APIs that call into optimized DSP libraries. This abstracts the low-level complexities of heterogeneous multi-core development, including IPC. The DSP side of C6EZAccel consists of an algorithm server that waits to receive messages from the ARM. Each message specifies one or more functions to be executed and provides the data buffers and configuration parameters to be used. C6EZAccel allows the ARM application to specify data using the standard OpenCV data types. Figure 7 gives a high-level view of C6EZAccel used by a C6-Integra DSP+ARM processor. The C6EZAccel tool also supports asynchronous calls to OpenCV APIs so that DSP processing can occur in parallel to other work on the ARM side. When used in asynchronous mode, C6EZAccel APIs save context information before starting DSP processing. The ARM application can then poll to check for DSP completion and use its saved context to restore data structures and pointers returning from the DSP. Figure 8 on the following page illustrates how asynchronous processing on the DSP can greatly accelerate the overall application. The DSP side algorithm links with a static OpenCV library that is built from the mainline OpenCV

source code with minimal modifications using TI's optimizing C compiler. There is a lot of room to further optimize the DSP side OpenCV library by rewriting OpenCV functions with the DSP architecture in mind, but the compiler-optimized library provides a useful starting point that developers can start exploring today.

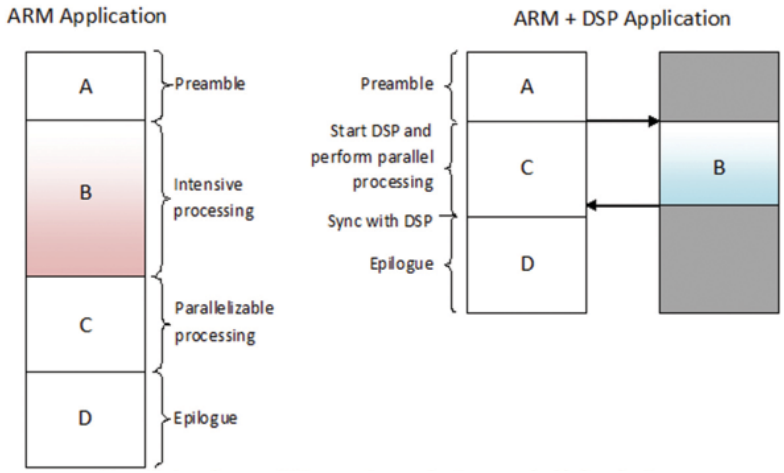


*Figure 8. Asynchronous DSP processing accelerates an embedded application.*

In order to easily call OpenCV APIs on the DSP, the ARM application also uses its own version of the OpenCV library. This library is used to load and prepare data for processing, as well as to call simple APIs that do not necessitate using the DSP. C6EZAccel also includes a custom version of OpenCV's cvAlloc function that is statically linked into the ARM application to override the default behavior and allocate contiguous data buffers using a Linux module called CMEM. This design allows the ARM application to freely share OpenCV-allocated data buffers with the DSP without modifying and rebuilding the entire ARM side OpenCV library.

Sharing OpenCV structures and data buffers between the ARM and the DSP requires two additional steps: address translation and cache management. Address translation involves converting virtual memory pointers on the ARM side to physical addresses that the DSP can interpret, then restoring the virtual address after DSP processing so that the data can be read and reused later in the ARM application. Cache management maintains data coherence between the independent ARM and DSP applications by writing back and invalidating cached memory that has been or will be modified by the other core. C6EZAccel ensures cache coherence in the ARM application by invalidating output buffers and writing back and invalidating input buffers prior to invoking the DSP side OpenCV APIs. Some OpenCV data structures require additional massaging before they can be passed on to the IPC framework; C6EZAccel takes care of this work as well. All of these tasks are handled transparently by C6EZAccel, so the ARM application looks very similar to an "ordinary" OpenCV application outside the embedded space.

The current performance of OpenCV on an ARM Cortex-A8 versus a DSP is summarized in Figure 9. Note that the DSP side OpenCV library is largely un-optimized, so there is a lot of room for future improvement. Even so, early results are promising; the DSP yields significant improvement beyond the ARM-only OpenCV library.

| OpenCV Function | ARM Cortex™-A8 with NEON (ms) | ARM Cortex-A8 with C674x DSP (ms) | Performance Improvement (cycle reduction) | Performance Improvement (x-factor) |
|---|---|---|---|---|
| cvWarpAffine | 52.2 | 41.24 | **21.0%** | **1.25** |
| cvAdaptiveThreshold | 85.029 | 33.433 | **60.68%** | **2.54** |
| cvDilate | 3.354 | 1.340 | **60.47%** | **2.50** |
| cvErode | 3.283 | 2.211 | **32.65%** | **1.48** |
| cvNormalize | 52.258 | 14.216 | **72.84%** | **3.68** |
| cvFilter2D | 36.21 | 11.838 | **67.3%** | **3.05** |
| cvDFT | 594.532 | 95.539 | **83.93%** | **6.22** |
| cvCvtColor | 16.537 | 14.09 | **14.79%** | **1.17** |
| cvMulSpectrum | 89.425 | 15.509 | **78.18%** | **5.76** |
| cvIntegral | 8.325 | 5.789 | **30.46%** | **1.44** |
| cvSmooth | 122.57 | 57.435 | **53.14%** | **2.14** |
| cvHoughLines2D | 2405.844 | 684.367 | **71.55%** | **3.52** |
| cvCornerHarris | 666.928 | 168.57 | **74.72%** | **3.91** |
| cvCornerEigenValsandVecs | 3400.336 | 1418.108 | **58.29%** | **2.40** |
| cvGoodFeaturesToTrack | 19.378 | 4.945 | **74.48%** | **4.29** |
| cvMatchTemplate | 1571.531 | 212.745 | **86.46%** | **7.43** |
| cvMatchshapes | 7.549 | 3.136 | **58.45%** | **2.48** |

***Figure 9. Performance benchmark for OpenCV functions on ARM Cortex-A8 (with NEON) versus C674x DSP. Image resolution: 640×480; ARM compiler: CS2009 (with –o3, -mfpu=neon); DSP compiler: TI CGT 7.2 (with –o3); both cores tested using TI TMSC6A816x C6-Integra™ DSP+ARM processor (ARM: 1 GHz, DSP: 800 MHz).***

## Conclusion

OpenCV is among the largest and most widely used tools in computer vision applications, and it has already started to migrate from servers and desktop PCs to the increasingly capable world of embedded devices. This paper has examined some of the key challenges faced by OpenCV in that transition, including tighter system constraints and difficulty in effectively utilizing custom embedded architectures. It has also shown the performance advantage developers can achieve by running OpenCV on a DSP compared to an ARM-only approach. Texas Instruments is currently accelerating OpenCV on its DSP and DSP+ARM platforms, offering vision developers an embedded hardware solution with high performance, high integration and low power consumption as well as a user-friendly framework with which developers can implement OpenCV. TI's support of OpenCV for its DSP and DSP+ARM platforms provides a great opportunity for embedded developers to address their performance, power and integration challenges and create a unique niche in the world of embedded vision.

## IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

| **Products** | | **Applications** | |
|---|---|---|---|
| Audio | www.ti.com/audio | Communications and Telecom | www.ti.com/communications |
| Amplifiers | amplifier.ti.com | Computers and Peripherals | www.ti.com/computers |
| Data Converters | dataconverter.ti.com | Consumer Electronics | www.ti.com/consumer-apps |
| DLP® Products | www.dlp.com | Energy and Lighting | www.ti.com/energy |
| DSP | dsp.ti.com | Industrial | www.ti.com/industrial |
| Clocks and Timers | www.ti.com/clocks | Medical | www.ti.com/medical |
| Interface | interface.ti.com | Security | www.ti.com/security |
| Logic | logic.ti.com | Space, Avionics and Defense | www.ti.com/space-avionics-defense |
| Power Mgmt | power.ti.com | Transportation and Automotive | www.ti.com/automotive |
| Microcontrollers | microcontroller.ti.com | Video and Imaging | www.ti.com/video |
| RFID | www.ti-rfid.com | Wireless | www.ti.com/wireless-apps |
| RF/IF and ZigBee® Solutions | www.ti.com/lprf | | |
| | **TI E2E Community Home Page** | | e2e.ti.com |

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265