Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms

Amir HajiRassouliha, Andrew J. Taberner, Martyn P. Nash, and Poul M. F. Nielsen

The Auckland Bioengineering Institute (ABI), The University of Auckland, Auckland, New Zealand

Abstract

Computer vision and image processing algorithms form essential components of many industrial, medical, commercial, and research-related applications. Modern imaging systems provide high resolution images at high frame rates, and are often required to perform complex computations to process image data. However, in many applications rapid processing is required, or it is important to minimise delays for analysis results. In these applications, central processing units (CPUs) are inadequate, as they cannot perform the calculations with sufficient speed. To reduce the computation time, algorithms can be implemented in hardware accelerators such as digital signal processors (DSPs), field-programmable gate arrays (FPGAs), and graphics processing units (GPUs). However, the selection of a suitable hardware accelerator for a specific application is challenging. Numerous families of DSPs, FPGAs, and GPUs are available, and the technical differences between various hardware accelerators make comparisons difficult. It is also important to know what speed can be achieved using a specific hardware accelerator for a particular algorithm, as the choice of hardware accelerator may depend on both the algorithm and the application. The technical details of hardware accelerators and their performance have been discussed in previous publications. However, there are limitations in many of these presentations, including: inadequate technical details to enable selection of a suitable hardware accelerator; comparisons of hardware accelerators at two different technological levels; and discussion of old technologies.

To address these issues, we introduce and discuss important considerations when selecting suitable hardware accelerators for computer vision and image processing tasks, and present a comprehensive review of hardware accelerators. We discuss the practical details of chip architectures, available tools and utilities, development time, and the relative advantages and disadvantages of using DSPs, FPGAs, and GPUs. We provide practical information about state-of-the-art DSPs, FPGAs, and GPUs as well as examples from the literature. Our goal is to enable developers to make a comprehensive comparison between various hardware accelerators, and to select a hardware accelerator that is most suitable for their specific application.

Keywords:
Review, Computer vision, Image processing, Digital signal processor (DSP), Field programmable gate array (FPGA), Graphics processing unit (GPU)

1. Introduction

Computer vision and image processing algorithms are used in a variety of applications in experimental mechanics [1], medical technologies [2], and human action recognition [3]. Many of the algorithms that have been used in these applications are computationally demanding, and in practical applications it is necessary to rapidly analyse the data. One of the main techniques for decreasing computation time is to use hardware with high computational power. Although the processing power of the central processing units (CPUs) in personal computers (PCs) is increasing, it remains insufficient for many applications. In addition, PCs cannot be used for computer vision tasks in mobile or portable devices. Hardware accelerators (e.g. digital signal processors (DSPs), field programmable gate arrays (FPGAs), and graphics processing units (GPUs)) are designed to address the increasing need for performing fast calculations in complicated algorithms. Furthermore, some hardware accelerators can be used in portable systems where it is not feasible to use PC-based systems.
Although DSPs, FPGAs, and GPUs have markedly different chip architectures, requiring different software development techniques, each can be used as a hardware accelerator to speed up computations. Microarchitecture and fabrication technologies are rapidly evolving, and commercial competition has motivated major hardware accelerator vendors to update and increase the capabilities of their products using the latest technological advances. However, different hardware accelerators are designed in ways that make them efficient for some algorithms but not others. Furthermore, the choice of a hardware accelerator is typically a trade-off between computational power, speed, development time, power consumption,

and price. Identifying a suitable hardware accelerator for a specific algorithm or application can be thus very challenging.
Previously published reviews have investigated different aspects of using hardware accelerators in computer vision and image processing tasks. These review papers can be divided into four main groups, which are discussed here.
In the first group of review papers, a specific algorithm or application is chosen and various hardware accelerators for that task are compared. An example is stereo vision algorithms for real-time systems, as in [4]. These review papers may help with the choice of a suitable hardware accelerator for specific applications. However, the system requirements can vary considerably for other applications or algorithms. For example, in some applications real-time execution is important (see [4]), while for other applications it may be adequate to simply increase the processing speed. The choice of a suitable hardware accelerator depends significantly on the application and the algorithm.
In the second group of reviews, specific hardware accelerators are chosen to test the performance of algorithms and their implementation. For instance, algorithm implementations for a single FPGA and a single GPU for sliding-window applications are discussed in [5]. In these hardware-oriented reviews, the fact that new technologies have many advantages over their older versions, was not considered, which does not help developers to find suitable modern hardware accelerators for their own applications. Furthermore, a specific FPGA or a specific GPU does not necessarily represent the capability of that type of hardware accelerator in general. Therefore, these review papers may not help researchers to obtain an accurate comparison between hardware accelerators, unless they decide to choose a hardware accelerator specifically from those that have been reviewed.
In the third group of reviews, a broader application is chosen and different hardware accelerators are discussed for that purpose. Some examples are: parallel computing with multicore CPUs, FPGAs, and GPUs in experimental mechanics [6]; medical image processing on GPUs [7], [8]; and medical image registration on GPUs [9] or multicore CPUs and GPUs [10]. There are also some technical details about the chip architectures in these papers. Even though these papers can provide useful information, some of them (such as [7, 8, 9, 10]) only discuss GPUs and do not cover FPGAs or DSPs. In addition, the hardware details are usually limited to a specific hardware and are of limited use for comparing different hardware accelerators.
In the fourth group of reviews, the chip architecture and software tools of hardware accelerators are discussed in detail. An example is heterogeneous computing (i.e. the combination of CPUs with FPGAs or GPUs) for general applications [11].

Even though such reviews provide useful information, there is a need to update and simplify the technical details to provide practical advice for researchers on the choice of suitable hardware accelerators for computer vision and image processing applications.
This review combines the approach of the third and fourth groups of review papers described above. Our goal was to provide sufficient information and practical examples to enable researchers to choose the most suitable hardware accelerator for computer vision and image processing applications. To this end, DSPs, FPGAs, and GPUs are discussed in separate sections, followed by examples that demonstrate the performance of the various devices in different computer vision and image processing applications.
One of the main challenges in reviewing different hardware accelerators is to provide a fair comparison. Since the model names of DSPs, FPGAs, and GPUs are not indicative of their performance, a speed normalisation factor was proposed [4] in an effort to improve the accuracy of comparison in the same chip architecture family. However, hardware accelerators are too complicated to limit the performance comparison only to the processing speed, which cannot indicate the advantage of one hardware accelerator over another, especially when they do not belong to the same family. Moreover, the processing speed of an algorithm is not only dependent on the hardware accelerator, but also on the programmers skill. In order to provide a practical comparison between hardware accelerators in this review, the most important features of DSPs, FPGAs, and GPUs for computer vision and image processing algorithms are introduced and discussed. Then, based on the technical specifications, hardware accelerators are divided into groups with similar levels of performance.
Another limitation of some review papers (such as [6]) is the discussion of outdated hardware technologies, which offer little help in assessing the performance and capabilities of modern hardware accelerators. This review addresses this issue by reporting on the latest improvements, and covers recent papers (published since 2009) with a focus on the latest hardware technologies.
This review is organised as follows. DSPs, FPGAs, and GPUs are discussed in sections 2, 3, and 4, respectively. In each section, and for each hardware accelerator, different families, available development tools and utilities, development time, and the advantages and disadvantages of using the type of hardware accelerator are discussed. Each section concludes with a separate literature review and summary, and each literature review section presents separate tables with a summary of the application, algorithms being implemented, hardware type used, and performance (or data throughput) of the algorithm. In addition, the papers be-

ing reviewed are sorted chronologically and the year of introduction of FPGAs and GPUs (as an indicator of their hardware technology level) is reported. Since FPGAs and GPUs have both been widely used in computer vision and image processing tasks, section 5 is devoted to the comparison of GPUs and FPGAs. Finally, section 6 summarises this review.

2. Digital signal processors (DSPs)

DSPs are microprocessors with an architecture that is specifically designed for performing signal processing tasks. Texas Instruments (TI) and Analog Devices (AD) are the two major companies in the DSP production market. TI-DSPs are more common in the computer vision and image processing research community than AD-DSPs, so this review focuses on TI-DSPs.
TI has designed various DSPs with different processing power ranges and capabilities for different purposes. TI-DSPs can be divided into 4 major groups [12]:

- Ultra-low power DSPs (C5000, and C55x)[13].
- Power optimised DSPs (C6000, C64x, C671x, C672x, C674x, and Open Multimedia Applications Platforms (OMAPs) [14].
- DaVinci digital media processors (DaVinci-DMPs)(DM64x, DM37x, and DM81x)[15].
- Multicore DSPs (C66x, C667x, C665x, and C647x)[16, 17].

The ultra-low power DSPs are cost-effective, but have low computational power, so are limited to simple computer vision and image processing algorithms.
Power optimised DSPs are mainly designed for portable or mobile devices where the power consumption is the most important feature. These DSPs can process algorithms with a moderate level of complexity. The main difference between the OMAP series and other members of this family is the addition of an ARM processor, which makes them a system on a chip (SoC). This ARM processor can handle interfacing with standard ports (e.g. USB and I2C), external memory modules (e.g. secure digital (SD), and multi-media cards (MMC)). As an example of their application, OMAPs are used in some mobile phones to handle digital camera, screen, and external memory interfaces.
DaVinci-DMPs are designed for multimedia applications such as video and image processing, and video capture. These DMPs include video and image hardware

codecs (e.g. MPEG, H.264, and JPEG) and hardware accelerators for video processing. Apart from a few DMPs in the DM64x series, the DaVinci-DMPs are a more advanced SoC (i.e. DSP + ARM) version of OMAPs. In comparison to power optimised DSPs and OMAPs, DaVinci-DMPs can perform more complex tasks at the expense of higher power consumption. DaVinci-DMPs also include peripherals needed for camera interfacing; hence video frame grabbers are a typical example of an application for these DSPs.
Multicore DSPs are optimised for computationally complex tasks and high performance computing (HPC). They consist of multiple DSP cores (up to 8 cores), which enable them to perform tasks in parallel. In this sense, the parallelism in multicore DSPs is similar to multicore CPUs in PCs. Multicore DSPs are designed based on two main architectures: Keystone architecture1 and Keystone architecture2 [18]. The principal difference between these two architectures is the addition of an ARM processor in the Keystone architecture2 to divide tasks between the DSP and the ARM processor. It should be considered that the reported maximum computational performance of multicore DSPs is based on the assumption that the task can be fully parallelised so that threads are executed in different cores simultaneously. However, this is often not feasible in practice, and advanced parallel programming techniques are required to optimise the computation speed of these DSPs.
The type of arithmetic computation support (i.e. fixed-point or floating-point operation) is another factor which needs to be considered in the selection of a suitable DSP. Floating-point operational support in DSPs facilitates the algorithm implementation, and increases the precision compared with fixed-point. In contrast, fixed-point operations can perform operations with fewer bits, but the programmer needs to carefully reposition the decimal point after each mathematical operation. However, fixed-point DSPs are usually cheaper than floating-point DSPs. In DSPs, arithmetic computations are performed in multiplieraccumulator (MAC) units. Table 1 summarises the fixed-/floating-point capabilities of the MAC units in various DSP series.

For the series with both fixed- and floating-point capabilities, MAC units can perform both types of calculations, but with fewer bits for floating-point operations. For instance, in a single clock cycle, the MAC units in the C66x can multiply either two 32-bit fixed-point or two 16-bit floating-point numbers.

Table 1: TI-DSP families and the fixed-/floating-point properties of their MAC units

DSP family	Fixed-/floating-point
C5000 , C55x	Fixed-point
C64x	Fixed-point
C671x, C672x	Floating-point
C674x, OMAP	Fixed- and floating-point
DM37x, DM64x	Fixed-point
DM81x	Fixed- and floating-point
C66x, C667x, C665x	Fixed- and floating-point
C647x	Fixed-point

2.1. Available development tools and utilities for DSPs

Code Composer Studio (CCS) [19] is an integrated development environment (IDE) for developing, debugging, and compiling codes in TI-DSPs. The most widely used programming language for DSPs is C/C++. Although DSPs can also be programmed in assembly language, because of its complexity, it is only used by professional programmers in developing highly optimised codes. Free libraries have been developed to assist programmers in different tasks with optimised basic functions. Some of these libraries are:

- DSP Library (DSPLIB) [20]: originally developed for single core C6000 DSPs. However, it can also be used in multicore DSPs, since the architecture of multicore DSPs is based on the C6000, and is backwardly compatible. This library includes functions for some digital signal processing tasks, such as fast Fourier transform (FFT) and convolution algorithms.

- Image Library (IMGLIB) [21]: an optimised library for image processing on C64x or C55x DSPs. IMGLIB can also be used in multicore DSPs, since the C66x family support the C64x libraries [22]. This library contains some of the basic image processing functions, such as digital image filters.

- Math Library for Floating Point Devices (MATHLIB) [23]: an optimised floating-point library that includes functions for performing basic mathematical operations, and basic vector calculations.

2.2. Embedded operating systems

SYS/BIOS (formerly DSP/BIOS) is a real-time operating system (OS) developed by TI for programming its DSPs, microcontrollers, and ARMs [24]. SYS/BIOS is specifically designed for embedded systems where synchronisation of tasks and input/output data management are important. Alternatively, DSP families that include an ARM processor can support Linux. The OS in multicore DSPs is responsible for managing the interaction between the different cores, allocating tasks to a particular core, and deciding when to pass data across cores. The SYS/BIOS multicore software development kit (MCSDK) [25] helps programmers to manage these responsibilities in multicore DSPs by providing optimised platform-specific drivers, run-time libraries (OpenMP and OpenEM), and basic network protocols [26].

To analyse and profile the code while the application is running in multi-core DSPs, a real-time tool called a multicore system analyser (MCSA) [27] is added to the MCSDK. The MCSA provides real-time performance evaluation and monitors parameters such as the computation and task loads of each core, the computation time of various parts of the code, and concurrency of tasks in different cores.

2.3. Development time

Many useful tools are available for developing and debugging codes in C/C++ for DSPs (sections 2.1 and 2.2), and available libraries include most of the general purpose algorithms for computer vision and image processing applications. In general, the development time for a simple task in single core DSPs is relatively short. In contrast, developing an optimised code using parallel programming techniques for a multicore DSP is challenging, and complex tasks require advanced programing skills. This can lead to long development times when creating optimised codes for complex computer vision and image processing algorithms in multi-core DSPs.

2.4. Advantages of using DSPs

Important advantages of using DSPs for computer vision and image processing applications include:

- The costs for developing a DSP-based portable system are typically low. DSP chips are generally cheaper than most of the other hardware accelerators for portable devices. In addition, TI provides free licenses for some versions of CCS and MCSDK with its hardware evaluation modules (EVMs), and all of the libraries introduced in sections 2.1 and 2.2 are available free of charge.

- Many computer vision and image processing applications involve sequential algorithms, and the chip architecture of DSPs is designed for the implementation of sequential tasks. Multi-core DSPs have added the capability of implementing coarse-grained parallelism for algorithms with a low-to-medium level of complexity.
- The development time for simple computer vision and image processing algorithms in single core DSPs is, in general, relatively short.
- Power consumption in DSPs is low. TI-OMAPs are specially designed for mobile applications making them a suitable option for portable devices that have limitations on their power consumption.
- DSPs are suitable for handling peripherals, standard ports (e.g. USB, and SATA), and communication protocols (e.g. TCP/IP) in portable devices.
- DaVinci-DMPs have video codec support, hence are suitable for video processing applications in portable devices.

2.5. Disadvantages of using DSPs

Disadvantages of using DSPs for computer vision and image processing applications include:

- Multicore DSPs are designed for HPC applications with a low-to-medium level of complexity. They are not suitable for high data throughput or high speed applications.
- DSPs are more suitable for sequential processing. Even though multicore DSPs are capable of performing coarse-grained parallelism, they are not a suitable choice for increasing the processing speed in massively parallel algorithms.
- It is not usually efficient to use DSPs along with CPUs in PCs for increasing the processing speed of an algorithm. DSPs and CPUs both have a similar sequential processing nature, whereas programming with CPUs is easier and more efficient than using DSPs.
- Although DSP chips are cheaper than FPGA and GPU chips for the same level of performance, TI-DSP boards are not usually cheap. A typical DSP board costs between 500 USD to 2000 USD [28], a similar price range to GPU boards. For PC-based systems that do not require external interfaces, DSP boards thus offer few advantages over GPU boards [29].

2.6. Review of applications that use DSPs

Table 2 provides a summary of some selected computer vision and image processing algorithms that have been implemented on TI-DSPs in recent literature.

As two examples of portable systems, an OMAP3530 was used for robotic applications in [30] and [31]. The DSP core of this OMAP is C64x, which is a fixed-point DSP (Table 1). The DSP core and the ARM processor (Cortex8) of C64x work at 520 MHz and 720 MHz, respectively. The energy efficiency of the system implemented in [30] was better than all previously published DSP implementations. Even though this implementation was power efficient, the maximum possible frame rate of the real-time application with this system was only 8 fps for an image size of 640 pixel $\times$ 480 pixel (Table 2).

DaVinci-DMPs have been used in [32], [33], and [34] for the implementation of an image processing system with a low level of complexity (details are in Table 2). DM642 ([32, 33]) and DM648 ([34]) are both fixed-point DSPs (Table 1) without an ARM processor. The maximum processing clock rates for the DSP cores of the DM642 and DM648 are 720 MHz and 1.1 GHz, respectively. In [32] and [33], the DM642 was used in a video processing application, for which the DaVinci-DMPs were specifically designed.

The C6416, which is a power optimised fixed-point DSP (Table 2), was used in [35] to implement a simple stereo vision tracking algorithm based on the sum of absolute differences (SAD). This system included a CMOS camera, and was designed to be used in stand-alone portable devices or robots.

The C6678, which is a multicore DSP with 8 cores (each core operates at 1 GHz), is able to perform both fixed-point and floating-point operations, and was used for rather complicated tasks in [36] and [37]. In [36], a motion estimation algorithm was implemented and, even though the C6678 is among the highest-performing DSPs available, the authors could not achieve high frame rates for large image sizes [36]. For an image size of 128 pixel $\times$ 128 pixel the maximum frame rate they could achieve was 9.79 fps, which outperformed the Intel Xeon CPU when one core was used (3.99 fps), but not when all 4 cores (8 threads) were used (20.91 fps). Nevertheless, the C6678 showed considerably lower power consumption compared with the CPU (single core and multiple core implementations). An image registration implementation for embedded systems was proposed in [37]. The authors combined four C6678 DSP boards to increase the processing power of their system, and achieved 10.75 fps for an image size of 4096 pixel $\times$ 4096 pixel (a processing time of 93 ms for each image). Advantages of this system include being cost-effective, energy-efficient, and suitable for embedded systems. Nevertheless, the algorithm they implemented was a rigid image registration algorithm

Table 2: A summary of some selected computer vision and image processing algorithms implemented on TI-DSPs in the recent literature

Application(s)	Algorithm(s) Implemented	Hardware (DSP)	Performance/ Data throughput
Stereo vision system for small robots (2011) [30]	• Sum of absolute differences (SAD) over a 7 pixel × 7 pixel window	OMAP3530 (Power optimised)	8 fps for 640 pixel × 480 pixel images and 60 levels of disparity (i.e. 147 M disparities/s)
Night-time vehicle detection (2011) [32]	• Bright object segmentation based on the image histogram • Spatial clustering • Feature-based identification and tracking	DM642 (DaVinci-DMPs)	The implementation was tested with real highway images, but no performance was reported
Stereo vision system for tracking of moving objects (2011) [35]	• Image feature extraction (Colour interpolation, brightness compensation, conversion to the grey scale) • SAD algorithm for finding the disparity map • Edge stereo matching	C6416 (Power optimised)	8.92 fps for 356 pixel × 292 pixel images of a stereo pair camera (the processing time was approximately 112 ms for each image)
Guidance, navigation and control for UAV landing (2012) [33]	• Image feature extraction • Image matching and identification	DM642 (DaVinci-DMPs)	No performance or data throughput was reported
Motion estimation (2013) [36]	• Multi-channel gradient model	C6678 (Multicore)	9.75 fps for a video with 128 pixel × 128 pixel resolution
2D to 3D conversion based on disparity map estimation (2014) [34]	• Discrete wavelet transform (DWT) • Edge detection • Disparity map estimation • Colour segmentation using K-means clustering • Adaptive filtering	DM648 (DaVinci-DMPs)	8.83 fps for 1390 pixel × 1110 pixel images of a stereo pair (The average processing time was 113.25 s for each image)
Image registration (2014) [37]	• Multilevel Gauss-Newton minimisation for the rigid alignment of two images based on the SSD of the image intensities	4 × C6678 (Multicore)	The algorithm could register two 4096 pixel × 4096 pixel images at maximum 10.75 fps (93 ms for each pair)
Real-time image processing for robots (2014) [31]	• Finding landmarks and position estimation	OMAP3530 (Power optimised)	The algorithm could estimate the position from 576 pixel × 720 pixel images

with moderate complexity, consisting of an optimisation process for minimising the sum of square differences (SSD) of the image intensity values (Table 2).

2.7. Summary and conclusion for DSPs

In this section, DSPs and their capabilities have been introduced and their pros and cons have been presented. Examples of their application in the literature indicate that they are not particularly suitable for high-performance applications. Despite recent advances in TI-multicore DSPs [38], it still is not feasible to implement complex computer vision and image processing algorithms on DSPs, especially when the data throughput requirement is high. Furthermore, DSPs are not particularly suitable for PC-based systems since, apart from external interfaces, they do not provide significant advantages over GPUs. In contrast, DSPs are an energy-efficient and cost-effective solution for embedded systems, and for mobile or portable devices in which the computational demands are not high, and the power consumption level is critical.

3. Field-programmable gate arrays (FPGAs)

The FPGA chip incorporates arrays of reprogrammable logic gates. As opposed to CPUs, DSPs, and GPUs, FPGA fabrics do not have a pre-structured chip architecture or a central processing unit. Thus, prior to programming the reconfigurable FPGAs, the programmer should design a hardware architecture for their specific application using the logic gates inside the FPGA.
The FPGA hardware architecture is configured by interconnecting FPGA logic gates to perform a specific task, and requires reconfiguration for each new algorithm. FPGAs are thus often referred to as reconfigurable devices. The programming languages for FPGAs are quite different from those for CPUs, DSPs, and GPUs. Hardware description languages (HDLs), such as VHDL [39], and Verilog [40] are the most common programming languages for configuring FPGAs. However, HDLs are low-level and complicated programming languages, particularly for beginners. To simplify the programming of FPGAs, some high-level programming languages, such as C-like languages (e.g. SystemC [41]), and domain-specific languages (DSLs) [42, 43, 44, 45] have been developed for FPGAs. Nevertheless, C-like languages are not particularly suitable for non-sequential algorithms, and DSLs are only suitable for a limited range of tasks. In general, developing efficient algorithms in FPGAs requires a good understanding of hardware-

Table 3: Xilinx FPGA families, their process technology, and their year of introduction

FPGA Family	Process Technology (Year of Introduction)
Spartan-2	180 nm (2000)
Virtex-2	150 nm (2001)
Virtex-2 pro	130 nm (2002)
Virtex-2 pro x	130nm (2003)
Virtex-4, Spartan-3E	90 nm (2005)
Virtex-5	65 nm (2006)
Spartan-3A	90 nm (2007)
Virtex-5 FXT	65 nm (2008)
Virtex-6, Spartan-6	40 nm (2009)
Virtex-7, Kintex, Artix, Zynq	28 nm (2010)
UltraScale (Virtex, Kintex, Zynq)	20 nm (2013)
UltraScale+ (Virtex, Kintex, Zynq)	16 nm (2013)

level details.
Xilinx and Altera are the two main vendors of FPGAs currently being used by the computer vision and image processing research community. These FPGA families are covered in the following two sections.

3.1. Xilinx FPGA families

Table 3 provides a summary of Xilinx FPGA families, their fabrication process technology, and their year of introduction. The Spartan series are low-cost FPGAs which are designed for relatively simple applications. The Virtex series are specifically designed for performing signal processing tasks, and are relatively expensive compared with other FPGA families. The Kintex and Artix series are low-performance and inexpensive versions of the Virtex-7. Zynq series are medium to high performance SoCs, which include an FPGA (Artix or Kintex) and an ARM processor to handle sequential tasks, memory interface, and standard input/output ports [46].

The UltraScale and UltraScale+ families are the latest Xilinx technologies.

UltraScale and UltraScale+ are manufactured using 20 nm and 16 nm planar fabrication processes, respectively. The Virtex UltraScale+ FPGA is Xilinxs highest performance FPGA, and is designed for high-performance and high-speed applications. In general, the Spartan and Artix families have low to medium performance, whereas the Virtex and Kintex families can be used for algorithms with medium to high levels of computational complexity. The detailed specifications of Xilinx FPGAs for the Virtex-5, Virtex-6, 7-series (including Virtex-7, Kintex 7, and Artix 7), and UltraScale series are available in [47], [48], [49], and [50], respectively.
Important features for the selection of a suitable FPGA for computer vision and image processing tasks are summarised below:

- **Configurable logic blocks (CLBs)** are hardware resources and logic gates for algorithm implementation in Xilinx FPGAs. The logic cells of CLBs can be connected to each other to form large shift registers (SRs), multiplexers (MUXs), look-up tables (LUTs), and distributed memories. The interconnection of CLBs forms a unified logic circuit, which is the hardware representation of the algorithm. FPGAs that have large numbers of CLBs are a suitable choice for complex algorithms. However, the type of logic gates, and the number of input bits vary across the different FPGA families. Because of this, the number of CLBs in an FPGA is not a useful comparison metric.

- **DSP slices** are designed for robust implementation of basic mathematical operations and signal processing tasks. These blocks are called DSP48 [51] in Xilinx FPGAs, and can have different numbers of bits across the various Xilinx families. DSP48 blocks were first introduced in Virtex-2 families, and have been further developed in later Virtex families.

- **DSP performance** is quantified by the maximum number of mathematical operations that a single DSP slice of Xilinx FPGAs (DSP48) can perform per second. DSP performance can be the main bottleneck when dealing with high throughput data in real-time applications.

- **Block RAMs** are designed with SRAM architecture for the storage or buffering of data, and are especially important for storing image data inside the FPGA chip. The storage demands of image processing or computer vision algorithms, and the available block RAMs in an FPGA, should be considered when choosing an appropriate option.

3.1.1. Tools and utilities for Xilinx FPGAs

There are several tools to facilitate the development of code in Xilinx FPGAs. These tools, introduced by Xilinx or third parties, are discussed in this section.
The Vivado design suite [52] is the main tool for configuring Xilinx FPGAs. Even though Vivado was introduced for 7-series FPGAs (i.e. Virtex-7, Kintex, Artix, and Zynq) and UltraScale families, it can be used to configure other Xilinx FPGAs. Vivado has replaced the Xilinx integrated synthesis environment (ISE) design suite [53], which was the tool formerly provided by Xilinx for configuring their FPGAs. Vivado is able to synthesise codes faster than Xilinx ISE using a new algorithm for configuring the FPGAs [54]. Vivado also includes a high-level synthesis (HLS) tool for C-based IP generations in a high-level language (C, C++, or SystemC). The Xilinx HLS tool was demonstrated to be faster than HDLs (i.e. VHDL, Verilog) for developing optimised codes for sequential algorithms [55].
Vivado is not a free tool, but Xilinx offers a free limited edition of Vivado called WebPack edition, and provides a free licence with its own FPGA boards. The Vivado WebPack does not support some of the Xilinx FPGA families, and does not have some of the features of the full version Vivado (such as Xilinx SysGen).
Xilinx intellectual property cores (IP-cores) [56] are optimised hardware-implemented algorithms for performing various tasks for a wide range of applications. Most of the Xilinx IP-cores are not available free of charge, but the basic algorithms are included in the Xilinx ISE licence. Third party IP-cores are also available for specialised algorithms, which can be purchased separately. IP-cores cover most of the fundamental functions, and can help to significantly reduce the development time of computer vision and image processing algorithms in FPGAs. The drawback is that purchasing IP-cores increases the cost of the project.
The Xilinx embedded development kit (EDK) [57] is a tool developed for designing and programming embedded processors inside the FPGA chip. Embedded processors have two main categories in Xilinx FPGAs, named Microblaze and PowerPC. Microblaze can be designed and added to an FPGA using the available logic cells of the FPGA (i.e. it is a soft processor), while PowerPC is a pre-built processor designed by Xilinx (i.e. it is a hard processor). PowerPCs are based on reduced instruction set computer (RISC) technology, and are only available in some of the Xilinx FPGA families. Xilinx embedded processors are programmed using the C/C++ language, and are suitable for sequential tasks or handling external interfaces (such as DDR SDRAMs). Xilinx embedded processor cores are inside the FPGA chip. It is thus simple to develop a data transfer interface between them and the rest of the code, and it is possible to achieve higher speeds compared

with external interfaces. In fact, Xilinx embedded processor cores were developed to add the sequential processing power of these processors to the parallel processing of the FPGA. However, the clock rate of Xilinx embedded processors is only in the order of hundreds of MHz, hence are unsuitable for computationally expensive algorithms. The ARM processors in Zynq and Zynq UltraScale+ have a considerably higher performance compared to Microblaze and PowerPC. However, neither Xilinx embedded processors nor ARMs can cope well with complicated algorithms, and FPGAs are still not particularly suitable for heavily sequential tasks.

Xilinx system generator (SysGen) [58] is a tool designed to simplify the implementation of digital signal and image processing algorithms. It is a high-level tool for designing high-performance systems using FPGAs in MATLABs Simulink environment. Xilinx SysGen can be used to develop efficient codes for Xilinx FPGAs, and can help to significantly reduce the development time for complicated modular-based signal processing algorithms. FPGA basic logic cells and Xilinx IP-cores are accessible within Xilinx SysGen in a modular and block-based format. One of the advantages of Xilinx SysGen is its ability to use the software blocks of MATLABs Simulink to test and debug the code implemented in hardware blocks of the FPGA (i.e. software and hardware co-simulation). This feature of Xilinx SysGen can significantly reduce the test and debug time compared with only using HDLs for developing codes.

Xilinx Integrated Logic Analyzer (ILA) IP core [59] is a software/hardware package for online and real-time debugging of FPGA codes. Xilinx ILA requires its special hardware to directly capture real-time data from the FPGA. The captured data is stored in a buffer inside the FPGA during runtime, and is transferred to the PC via a USB port for debugging with the Xilinx ILA software. Even though this tool assists with the debugging process, the drawback is the need to have sufficient free memory space inside the FPGA for buffering the data. SDAccel [60] is the Xilinx development environment for OpenCL. OpenCL is an open standard, which is maintained by a technology consortium called the Khronos Group [61]. The OpenCL development environment facilitates the development of C-based high-level codes for FPGAs. This tool can be used to develop and emulate kernels in the host PC, debug them and generate an implementation report for the FPGA, and then convert that kernel to an FPGA code. This tool helps programmers to decrease the development time for parallelisable algorithms by writing the code and debugging it in a high-level language. For instance, the PCIe interface can be configured with OpenCL to transfer data to an FPGA from the host PC.

In addition to official Xilinx tools, some third party graphical language tools are

Table 4: Altera FPGA families, their process technology, and their year of introduction

FPGA Family	Process Technology (Year of Introduction)
Stratix, Cylone	130 nm (2002)
Stratix-GX	130 nm (2003)
Stratix-2, Cylone-2	90 nm (2004)
Stratix-2 GX	90 nm (2005)
Stratix-3	65 nm (2006)
Cylone-3, Arria-GX	65 nm (2007)
Stratix-4	40-nm (2008)
Cylone-4, Arria-2	60 nm (2009)
Stratix-5, Arria-2-GZ	28 nm (2010)
Cylone-5, Arria-5	28 nm (2011)
Arria-10	20 nm (2013)
Stratix-10	14 nm (2016)

also available for programming Xilinx FPGAs. The National Instruments (NI) LabView FPGA module [62] is one of the most popular. These graphical languages may generate inefficient codes if the programmer is inexperienced in the use of high-level modules. To help develop efficient FPGA codes in NI LabView, Xilinx IP-cores were added to the versions since 2014. The main disadvantage of the NI LabView FPGA module is its limitation to NI FPGA boards. NI FPGA boards usually require special hardware accessories, such as the NI chassis (further details are described in [63]), and are considerably more expensive than similar Xilinx FPGA boards.

3.2. Altera FPGA families

Altera FPGAs comprise three main families: Stratix [64]; Arria [65]; and Cyclone [66]. Table 4 shows the Altera FPGAs, their process technology, and their year of introduction. Among them, the Stratix series is designed for medium to high performance algorithms, and is similar to the Virtex series in Xilinx FPGAs. The Altera Stratix-10 is a high-performance FPGA that has twice the performance and 70% lower power consumption compared to Stratix-5 [67]. Comparing Table 3 and Table 4 shows that the Xilinx and Altera FPGAs of the same class of performance usually use the same fabrication process technology.

Table 5: Corresponding tools for developing codes in Altera and Xilinx FPGAs

Altera FPGAs	Xilinx FPGAs
Altera Quartus II	Xilinx Vivado, Xilinx ISE
Altera Nios II EDS	Xilinx EDK
Altera DSP builder	Xilinx System Generator
Altera OpenCL SDK	Xilinx SDAccel

Altera Stratix FPGAs have dedicated hardware blocks for performing robust mathematical and logical operations [68]. These blocks have similar functionality to DSP slices (DSP48) in Xilinx Virtex FPGAs, and can increase the speed and performance of signal processing algorithms.

3.2.1. Tools and utilities for Altera FPGAs

Altera has introduced some tools to facilitate the development of code in its FPGAs, similar to those previously discussed in section 3.1.1 for Xilinx FPGAs. These software tools are Quartus II for developing and compiling FPGA codes [69], the Nios II embedded design suite (EDS) for embedded software development [70], DSP builder for developing FPGA signal processing algorithms in MATLABs Simulink environment [71], and an SDK for OpenCL programming [72]. Table 5 lists Alteras tools for developing FPGA codes, and the corresponding tools for Xilinx FPGAs.

Altera and third parties have developed IP-cores for performing various tasks in Altera FPGAs. These IP-cores can simplify the implementation of computer vision and image processing algorithms. In addition, some open-source libraries are available in OpenCL for Altera FPGAs.

3.3. Development time

The development time for FPGAs is longer than for other hardware accelerators. Even though recent offerings in software tools were intended to reduce the long development time in FPGAs, developing robust codes in FPGAs is still challenging. Parallel programming in FPGAs requires skilled programmers with sufficient technical knowledge of the FPGA hardware details. High-level functions are not available for FPGAs without payment. Hence algorithms must often be developed using basic functions, which is a time-consuming process. For instance, in a study in 2012, it took 12 months for two postdoctoral employees to

implement the algorithms for extracting dense optical flow, and image features in a Virtex-5 FPGA [73].
Nevertheless, it is expected that the development time of FPGAs will reduce with the emergence of the new generation of HDLs, such as Bluespec System Verilog (BSV) [74] and Chisel [75].

3.4. Advantages of using FPGAs

Important advantages of using FPGAs for computer vision and image processing algorithms include:

- The processing speed of FPGAs is higher than other hardware accelerators. Recent FPGAs from both Xilinx and Altera can process billions of operations per second in parallel with their DSP blocks ([76] and [77], respectively). Such speeds have not been matched by any other hardware accelerator.

- FPGAs can have high data throughput, hence are good choices for data capture cards (e.g. video capture cards).

- The parallel and reconfigurable nature of the FPGAs is a useful feature, enabling FPGA hardware to be designed and configured for high performance applications.

- FPGAs are relatively energy efficient. In many studies, such as [73], it has been shown that FPGAs have the highest processing power when normalised for power consumption.

- Because of their low power demands, FPGAs are good candidates for use in portable devices.

- FPGAs have industrial and military grades for use in harsh working conditions.

- Programmers can implement flexible and efficient algorithms in FPGAs by reconfiguring the FPGA hardware optimised for the algorithm.

- FPGA codes can be adapted for use in application-specific integrated circuits (ASICs). ASICs reduce mass production costs.

- Using the PCIe interface for data communication between FPGA boards and the host PC has been significantly simplified by open source packages, such as RIFFA [78], ThreadPoolComposer [79], and JetStream [80], [81].

3.5. Disadvantages of using FPGAs

Important disadvantages of using FPGAs for computer vision and image processing algorithms include:

- The long development time of FPGAs is their main drawback.
- It is challenging to develop efficient codes in FPGAs. Even though new tools have been developed for FPGAs to make this process simpler and faster, the programmer still needs sufficient technical skills to develop robust codes in FPGAs.

3.6. Review of applications that use FPGAs

In the first part of this section, the use of FPGAs in stereo vision systems is discussed. In the second part, the use of FPGAs in other image processing and computer vision applications is reviewed.

3.6.1. FPGAs in stereo vision systems

The most common application of FPGAs in stereo vision systems is the implementation and optimisation of stereo-correspondence algorithms. In 2013, Tippetts et al. published a comprehensive review of various stereo vision algorithms, and their suitability for resource-limited systems (including FPGAs) [4]. In their review, the algorithms were evaluated based on accuracy and speed. Despite this being a common way of comparing algorithms, the accuracy and speed of an algorithm is dependent on the programming language, programmer skills, and the type of hardware being used. Therefore, such comparisons are not precise enough to show the suitability of the hardware accelerator itself. To compare different algorithms in FPGAs, Tippetts et al. [4] reviewed 12 papers, of which the highest rate of disparities for stereo reconstruction was reported for the implementation of Ambrosch et al. [82]. However, they reported Quartus II as the FPGA of the implementation of Ambrosch et al., whereas in fact Quartus II is not an FPGA, but Alteras software tool for developing codes (Table 5). Ambrosch et al. simulated their algorithm for an Altera Stratix-2 FPGA, and could achieve 10,108 M disparities/s [82] (reported to be 6,062.9 disparities/s in Tippetts et al.s review paper [4]).

Stereo correspondence algorithms and their implementation for FPGAs were evaluated by Colodro-Conde et al. [83] in 2014. The algorithms involved a trade-off between the hardware resources they used, and the speed they could reach. Colodro-Conde et al. [83] concluded that SAD can be easily implemented in FPGAs due to its highly parallelised nature and relatively straightforward implementation. However, SAD is not an accurate method for measuring depth data. Among the stereo correspondence algorithms, global matching algorithms achieve good accuracy scores in standard benchmarks, such as KITTI [84] and Middlebury [85].
Table 6 provides a summary of some of the high performance FPGA implementations for stereo vision algorithms in recent literature. The most common algorithm for stereo vision systems in these papers was SAD, which is in accordance with its simple implementation in FPGAs [83]. However, details of the implemented stereo matching algorithms differ between these papers, hindering direct comparison of the performance of these FPGAs. Semi-global matching algorithms were implemented by Banz et al. [86] on a Virtex-5, Hofmann et al. [87] on a Zynq-7000 and a Virtex-7, and Zha et al. [88] on a Kintex 7. The performance of each implementation is illustrated in Table 6.

3.6.2. FPGA in non-stereo computer vision and image processing applications

Table 7 provides a summary of some selected FPGA implementations for some non-stereo computer vision and image processing algorithms in the recent literature. Not all of these papers reported a speedup ratio. However, for those in which it was reported, the algorithms implemented in Virtex-6, or Virtex-7 FPGAs could achieve considerably higher speedup ratios, compared to those in Virtex-2, or Virtex-4 FPGAs. Among the algorithms in Table 7, the complexity was substantially greater for the algorithms implemented in a Virtex-6 [89], [90], a Virtex-7 [91], [92], or a Stratix-5 [93]. Even though Xilinx Virtex and Altera Stratix are high-performance FPGA series, they are not the only options suitable for computer vision and image processing algorithms. For instance, Table 6 illustrates some examples of the use of lower performance FPGAs, such as Xilinx Zynq-7000 for relatively complicated algorithms [87], [94].
As an example of using embedded processor cores, Microblaze was used in [95] to handle the external interfaces and manage the partial reconfiguration capability of the code (i.e. the reconfiguration of some parts of the code while the code is running). To shorten the development time, Xilinx SysGen was used in [96] and [97] to implement modular-based signal or image processing algorithms.

Table 6: A summary of some selected FPGA implementation of stereo vision algorithms in the recent literature

Application(s)	Algorithm(s) Implemented	Hardware (FPGA)	Performance/Data throughput
Real-time stereo vision (2010) [86]	• Noise reduction • Rectification • Semi-global matching disparity estimation • Rendering for visual inspection	Xilinx (Virtex-5)	4050M disparities/s (i.e. 103 fps for 640 pixel × 480 pixel images and 128 disparity level)
Real-time stereo vision system (2010) [99]	• Image rectification • Stereo matching (based on census transform, and correlation) • Post-processing of the disparity map	Xilinx (Virtex-4)	4522 M disparities/s (i.e. 230 fps for 640 pixel × 480 pixel images and 64 disparity level)
Disparity map computation (2010) [100]	• SAD • Pyramid-based zero-mean normalised crossed-correlation (ZNCC)	Altera (Stratix-4)	7864 M disparities/s (i.e. 320 fps for 640 pixel × 480 pixel images and 80 disparity level)
Low-cost FPGA stereo vision system (2012) [101]	• Lens distortion removal (for radial and tangential distortion) • Image rectification • Minimum SAD for stereo matching • Removal of the unreliable matches	Xilinx (Virtex-4)	8986 M disparities/s (i.e. 325 fps for 640 pixel × 480 pixel images and 90 disparity level)
Stereo-vision system (2013) [102]	• Image rectification of stereo images using a lookup table; • Stereo matching using symmetric dynamic programming	Altera (Stratix-3)	3020M disparities/s (i.e. 30 fps for 1024 pixel × 768 pixel images and 128 disparity level)
Edge-directed real-time disparity map computation (2013) [103]	• Sobel edge detection (consisted of a convolution unit with the Sobel horizontal and vertical kernels); • SAD computation over an 11 pixel × 11 pixel window	Xilinx (Virtex-5)	7864 M disparities/s (i.e. 50 fps for 1280 pixel × 1024 pixel images and 120 disparity level)
Stereo Matching (2014) [104]	• Guided Image Filtering (GIF)	Xilinx (Kintex-7)	3538 M disparities/s (i.e. 60 fps for 1280 pixel × 720 pixel images and 60 disparity level)
Semi-global matching for real-time stereo vision (2016) [87]	• Semi-global matching disparity estimation • 3 pixel × 3 pixel median filter	1) Xilinx (Zynq-7000) 2) Xilinx (Virtex-7)	1) 3775 M disparities/s (i.e. 32 fps for 1280 pixel × 720 pixel images and 128 disparity level) 2) 5308 M disparities/s (i.e. 45 fps for 1280 pixel × 720 pixel images and 128 disparity level)
Global stereo-matching (2016) [88]	• global stereo matching • block-based cross tree	Xilinx (Kintex 7)	5806 M disparities/s (i.e. 30 fps for 1920 pixel × 1680 pixel images and 60 disparity level)
Real-Time stereo vision (2016) [94]	• DNA sequence alignment	Xilinx (Zynq-7000)	1769 M disparities/s (i.e. 30 fps for 1280 pixel × 720 pixel images and 64 disparity level)

3.7. Summary and conclusion for FPGAs

FPGAs are the most flexible hardware accelerators for the implementation of customised computer vision and image processing algorithms. However, good knowledge in digital logic design, hardware architecture, HDLs, and programming tools are essential for implementing efficient complex algorithms in FPGAs. Most of the computer vision and image processing algorithms are designed for sequential processors. Hence, achieving an acceptable performance in FPGAs is only possible when the algorithm is modified and optimised for parallel processing. For instance, the standard sequential algorithms for corner detection and frontal face detection were modified to be optimised for FPGA implementation by Lim et al. [98]. As a result, the optimised code could be implemented in an Altera Cylone-4 FPGA (a relatively inexpensive, low-end FPGA with few hardware resources). Lim et al. in [98] showed that even though they had used a low-end FPGA, their optimised algorithm could achieve a similar or higher speed compared with similar non-optimised algorithms implemented in high-end FPGAs. In summary, some applications for which FPGAs are suitable options include:

- FPGAs are the best option for algorithms with high computational demands in a portable PC-independent device. FPGAs are low-power, can be used in embedded systems, and are designed for high performance tasks.
- For designs that will be mass produced, FPGAs are suitable options, since an ASIC can easily be designed and produced based on an FPGA design. ASICs substantially reduce the costs for mass production.
- Because of their high data throughput, FPGAs are the most suitable option for capturing and processing high-frame-rate data from high-speed cameras. The image data can be processed in the FPGA at a high speed.

FPGAs chosen from the latest technologies (i.e. Virtex-6, and Virtex-7 in Table 7) showed a good performance. For example, an impressive speedup of 11507-fold was achieved with a Virtex-7 in [91]. However, FPGAs are expensive, and development times using traditional methods are usually extensive.

Table 7: A summary of some selected FPGA implementation of non-stereo computer vision and image processing algorithms in the recent literature

Application(s)	Algorithm(s) Implemented	Hardware (FPGA)	Performance/Data throughput
Real-time 3D surface model reconstruction from Integral Images (2010) [115]	• Estimation of the pixel distances, and finding their minimum using SAD	Xilinx (Virtex-5)	The proposed architecture was able to process 3D data at 34 images/s for image size of 2048 pixel × 2048 pixel
Roadway path extraction and tracking (2010) [96]	• Pre-processing (including Gaussian noise reduction, histogram stretching, and morphological operations); • Model fitting (modified version of RANSAC algorithm); • Model tracking	Xilinx (Spartan-3)	The implemented algorithm was able to process video sequences at 30 frames/s
Digital hologram generator (2010) [116]	• Modified computer-generated hologram (CGH) algorithm with CGH kernels and cells	Xilinx (Virtex-2)	The implemented algorithm could generate one frame of the CGH with a size of 1408 pixel × 1050 pixel and 10,000 light sources in 0.0093 s
Atomistic magnetic spin simulations (2011) [117]	• Ising Model with a Monte Carlo update (The algorithm includes parallel convolutions units, delays and LUTs).	Xilinx (Spartan-3)	The FPGA implementation was faster than an Intel Xeon X5560 CPU at a clock rate of 2.80 GHz
Run-time self-reconfigurable 2D convolver (2011) [95]	• 2D convolution	Xilinx (Virtex-4)	The implementation was able to achieve speedup of 4× to 20× in comparison to an Intel Core2Duo CPU for the application of 2D convolution in edge detection, noise filtering, binarization, and smoothing of the images
High-speed face detection (2012) [118]	• Symmetric image downscaling; • Classifier sharing; • Cascade merging	Xilinx (Virtex-5)	307 frames/s for image size of 640 pixel × 480 pixel
Image rectification for stereo vision (2013) [119]	• Mapping between the distorted and undistorted image; • Pixel reconstruction (i.e. using the mapping function and interpolation to find the new pixel positions)	Xilinx (Virtex-4)	367 fps for the image size of 640 pixel × 480 pixel, and 120 fps for the image size of 1280 pixel × 720 pixel
2D cross-correlation for real-time surface tracking (2013) [97]	• Hierarchical 2D cross-correlation	Xilinx (Virtex-6)	Simulations showed approximately speedup of 200-fold in the processing time in comparison to an Intel Xeon CPU, and an ordinary laptops GPU
Geometric Algebra in colour Edge Detection (2013) [120]	• Edge detection in 3D using rotors in geometric algebra	Xilinx (Virtex-7)	The implementation was able to achieve speedup of 11.8× against an Intel Core i7 CPU clocked at 3.20 GHz.
Real-time background generation and foreground object segmentation for high-definition colour video stream (2014) [89]	• Image acquisition; • Background generation; • Segmentation; • Presentation of the results	Xilinx (Virtex-6)	60 fps for colour images with a resolution of 1920 pixel × 1080 pixel. As a comparison, it took 1.7 s to process a single frame with a C++ code in an Intel Core i-7 CPU (speedup of 102-fold)
Image boundary detection (2014) [91]	• Accelerating the probability boundary (Pb) detector algorithm (Pb is a gradient-based algorithm);	Xilinx (Virtex-7)	The execution time was 0.0063 s in the FPGA as oppose to 72.494 s in a 2.1 GHz CPU (speedup of 11507-fold)
Partial Image matching (2016) [93]	• Measurement of the Hamming distance	Altera (Stratix-5)	The execution time was 1.17 s in the FPGA as oppose to 38.4 s in a 2.6 GHz Intel CPU (speedup of 32.8 fold)
Real-time image denoising (2017) [90]	• Discrete Kalman filter (DKF)	Xilinx (Virtex-6)	The implementation was able to denoise 512 pixel × 512 pixel images at 33 fps.
Real-time image processing (2017) [92]	• KMGA (Kubelka-Munk (KM) function, with a function optimiser of genetic algorithm (GA)).	Xilinx (Virtex-7)	The FPGA implementation was 6 times faster than the Matlab implementation and 3 times faster than the C implementation.

4. Graphics processing units (GPUs)

The first graphics accelerators were built for professional graphics workstations, such as the Infinite Reality for the Onyx series [105]. GPUs consist of many processing cores, and are accelerators that are optimised for performing fast matrix calculations in parallel (images are in the form of 2D matrices). These devices are typically very affordable, since their development is motivated by the gaming industry. GPUs are thus cost-effective hardware accelerators for massively parallel algorithms. GPUs have been used in a wide range of applications, other than games, over the last ten years.
NVidia is the most widely known vendor of GPUs. AMD and Intel are other major producers of GPUs. In this review, we only evaluate NVidia GPUs, since they are widely used by the research community.

4.1. NVidia GPU series

NVidia GPU series have a range of core microarchitectures, and can be used for various image processing applications [106] [107]. Some of the main features to consider when selecting a suitable GPU for a specific application include (more details about the GPU hardware architecture can be found in [11]):

- **GPU microarchitecture technology**. The most recent NVidia microarchitectures were named Tesla [108], Fermi [109], Kepler [110], Maxwell [111], Pascal [112], and Volta [113]. They were introduced in 2008, 2010, 2012, 2014, 2016, and 2017, respectively. Successive microarchitecture technologies typically add features and provide improvements over previous generations. For example, the Kepler microarchitecture improved computational power, the Maxwell microarchitecture provided a power-efficient design and an improved scheduler [114], the Pascal microarchitecture featured faster clocks [112], while the Volta microarchitecture had higher memory bandwidth compared to previous microarchitectures.

- **Memory**. GPUs have varying levels of memory. The internal memory of GPUs is typically used for storing image data. This memory is of DRAM type, but is based on different technologies in different GPUs. The main DRAM technologies are DDR2, DDR3, GDDR3, and GDDR5, and each has a different speed and bandwidth. Even though an adequate amount of DRAM memory is necessary for storing the image data for the algorithm, DRAM memory is costly, and GPUs of the same generation that have more memory are more expensive. Therefore, a suitable GPU should be chosen by considering the memory demands of the algorithm.

- **Compute unified device architecture (CUDA) cores**. CUDA cores or stream processors are the smallest processing units of NVidia GPUs, and each task can be assigned to one of them. NVidia microarchitectures have different numbers of CUDA cores. A group of CUDA cores forms a streaming multiprocessor (SM). An SM has 32 cores in the Fermi architecture, 192 cores in the Kepler architecture, 128 cores in the Maxwell architecture, and 64 cores in the Pascal architecture. Massively parallel computer vision and image processing algorithms are suitable to be implemented in GPUs with large numbers of CUDA cores.

- **CUDA compute capability**. CUDA compute capability version indicates some of the main features for programming in GPUs, including the maximum shared memory and the maximum 2D and 3D array size in CUDA cores. NVidia has released six major versions of CUDA compute capability up to 2017 (1.x, 2.x, 3.x, 5.x, 6.x, and 7.x).

- **Processing power**. The maximum number of floating point operations (single or double precision) per second that a GPU can perform is defined as the processing power of that GPU in FLOPs (an acronym for floating-point operations per second). The performance of a GPU in high-performance computing is often measured based on its capability in performing multiplication and addition or fused multiply-add (FMA), which is two FLOPs per instruction.

- **Bus interface**. All modern GPUs use a PCIe interface for data communication with PCs. However, the generation of the PCIe interface, and the number of data lanes that the GPU can use, will determine the maximum data transfer rate.

Among NVidia GPUs, the Tesla series with Kepler microarchitecture was particularly designed for performing technical and scientific computing [121], hence they have been used in high performance applications. GPUs are inherently suitable for performing computer vision and image processing tasks, and many different GPU series can be used for this purpose. NVidia has introduced some examples of computer vision and image processing tasks in [122].

4.2. Tools and utilities for NVidia GPUs

CUDA and OpenCL are the two main programming languages for GPUs. Fang et al. [123] published a detailed performance comparison of CUDA and OpenCL, in 2011.
CUDA was created by NVidia, and is a parallel computing platform and programming model [124]. Not all NVidia GPUs support CUDA, but all the GPUs that are released after 2007 have CUDA support (the list of NVidia GPUs with CUDA support is available in [125]). OpenCL is a programming language which can be used in many different platforms, such as GPUs, CPUs, DSPs, and recently some FPGAs and ARMs (an example is Alteras OpenCL SDK, which was introduced in section 3.2.1). One of the main advantages of OpenCL is its portability across different platforms. OpenCL has the potential to have the same performance as CUDA under a fair comparison, but it requires a higher level of programming skill [123]. CUDA is the preferred programming language for NVidia GPUs. CUDA has a wide range of support from NVidia, it does not require advanced programming skills, and has many libraries and development tools.
The list and description of the libraries and tools available for CUDA can be found in [126]. Some of the basic CUDA libraries for image processing algorithms include:

- CUBLAS was developed for linear algebra calculations. According to NVidia CUBLAS was 6 to 17 times faster than Intels math kernel library (MKL) BLAS for CPUs [127].
- CUFFT is a library for performing FFT. According to NVidia CUFFT was also faster than Intels MKL FFT function for CPUs [128].
- NVidia performance primitives (NPP) is a library of basic image, video, and signal processing functions [129].

CUDA supports different programming languages, of which C/C++ is one of the most widely used. NVidia has introduced some tools to help programmers develop GPU codes in C/C++ language, such as Nvidia Nsight for debugging, building, profiling, and tracing NVidia GPU codes with CUDA and C/C++. NVidia

Nsight is available under both Windows (in the Microsoft visual studio edition [130]) and Linux (the Eclipse edition [131]).
In addition to the C/C++ programming language, NVidia GPUs can be programmed in MATLAB using the MATLAB parallel computing toolbox [132, 133] (NVidia has suggested using the Tesla family for this purpose [134]). ArrayFire is another software library available for GPU programming [135]. ArrayFire acts as an application program interface (API) and simplifies programming with CUDA-capable NVidia GPUs and some other OpenCL devices. ArrayFire functions (the list and details are available in [136]) are designed to perform calculations on arrays to speed up the process, while maintaining a simple interface for programming. There are some other tools for accelerating MATLAB codes using GPUs (refer to [137] for a brief review and comparison).
Some computer vision libraries such as OpenCV [166] and Point Cloud Library (PCL) [167] have added CUDA-capable functions to their recent versions (the PCL CUDA-enabled functions are under development [170]). The OpenCV CUDA module includes low-level primitives (e.g. image filters, corner detection, and edge detection) and high-level functions (e.g. stereo correspondence and face and people detectors) of the OpenCV library. OpenCV CUDA functions could show speedup compared to their CPU-based equivalents. For example, the implementations of primitive image processing functions and stereo vision functions on an NVidia Tesla C2050 were respectively up to 30 and 7 times faster than their CPU-based functions on an Intel Core i5-760 2.8 Ghz [168]. The OpenCV library also offers the OpenCL implementation of some of its functions ([169]) that can be used for non-NVidia GPUs or other hardware platforms such as ARMs or FPGAs.
CUDA-capable NVidia GPUs can also be programmed in NI LabView using the NI-GPU analysis toolkit [138]. In addition to the basic CUDA functions, CUBLAS and CUFFT functions can be wrapped for use in NI LabView with the NI-GPU analysis toolkit [139].

4.3. Tools and utilities for NVidia GPUs

The development time for GPUs is shorter than for FPGAs and DSPs. CUDA and NVidia Nsight facilitate the process of developing and debugging complex GPU codes. It was estimated in a 2012 study that developing algorithms in a GPU for extracting dense optical flow, stereo and local image features will take 2 months for one post-doctoral employee, while developing the same algorithms in an FPGA will take 12 months for two post-doctoral employees [73]. Thus, in this specific application, the development time for GPUs may be 12 times faster than for FPGAs. This shorter development time results from both easier programming, and the simpler architecture of GPUs compared to FPGAs. The development time in GPUs can be made even shorter using the Matlab parallel computing toolbox or ArrayFire. However, writing optimised codes in GPUs for high-throughput algorithms in high-speed applications requires understanding of the GPU hardware architecture and optimisation techniques. Often, algorithms should be modified in a way to suit the particular GPU hardware which is chosen for implementations. Furthermore, careful memory management and data transfer between the host (PC) and the GPU card is necessary.

4.4. Advantages of using GPUs

Important advantages of using GPUs over other hardware accelerators include:

- GPUs are mass produced (primarily for the entertainment industry). Hence they are relatively inexpensive compared to FPGAs, and have the best processing power to price ratio among hardware accelerators [9].
- GPUs are specially designed for performing image and video processing.
- GPUs are programmed with high-level programming languages. Developing and debugging code in GPUs is faster and easier than in FPGAs.
- The PCIe interface between the GPU cards and the host PC can be easily used by programmers.
- NVidia GPUs can be programmed using NI LabView, which is an advantage for using GPUs in instrumentation projects.
- GPU technologies are rapidly advancing and, despite new technologies having higher capabilities, they are often not much more expensive.

4.5. Disadvantages of using GPUs

Important disadvantages of using GPUs over other hardware accelerators include:

- GPUs consume significantly more power compared to FPGAs in the same class of performance, and are thus generally unsuitable for power-sensitive systems that include complicated image processing algorithms.

- GPUs are designed for problems that have massive data parallelism. The performance of GPUs will decrease considerably if they have to wait for data, or if the processing of data is time-consuming and slow.
- The main speed bottleneck in using GPUs in PC-based systems is the data transfer time between the host PC and the GPU. A non-optimised GPU code might not help to increase the processing speed. Thus, it is very important to minimise the GPU data access to the host PC.
- Even though some tools are available for managing the memory in NVidia GPUs (e.g. MATOG [140]), the management of memory and the choice between shared memory, local memory, global memory, constant memory, and texture memory are not straightforward for high performance applications.
- Double precision calculations are theoretically around two times slower than single precision calculations inside GPUs [141]. Although the actual speed reduction of double precision calculations is less than two times in practice (since they require less data throughput), it is important to consider this limitation before deciding about the type of calculation in GPUs.
- The development of many low-level functions in GPUs often requires using the assembly language custom codes. The GPUs hardware is pre-structured and has a lower flexibility compared to that of FPGAs.

4.6. Review of applications that use GPUs

A survey of the use of GPUs in medical image registration was published by Shams et al. [10], in 2010. They investigated a number of different registration criteria algorithms in GPUs. Some of these algorithms include (details are in [10]):

- Sum of square differences (SSD);
- Sum of absolute differences (SAD);
- Normalized cross correlation (NCC);
- Correlation coefficient;
- Gradient correlation;

- Mutual information (MI);
- Normalized mutual information;
- Correlation ratio.

Among these algorithms, SSD was the fastest in GPUs. They also investigated different optimisation methods. Some of these methods include (details are in [10]):

- Powell;
- Simplex;
- Soblex;
- Gradient descent;
- Quasi-Newton;
- Levenberg-Marquardt;
- Simulated annealing;
- Genetic.

To compare the performance of GPUs with FPGAs, Shams et al. [10] defined a normalised performance value, which was the average execution time in milliseconds for a single iteration of the optimisation algorithm and for processing 1 million voxel pairs. The average normalised performance value reported by Shams et al. [10] was higher for GPUs than FPGAs.
In 2011, Fluck et al. [9] published a related survey for 3D and 2D medical image registration on GPUs. They divided the registration transformations into rigid and non-rigid, and they investigated SSD, SAD, NCC, and MI as image registration similarity metrics for the registration criteria. Fluck et al. [9] did not carry out any performance analysis between the registration algorithms, but they summarised the strengths and weaknesses of each. They also investigated programming models and interfaces. For the programming language, they concluded that CUDA has established itself as a popular platform for image registration and other image processing tasks, while OpenCL is an emerging standard for parallel programming.
In another review paper, Eklund et al. [7] published a survey of GPU accelerated

medical image processing. The algorithms were divided into basic image processing operations (filtering, interpolation, histogram estimation, and distance transforms), and commonly used algorithms (image registration, image segmentation, and image denoising). The majority of the image registration methods investigated in their review paper was based on image intensity rather than phase-based optical flow techniques. Phase-based optical flow approaches are computationally complex, since they require using a filter bank to decompose the image into components, and for every component the temporal phase gradient needs to be calculated. For this reason, it is difficult to implement such algorithms in GPUs.
Castao-Dez et al. [142] evaluated the performance of some image processing algorithms such as FFT, matrix algebra, geometrical operations, image reconstruction, and principal component analysis (PCA), in GPUs. They implemented these algorithms in GPUs to enable a comparison with an ordinary CPU. The typical speedup ratio of their GPU implementation compared to the CPU implementation of the same algorithm was between 10 times and 20 times. However, the speedup ratio was very dependent on the type of algorithm, and the image size. For example, the speedup ratio of the FFT algorithm using the NVidia CUFFT library was on average 15 for various image sizes, whereas it was 33 at its maximum for image sizes of approximately 1000 pixel $\times$ 1000 pixel.
Tables 9 and 10 provide a summary of some selected GPU implementations of computer vision and image processing algorithms in recent literature. The most commonly used GPUs were the NVidia GeForce series. Table 8 lists the microarchitecture of the NVidia GeForce series, and other GPUs which were used in the literature of Tables 9 and 10 for implementation of the algorithms.
In Tables 9 and 10, neither the speedup ratio, nor the data throughput, was remarkable for implementation of algorithms in GPUs with Tesla or Fermi microarchitectures (Table 8). However, the GPUs with Kepler, Maxwell, and Pascal microarchitectures (Table 8) showed superior performance. For example, 11475 M disparities/s were achieved using a GTX 680 [149], and speedup ratios of up to 76 and up to 28, compared to the CPU implementation, were achieved using a GTX 760 [153] and GTX-750 Ti, respectively [155]. However, the speedup ratio of the fingerprint identification algorithm in Tables 9 and 10 was not much higher in a Tesla K20 with Kepler microarchitecture than in a Tesla M2090 with Fermi microarchitecture in [154].

4.7. Summary and conclusion for GPUs

In recent years, there has been increasing interest in using GPUs to perform scientific computations. GPUs are relatively inexpensive, have high processing

Table 8: The microarchitecture of some of NVidia GPUs used in the literature

NVidia GPUs	Microarchitecture
GeForce 200 series (GTX 260, GTX 280, GTX 285 , GTX 295)	Tesla
GeForce 300 series	Tesla
Quadro FX 5000 series (FX 5800)	Tesla
GeForce 9 series (GeForce 9400M)	Tesla
Tesla C1060	Tesla
Tesla C2050	Fermi
GeForce 400 series (GTX 480)	Fermi
GeForce 500 series (GTX 570, GTX 580)	Fermi
Tesla M2090	Fermi
GeForce 600 series (GTX 650, GTX 660, GTX 680)	Kepler
GeForce 700 series (GTX 760)	Kepler
Tesla K20	Kepler
GTX 970	Maxwell
GTX-750 Ti	Maxwell
Quadro M6000	Maxwell
GeForce 10 series (GTX 1080)	Pascal

Table 9: A summary of some selected implemented computer vision and image processing algorithms on GPUs in the recent literature

Application(s)	Algorithm(s) Implemented	Hardware (GPU)	Performance/Data throughput
Stereoscopic scene flow computation for 3D motion analysis (2011) [143]	• Image rectification • Residual processing • Stereo matching • Disparity map calculation	GTX 480 (2010)	20 fps for an image size of 320 pixel × 240 pixel
Stereo matching with slanted surface modelling (2011) [144]	• Coarse stereo matching • Disparity plane fitting • Subpixel stereo matching • Disparity map calculation	GTX 480 (2010)	8 M disparities/s with subpixel accuracy (i.e. 5 fps for an image size of 384 pixel × 288 pixel and 16 levels of disparity)
Real-Time Surface Curvature Estimation (2012) [145]	• Computation of principal curvatures, principal directions of curvatures, and the derivative of curvature	GTX 480 (2010) and Quadro FX 5800 (2008)	Speedup of 6 to 8-fold in a Quadro FX 5800 (only), and speedup of 18 to 20-fold in a GTX 480 (only) compared to the multithreaded CPU algorithm
Tomographic reconstruction (2012) [146]	• WBP and SIRT methods (the two most common methods for electron tomography reconstruction)	Two hybrid system of CPUs and GPUs: 2 × Tesla C2050 (2010) 1 × GTX 285 (2009)	The hybrid system could achieve speedup of 2-fold with a Tesla C2050 GPUs, and speedup of 1.5-fold with a GTX 285
Real-time 3D range video encoding and decoding (2012) [147]	• The Holovideo technique, which is devised from the digital fringe projection technique	GeForce 9400M (2008)	18 fps for an image size of 512 pixel × 512 pixel
Colour Stereo Matching (2013) [148]	• Calculation of SAD, and the arm-length-differences (ALD)	GTX 570 (2010)	The implementation could generate matching results for each pair of images in less than 100 milliseconds for an image size of 450 pixel × 375 pixel
A stereo vision system for real-time tracking (2014) [149]	• Stereo matching with symmetric dynamic programming stereo (SDPS) • Image rectification • SDPS disparity generation • Joint colour disparity filter • Foreground 3D reprojection • Post-processing (speckle filtering)	GTX 680 (2012)	11475 M disparities/s (i.e. 114 fps for 1024 pixel × 768 pixel images and 128 levels of disparity)
Large-size VHR images registration (2014) [150]	• Coarse registration (using SIFT and RANSAC algorithms) • Fine registration • Image rectification based on the triangulated Images	GTX 650 (2012)	The image rectification speed was increased by 16 times compared to CPU implementation
Digital volume correlation (2014) [151]	• Coarse search and registration using FFT and IFFT • Subpixel registration based on an optimisation process with Broyden-Fletcher-Goldfarb-Shanno algorithm	3 × Tesla M2090 (2012)	Speedup of 8-fold by using three Tesla M2090 GPUs in comparison to the CPU alone
Training of image convolution filter weights using genetic algorithms (2015) [152]	• Sub-images-based method for training of the genetic algorithm	GTX 660 (2012)	Speedup of 55 to 90-fold using a GeForce GTX 660 over sequential implementation in a 3.5GHz CPU
Digital image correlation (2015) [153]	• FFT and IFFT for finding the integer shift • Inverse compositional GaussNewton (IC-GN) algorithm for finding the sub-pixel shift	GTX 760 (2013)	Speedup of 57 to 76-fold over sequential implementation on a CPU, with the same accuracy

Table 10: A summary of some selected implemented computer vision and image processing algorithms on GPUs in the recent literature

Application(s)	Algorithm(s) Implemented	Hardware (GPU)	Performance/Data throughput
Fingerprint identification (2015) [154]	• Feature matching (global and local) for fingerprint images	2 × Tesla K20 (2012) 2 × Tesla M2090 (2012)	Speedups compared to the multi-threaded CPU implementations are: • 15.69 for 1 × Tesla K20 • 14.79 for 1 × Tesla M2090 • 30.49 for 2 × Tesla K20 • 28.71 for 2 × Tesla M2090 • 54.20 for all 4 GPUs
Volume image registration (2015) [155]	• Sorting the data • Estimating a cost function • Calculating the correlation ratio	GTX-750 Ti (2014)	Speedup of 28-fold in comparison to the CPU alone
A ParaView library for image compression (2016) [156]	• H.264 (a block-oriented motion-compensation-based video compression standard)	GTX 1080 (2016)	Compression rate of 150 fps compared to 30 fps for a CPU implementation (Speedup of 5-fold)
Flow-Guided Image Warping (2016) [157]	• Structure tensor • Flow calculation • Image warping	Quadro M6000 (2015)	10.3 s to warp a 1600 pixel × 1200 pixel image for a neighbourhood size of 100 pixel × 100 pixel
A 3D detection radar in the THz band (2017) [158]	• Spatial domain windowing • FFT • Peak detection • Finding surface coordinates	GTX 970 (2014)	Image refresh at 8 fps for images composed of 6000 pixels, corresponding to a scanned area of 50 cm x 90 cm.

speeds in their latest generations, and it is relatively simple to develop complicated codes for them. The development time for computer vision and image processing applications for GPUs is less than for FPGAs, and GPUs can provide a good performance in applications where no data acquisition is required.
The speedup that one can obtain by implementing the algorithms in a GPU varies depending on the algorithm type, the GPU microarchitecture, and programming techniques. However, Brodtkorb el al [159] suggested that considering the latest technologies in GPUs and CPUs, when the algorithms are not being adapted for GPUs, the performance speedup in the GPU can be approximately seven times compared to the CPU implementation [159].
Even though recent NVidia GPUs for laptops show higher performance, they are mainly designed for gaming not scientific computing. Because of this, desktop GPUs are still a more suitable option for scientific computing. There are some cost-efficient GPUs available on single board computers (SBC) for relatively simple image processing tasks. The most commonly used example is the Raspberry Pi [171] which includes an on-board Broadcom VideoCore IV GPU at 250 MHz [172]. The Raspberry Pi has been used for implementation of several image

processing and computer vision algorithms, such as face detection [173], model-based detection and tracking [174], and some deep learning methods for computer vision applications[175].

In summary, there are some applications in which GPUs are a more suitable option compared to other hardware accelerators. Some examples include:

- The programming language of GPUs is high-level, and their development time is shorter than FPGAs, hence they are more suitable for fast prototyping.
- GPUs are suitable hardware-accelerators in PC-based systems in which capturing of high throughput data is not required.
- GPUs are less expensive than FPGAs, hence GPUs are more suitable for cost-sensitive applications.
- It is usually easier to transfer codes to new hardware in GPUs than in FPGAs. GPUs are thus more suitable hardware accelerators for algorithms which need to be frequently updated.

The GPUs being used in the literature are typically selected from recent technologies at the time of publication. The codes could even be developed before the release date of the GPUs, and then quickly modified for that specific GPU.
Occasionally, even the latest GPUs do not have sufficient computational power and/or memory units for complex algorithms. The number of CUDA cores that can be included into a single GPU is limited by the manufacturing constraints. One solution is to use multiple GPUs either in a single system or across a network. In comparison, multiple GPUs in a single PC have better performance per watt compared than single GPUs in a network. Scalable Link Interface (SLI) is a brand name for the multi-GPU technology from NVidia for a single PC. SLI can support up to four individual GPUs in one motherboard (note that some of NVidia GPUs, such as GeForce GTX Titan Z, have two GPUs on one card).
Multiple GPUs have been used for image processing and computer vision algorithms. For instance, two Tesla K20s and two Tesla M-2090s were used in two separate implementations for fingerprint identification in [154], and 2 Tesla C2050s were used for tomographic reconstruction in [145].
There are some limitations associated with using multiple GPUs. For example, it is challenging to manage data communication and computations between multiple GPUs. Hence, the computation speed of multiple GPUs compared to a single

GPU is not typically proportional to the number of GPUs used~~, or even in~~ some cases, it may be lower than that for a single GPU. Furthermore, in an SLI configuration all of the GPUs must be from the same GPU series.
GPUs and FPGAs are more commonly used compared to DSPs for implementing image processing and computer vision algorithms. It is thus interesting to compare implementations in an FPGA and a GPU. However, an accurate performance comparison between FPGAs, and GPUs is not practical since each application has different demands, and the performance depends on the level of programmer proficiency and the hardware being used. In most of the published papers, the comparison between FPGAs and GPUs is not performed for comparable technology levels, with GPUs usually belonging to a more recent generation compared to FPGAs. This may be a consequence of cheaper costs of GPUs over FPGAs, or their relative ease of programming. However, comparing some examples of implementation of the same algorithm in both FPGAs and GPUs can help developers to achieve an approximate performance comparison. Section 7 provides a brief comparison between FPGAs and GPUs that are being used as hardware accelerators for computer vision and image processing applications.

5. Portability of software over different hardware

It is sometimes required to transfer codes from one hardware accelerator to another of the same type, such as when upgrading to a new generation hardware, or when testing the code in another device. The transfer process may be challenging if the available code is crafted to take advantage of the specific architecture of the original hardware. In this section, we discuss how it could be possible to transfer the code and potential challenges for DSPs, FPGAs, and GPUs.

5.1. Transferring codes among DSPs

C/C++ is the most widely used programming language for all of the DSP families (section 2.1). Hence, transferring codes among DSPs does not require re-writing them. However, the transfer can be difficult if the code uses specific functions, hardware codecs, or memory units of the DSP.
The simplest code transfer happens from a lower-performance DSP to a higher-performance DSP of the same family. For instance, C6000 series codes can be easily transferred to another DSP from the C64x series. Transferring DSP codes from a low-performance DSP to a high-performance DSP of another family is

also relatively simple. For instance, power optimised DSP codes (from the C6000 family) can be easily transferred to a multi-core DSP (from the C66x family). However, the transferred code will not use the parallel processing capabilities that the multi-core DSP offers in this example.
Transferring codes from two high-performance DSPs can become challenging if the code uses particular hardware capabilities of the source DSP. For instance, DaVinci-DMPs (section 2) have hardware codecs for video and audio processing which are not available in multi-core DSPs. Moreover, multi-core DSPs can perform parallel computations, which are not available in DaVinci-DMPs. In this example, the transferring of code should be decided based on which DSP is a better choice for the particular algorithm.

5.2. Transferring codes among FPGAs

Transferring codes among Xilinx FPGAs and Altera FPGAs is not straightforward. Even though the FPGA code may be written in the VHDL language (which is the basic language for both), Xilinx Vivado and Altera Quartus II use different approaches. Furthermore, Xilinx and Altera have different IP-cores that are specifically designed for their own FPGAs.
It is relatively simple to transfer codes from a lower-performance FPGA to a higher-performance FPGA of the same series in both Xilinx and Altera FPGAs. It is only required to re-synthesise the code for the new hardware. The re-synthesised code can take advantage of the new hardware capabilities of the target FPGA. Nevertheless, the code needs to be re-written to use sequential processing if the target FPGA has an embedded processor.
Transferring code from a higher-performance FPGA to a lower performance FPGA is not difficult if the lower performance FPGA has sufficient logic units for the code, and the code is not using specific hardware units of the higher-performance FPGA (such as the embedded processor). The only required step is to re-synthesise the code for the new FPGA.

5.3. Transferring codes among GPUs

Transferring GPU codes among NVidia GPUs of the same generation is not usually challenging. The memory size and the processing power of the target GPU are the two major considerations that should be taken into account.
The transfer of code from a lower-performance GPU to a higher-performance GPU only requires ~~re-compilation~~ of the code. Nevertheless, the code needs to

be ~~re-written~~ to use any of the new capabilities of the target GPU.
Transferring code from a higher-performance GPU to a lower performance GPU only requires ~~re-compilation~~ of the code if the lower performance GPU has sufficient memory blocks and uses the same CUDA compute compatibility.

6. Heterogeneous hardware accelerators

Heterogeneous hardware accelerators are designed to use the advantages of a hardware accelerator while offsetting its disadvantages by fusing its functionality with another hardware accelerator. For instance, as discussed in section 4.5, one of the disadvantages of GPUs is the data transfer time between the host PC and the GPU. This time is decreased in heterogeneous CPU-GPU computing architectures, such as accelerated processing units (APUs) designed by AMD (formerly known as Fusion). An AMD APU is the fusion of an AMD 64-bit microprocessor and an AMD GPU on a single chip. AMD APUs have shown an improved performance in data transfer compared to discrete GPUs. For example, in a benchmark study, an AMD Fusion had 3.5 times faster data transfer rate when compared to a discrete AMD Radeon HD 5870 GPU [176].
NVidia ~~has also~~ a plan for producing a new generation of GPUs with integrated CPU and GPU cores based on the ARM architecture [159]. This technology will make GPUs a more suitable option for mobile applications, and will help to decrease the need for transferring data between the GPU and the CPU, which is currently one of the main performance bottlenecks for GPUs. However, those GPUs are not designed for high-performance applications.
Manycore processors can also operate similarly to a CPU-GPU hardware and heterogeneous computers. For instance, the Intel Xeon Phi is an example of a manycore x86 processor that was designed based on an earlier GPU design by Intel. The early versions of Intel manycore processors were known by the codename of Knights (e.g. Knights Ferry).
Hybrid CPU-FPGA architectures have also been designed and developed. The acquisition of Altera by Intel in 2015 [177] has accelerated this process. Although Intel CPU-FPGA processors have some overlapping applications with the Intel Xeon Phi, they are more suitable for the applications for which FPGAs have advantages over GPUs. Intel has introduced the Acceleration Stack technology for Intel Xeon CPUs combined with FPGAs [178]. This technology was the basis of a recently released CPU(Xeon)-FPGA(Arria 10 GX) board by Intel-Altera [179]. The software development for CPU-GPU architectures is also an active area. The

Open Programmable Acceleration Engine (OPAE) [180] is an example that has been developed to simplify the programming of Intel CPU-FPGA SoC devices.

7. Comparison of FPGAs and GPU for implementing image processing, and computer vision algorithms

NVidia GPUs have been used more than FPGAs for high performance applications in recent years. For instance, the second fastest supercomputer in the world, named Titan, includes 18,688 NVidia Tesla GPUs, and has a processing power of more than 2×1016 calculations per second [160].
Among computer vision and image processing algorithms, stereo vision algorithms are the most common application implemented in hardware accelerators. Tippetts et al. [4] reviewed the implementation of various stereo vision algorithms in CPUs, GPUs, and FPGAs. Among the reported publications in their review, the maximum disparities/s for algorithms were 6 million in CPUs, 7,247 million in GPUs, and 6,062 million in FPGAs. Among the papers reviewed by Tippetts et al. [4], the fastest GPU and FPGA implementations were approximately 1000 times faster than the fastest CPU implementation. However, the CPU implementation was in a single core Pentium 4 CPU (introduced in 2000), while the latest FPGAs being used were Xilinx Virtex-5 and Altera Stratix-3 (both introduced in 2006), and the latest GPU was NVidia Geforce GTX 280 (GeForce 200 series, introduced in 2008 (Table 8)). Thus, all of these hardware technologies reviewed by Tippetts et al. [4] were far from state-of-the-art, being outdated even at the publication time of Tippetts et al.s review [4] in 2013.
Brodtkorb et al. [11] published a comprehensive review paper, and described hardware and software tools in heterogeneous computing. They summarised the differences between FPGAs and GPUs in architecture, performance, power consumption, cost, programming languages, and debugging tools. In addition, they investigated the implementation of some basic algorithms. For example, Brodtkorb et al. [11] reviewed the implementation of FFT with NVidia CUFFT library in a GTX 280 GPU (Table 8), which they reported to be 8 times to 40 times faster than Intels MKL on a 3 GHz Intel Core2 Quad [11]. Unlike the review of Tippetts et al. [4], in the review by Brodtkorb et al. [11] the hardware technologies were relatively recent in 2010 when the paper was published. However, for implementation of FFT in FPGAs, even though Brodtkorb et al. [11] had introduced Xilinx and Altera IP-cores, there was no performance comparison in the FPGA section of that paper.

Gao et al. [6] reviewed the applications of parallel computing in experimental mechanics and optical measurements using hardware accelerators (DSPs, FPGAs and GPUs). They mentioned that it is difficult to increase the speed of such algorithms in CPU-based systems, whereas using data parallelism techniques makes them suitable choices for parallel hardware, such as FPGAs and GPUs. As an example, they reported a 47 times speedup using an NVidia GeForce 6600 GPU in comparison to a Pentium 4 CPU. However, both of these were old technologies even at the publication date of the review paper in 2012 (GeForce 6600 was introduced in 2004). In total, Gao et al. [6] reviewed 38 papers from 2008 to 2012, of which 42% used GPUs, 29% used FPGAs, and 29% used others. Based on their survey, GPUs were the most popular hardware accelerators in experimental mechanics and optical measurements.

Asano et al. [161] reviewed the performance of FPGAs, GPUs, and CPUs in three different applications in image processing. The algorithms they investigated included two-dimensional filters, stereo-vision (feature matching, and 3D projection), and k-means clustering [161]. The GPU used in their study was an NVidia GTX 280 (Table 8), and the FPGA they used was a Xilinx Virtex-4 (introduced in 2005 (Table 3)). In their comparisons, the GPU performed better for 2D filter sizes less than 11, and the FPGA performed better for stereo vision (feature matching, and 3D projection), and *k*-means clustering. Asano et al. [161] concluded that the GPU is preferable for simple tasks where pixels can be processed independently, whereas for more sophisticated algorithms FPGAs perform better.

Fowers et al. [5] compared the performance and power consumption of FPGAs, GPUs, and multicore processors for sliding-window applications. The GPU and FPGA used for the sliding-window applications in [5] were a GTX 295 (introduced in 2008 (Table 8)), and an Altera Stratix-3 (introduced in 2006 (Table 4)). They investigated SAD, 2D convolution, and correntropy [162] (i.e. a measure of similarity based on information theoretic learning) as sliding-window applications [5]. In their implementations, the FPGA implementation gave the best performance for SAD, and was the only device which was able to do real-time computations up to 50×50 kernel size. The FPGA implementation could also reach approximately 80 fps for a 720p image size, while the frame rate was approximately 10 fps for the GPU implementation, and less than 1 fps for the CPU sequential C++ implementation.

Fowers et al. [5] also analysed the 2D convolution performance for the FPGA and GPU implementations in the time domain, as well as the GPU implementation in the frequency domain using FFT [5]. For the 2D convolution, the frequency domain GPU code (GPU-FFT) showed the best performance where, for an image size of 1280 pixel $\times$ 720 pixel and a kernel size of 25, a frame rate of approximately 120 fps was achieved in the GPU, whereas the FPGA implementation in the time domain could achieve 80 fps, and the GPU implementation in the time domain could achieve 60 fps. However, their comparison of the frequency domain implementation in GPU and the time domain implementations in FPGA is unfair, since implementation of 2D convolution in frequency domain for large kernel sizes is faster than time domain. As can be seen, the FPGA implementation of Fowers et al. for 2D convolution in the time domain could achieve 20 fps more than the one implemented in the GPU in the time domain.

Fowers et al. [5] improved their methodology for comparing FPGA and GPU implementations in their next paper [163] by adding the FPGA frequency domain implementation of 1D convolution to their comparisons. The FPGA in their study was a Gidel ProcSTAR III board, which contains four Altera Stratix-3 FPGAs, and the GPU was an NVidia GeForce 295 GTX (Table 8). They considered two scenarios in the FPGA performance analysis: first where the data is sent to the FPGA board via PCIe; and the second was a standalone FPGA, where the data source was directly connected to the FPGA (similar to the embedded systems). From these two scenarios, the standalone FPGA showed better performance for the implementation of 1D convolution in the frequency domain compared to the PCIe FPGA board. For instance, when performance was compared to the frequency domain implementation in the CPU, the speedup ratios of the standalone FPGA and the PCIe FPGA were approximately 10 and 5, respectively. The standalone FPGA could perform better than the GPU for large signal sizes with small convolution kernel sizes. However, in other cases (such as small or large signal sizes with large convolution kernel sizes) the GPU had better performance.

Pauwels et al. [73] compared the performance of FPGAs and GPUs for real-time phase-based optical flow, stereo, and local image feature matching [73]. The FPGAs they used were Xilinx Virtex-4 and Virtex-5, and the GPUs were GeForce GTX 280 (Table 8) and GTX 580 (introduced in 2010 (Table 8)). A comparison was performed for accuracy, speed, power consumption, cost, and design time. They found that GPUs and FPGAs were suitable for different applications. For example, FPGAs were more suitable than GPUs for local feature matching, median filtering, and embedded platforms. FPGAs also had advantages over GPUs for their low power consumption, and level of flexibility. On the other hand, Pauwels et al. [73] concluded that GPUs were more suitable for image warping and applying spatial filters on image pyramids. In their study they suggested that the advantages of GPUs over FPGAs are their low cost and short design time.

In another comparison between FPGAs and GPUs, Cortie et al. [117] compared an FPGA implementation in Xilinx Spartan-3 for parallel convolutions to an Intel Xeon CPU, and an NVidia Tesla C1060 GPU (introduced in 2008 (Table 8)) [117] (refer to Table 7 for more details about FPGA implementation). They showed that, although they could reach higher speeds with both the FPGA and GPU in comparison to the CPU, the FPGA was faster in smaller systems and the GPU performed better in larger systems.

Choi et al. [164] compared the speed of their implementation of stereo matching using Markov random field in four Xilinx Virtex-5 FPGAs to its GPU implementation on a GTX 260. The GTX 260 was introduced in 2008 and has the Tesla microarchitecture (Table 8). The speed of the FPGA implementation of the Choi et al. [164] was 26.10 ms, as opposed to 61.41 ms for the GPU implementation. However, despite the FPGA implementation being faster than the GPU implementation, it required the use of four FPGAs. Hence, it is likely that the implementation of the algorithm in a single Virtex-5 FPGA would not be faster than its GPU implementation in a GTX 260.

Very few papers have evaluated the performance of the same technology level for FPGAs and GPUs. In one recent paper, Birk et al. [165] performed an efficiency comparison for algorithms in 3D ultrasound computer tomography in 40 nm and 28 nm fabrication technology generations of FPGAs and GPUs. The hardware accelerators from the 40 nm fabrication technology were Xilinx Virtex-6 (Table 3) for the FPGA and NVidia GTX 580 (Table 8) for the GPU. The hardware accelerators from the 28 nm fabrication technology were Xilinx Virtex-7 (Table 3) for the FPGA and NVidia Tesla K20 (Table 8) for the GPU. This selection of hardware accelerators from the same fabrication technology enabled a rigorous and fair comparison between FPGAs and GPUs. Birk et al. concluded that the GPU and the FPGA using the 40 nm fabrication technology can provide similar performance and efficiency, if the power consumption is not considered. In contrast, for the 28 nm fabrication technology, the FPGA implementation had a speedup of 1.86 compared to its GPU counterpart. This implies that FPGAs have substantially improved in their latest generation. The comparisons between the GPU and FPGA implementations discussed in this section are summarised in Table 11.

One of the differences between FPGAs and GPUs in the literature is in using multiple hardware accelerators at the same time to implement complex algorithms. NVidia GPUs use PCIe as their interface to PCs, and the interface is fully developed by NVidia. It is thus simple to use multiple GPU cards at the same time in one PC. In contrast, developing PCIe interfaces for multiple FPGA cards is difficult. Using multiple GPU cards is thus more common compared to multiple FPGA boards when implementing complex algorithms (some examples of using

Table 11: A summary of comparisons between GPU and FPGA implementation of some of the image processing algorithms in the recent literature

Algorithm(s) Implemented	Hardware (GPU)	Hardware (FPGA)
Disparity measurement [4]	NVidia GTX 280: 7.2M Disparities/s	Xilix Virtex-5: 6M Disparities/s
FFT computation [11]	NVidia GTX 280: $8\times$ to $40\times$ faster than its CPU implementation	Not available
2D filters, stereo vision, and k-mean clustering [161]	NVidia GTX 280: Performed better than the FPGA for 2D filter sizes smaller than 11.	Xilinx Virtex-4: Performed better than the GPU for stereo vision, and k-mean clustering.
SAD algorithm [5]	NVidia GTX 295: 80 fps for a 720p image.	Altera Stratix-3: 10 fps for a 720p image.
2D convolution [5]	NVidia GTX 295: 120 fps for an image size of 1280 pixel $\times$ 720 pixel and a kernel size of 25.	Altera Stratix-3: 80 fps for an image size of 1280 pixel $\times$ 720 pixel and a kernel size of 25.
1D convolution [5]	NVidia GTX 295: Performed better than the FPGAs for small or large signal sizes with large convolutional kernel sizes.	$4\times$ Altera Stratix-3: Performed better than the GPU for large signal sizes with small convolutional kernel sizes.
Phase-based optical flow [73]	NVidia GTX 580: Performed better than the FPGA for image warping and applying spatial filters on image pyramids.	Xilinx Viretex-5: Performed better than the GPU for local feature matching and median filtering.
Parallel convolution [117]	NVidia Tesla C1060: Performed better than the FPGA in larger kernels.	Xilinx Spartan-3: Performed better than the GPU in smaller kernels.
Stereo matching using Markov random field [164]	NVidia GTX 260: Processing time of 61.41 ms.	$4\times$ Xilinx Virtex-5: Processing time of 26.10 ms.
3D ultrasound computer tomography image processing [164]	NVidia Tesla K20: The implementation was 1.86 slower than the FPGA.	Xilinx Virtex-7: The implementation was 1.86 faster than the GPU.

multiple GPUs were introduced in Tables 9 and 10).

8. Hardware accelerators designed for machine learning

In recent years, the application of machine learning techniques has been growing very rapidly. In particular, deep neural networks (i.e. deep learning) and convolutional neural networks have been used extensively in various applications. Image processing and computer vision applications have also taken advantage of machine learning [181] and deep learning [182, 184] techniques. GPUs are naturally ~~suitable for~~ the implementation of neural networks because of the similarity between the mathematical basis of neural networks and image manipulation tasks of GPUs. In particular, GPUs are very suitable for training convolutional neural networks because they consist of many matrix multiplications. Given their advantages, GPUs have been the most common choice of hardware implementation. ImageNet [183], which is one the most famous image classification algorithms based on deep learning, has been trained on two GTX 580 3GB GPUs [183] (the training took between 5 and 6 days). In another example, an image enhancement algorithm based on deep learning was implemented on a Titan X GPU [187].
FPGAs have also been used for the implementation of neural networks [185] and deep learning algorithms [184]. The efficient integer arithmetic of FPGAs is an advantage to ~~speedup~~ the training and using of neural networks. In addition, FPGAs offer more flexibility compared to GPUs for the implementation of neural networks. However, the use of FPGAs for nonlinear operations and feedback loops in neural networks is challenging. One way to address this challenge is to use the FPGA in combination with a sequential processor (i.e. in the form of heterogeneous computing or SoC). For instance, Microsoft has designed a new cloud-scale, FPGA-based acceleration architecture for data centres that uses both FPGAs (Altera Stratix V) and CPUs [186]. The acquisition of Altera by Intel (as discussed in Section 6) is another indication of the future trends to develop FPGA-CPU SoCs.
Due to ~~the high demands~~ for using hardware accelerators to implement machine learning techniques, some new SoCs are designed specifically for machine learning tasks. NeuFlow is an SoC (FPGA-CPU) designed to accelerate neural networks and computer vision algorithms based on convolutions operations [188]. This SoC could achieve 320 ~~Giga~~ operations per second (GOP/s) with an average power consumption of 0.6 W [188]. DianNao is another SoC designed for convolutional neural network and computer vision algorithms based on convolution

operations [189]. DianNao has an efficient memory architecture and was capable of 452 GOP/s [189]. ShiDianNao is a newer version of DianNao designed for neural networks and visual recognition algorithms [190]. ShiDianNao was approximately 2 times faster than DianNao for the implementation of neural networks and consumed 60 times less energy [190]. The Tensor Processing Unit (TPU) is a SoC designed by Google for deep neural networks [191]. TPU performed 15 times faster than an NVidia k80 GPU while using two times less power [191].
The significant interest around the application of machine learning techniques has generated a new area in the design of hardware accelerators. This has affected the conventional ways of using hardware accelerators for image processing and computer vision tasks. The future trend for algorithms based on neural networks is to use hardware accelerators specifically designed for the computation requirements of neural networks.

9. Summary and conclusions

In this review, we have provided practical information for selecting suitable hardware accelerators for computer vision and image processing algorithms. We discussed the hardware architectures of the most recent DSPs, FPGAs, and GPUs, and the important features of these hardware accelerators for computer vision and image processing algorithms. For each hardware accelerator, available tools and utilities, development time, advantages, and disadvantages were discussed in an attempt to help developers to choose the most appropriate hardware for their application. Examples from the literature were reviewed in separate sections for applications of DSPs, FPGAs, and GPUs in accelerating computer vision and image processing algorithms. Details of the implemented algorithm, the hardware type, and the hardware introduction year were included. Among hardware accelerators, FPGAs and GPUs are widely used in computer vision and image processing applications. Thus a specific comparison of the performances of FPGAs and GPUs was provided. The most suitable applications for each hardware are summarised in Table 12.

Among the hardware accelerators, DSPs are the least commonly used for computer vision and image processing tasks, and GPUs are the most commonly used. GPUs are suitable hardware accelerators for applications that need short development times or fast prototyping, require good processing speed, or need significant on-board memory. In comparison, FPGAs are the most suitable hardware accel-

Table 12: A summary of the most suitable applications for each hardware

Hardware	Most suitable applications
DSP	• Low-power. • Low-cost. • Portable. • Computationally simple algorithms.
FPGA	• Low-power. • Portable. • Computationally simple or complex algorithms. • Algorithms that can take advantage of a flexible and/or custom-designed hardware.
GPU	• Relatively low-cost. • Require a fast development time. • Computationally simple or complex algorithms. • Algorithms that can become massively data-parallel.

erators for algorithms that require significant processing speed, need to process complicated algorithms in low-power applications, or include customised algorithms. DSPs are suitable hardware accelerators for very low-power applications, or cheap portable devices.

Even though GPUs are frequently used for computer vision and image processing algorithms, and it is often believed that they are the most suitable choice, recent developments have made FPGAs an attractive option for many applications. The tools and utilities from Xilinx and Altera have simplified the code development process, and shortened the development time when using FPGAs. Furthermore, the latest FPGA technologies provide a good level of performance.

Processing speed is an important factor in most computer vision and image processing applications, and hardware technologies are rapidly growing to address this demand. Selection of a suitable hardware accelerator could thus have a great impact on the performance of the system. In this review, we have attempted to provide practical information about the state-of-the-art hardware accelerators to assist researchers and developers in selecting suitable hardware accelerators for their specific applications.

References

[1] M. A. Sutton, Computer vision-based, noncontacting deformation measurements in mechanics: a generational transformation, Appl. Mech. Rev., vol. 65, no. 5, p. 50802, Aug. 2013.

[2] P. Markelj, D. Tomaevi, B. Likar, and F. Pernu, A review of 3D/2D registration methods for image-guided interventions., Med. Image Anal., vol. 16, no. 3, pp. 64261, Apr. 2012.

[3] R. Poppe, A survey on vision-based human action recognition, Image Vis. Comput., vol. 28, no. 6, pp. 976990, 2010.

[4] B. Tippetts, D. J. Lee, K. Lillywhite, and J. Archibald, Review of stereo vision algorithms and their suitability for resource-limited systems, J. Real-Time Image Process., pp. 121, 2013.

[5] J. Fowers, G. Brown, P. Cooke, and G. Stitt, A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications, Proc. ACM/SIGDA Int. Symp. F. Program. Gate Arrays - FPGA 12, p. 47, 2012.

[6] W. Gao and Q. Kemao, Parallel computing in experimental mechanics and optical measurement: a review, Opt. Lasers Eng., vol. 50, no. 4, pp. 608617, 2012.

[7] A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, Medical image processing on the GPUPast, present and future, Med. Image Anal., vol. 17, no. 8, pp. 10731094, 2013.

[8] L. Shi, W. Liu, H. Zhang, Y. Xie, and D. Wang, A survey of GPU-based medical image computing techniques., Quant. Imaging Med. Surg., vol. 2, no. 3, pp. 188206, Sep. 2012.

[9] O. Fluck, C. Vetter, W. Wein, a Kamen, B. Preim, and R. Westermann, A survey of medical image registration on graphics hardware., Comput. Methods Programs Biomed., vol. 104, no. 3, pp. e45-57, Dec. 2011.

[10] R. Shams, P. Sadeghi, R. A. Kennedy, and R. I. Hartley, A survey of medical image registration on multicore and the GPU, Signal Process. Mag. IEEE, vol. 27, no. 2, pp. 5060, 2010.

[11] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, State-of-the-art in heterogeneous computing, Sci. Program., vol. 18, no. 1, pp. 133, 2010.

[12] `http://www.ti.com/lsds/ti/dsp/overview.page`.

[13] `http://www.ti.com/lsds/ti/dsp/c5000_dsp/overview.page`.

[14] `http://www.ti.com/lsds/ti/dsp/c6000_dsp/overview.page`.

[15] `http://www.ti.com/lsds/ti/dsp/video_processors/overview.page`.

[16] `http://processors.wiki.ti.com/index.php/Keystone_Device_Architecture`.

[17] `http://www.ti.com/lsds/ti/dsp/keystone_arm/overview.page`.

[18] `http://processors.wiki.ti.com/index.php/Multicore`.

[19] `http://www.ti.com/tool/ccstudio`.

[20] `DSPLIB:http://www.ti.com/tool/sprc265`.

[21] `http://processors.wiki.ti.com/index.php/IMGLIB`.

[22] `http://www.ti.com/tool/sprc264`.

[23] `MathLIB:http://www.ti.com/tool/mathlib`.

[24] `http://processors.wiki.ti.com/index.php/Category:SYSBIOS`.

[25] `MCSDK:http://www.ti.com/tool/bioslinuxmcsdk`.

[26] `http://processors.wiki.ti.com/index.php/BIOS_MCSDK_2.0_User_Guide`.

[27] `http://processors.wiki.ti.com/index.php/Multicore_System_Analyzer`.

[28] DSPboards:http://www.ti.com/lit/sg/sprt285f/sprt285f.pdf.

[29] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze, A fast stereo matching algorithm suitable for embedded real-time systems, Comput. Vis. Image Underst., vol. 114, no. 11, pp. 11801202, Nov. 2010.

[30] S. B. Goldberg and L. Matthies, Stereo and IMU assisted visual odometry on an OMAP3530 for small robots, in Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on, 2011, pp. 169176.

[31] Z. Jun and G.-O. Liu, Design and Implementation of Networked Real-time Control System with Image Processing Capability, UKACC Int. Conf. Control, pp. 15, 2014.

[32] Y. Chen, B. Wu, S. Member, H. Huang, and C. Fan, A Real-Time Vision System for Nighttime Vehicle Detection and Traffic Surveillance, vol. 58, no. 5, pp. 20302044, 2011.

[33] Y. F. Cao, M. Ding, L. K. Zhuang, and Y. K. Cao, Vision-based Guidance, Navigation and Control for Unmanned Aerial Vehicle Landing, 9th Int. Bhurban Conf. Appl. Sci. Technol., pp. 8791, 2012.

[34] V. Gonzalez-Huitron, E. Ramos-Diaz, V. Kravchenko, and V. Ponomaryov, 2D to 3D Conversion Based on Disparity Map Estimation, CIARP 2014, pp. 982989, 2014.

[35] S.-J. Huang and F.-R. Ying, Stereo vision system for moving object detecting and locating based on CMOS image sensor and DSP chip, Pattern Anal. Appl., vol. 15, no. 2, pp. 189202, Jan. 2011.

[36] F. D. Igual, G. Botella, C. Garca, M. Prieto, and F. Tirado, Robust motion estimation on a low-power multi-core DSP, EURASIP J. Adv. Signal Process., vol. 2013, no. 1, pp. 115, 2013.

[37] R. Berg, L. Knig, J. Rhaak, R. Lausen, and B. Fischer, Highly efficient image registration for embedded systems using a distributed multicore DSP architecture, J. Real-Time Image Process., Nov. 2014.

[38] J. Karam, I. Alkamal, A. Gatherer, G. A. Frantz, D. V Anderson, and B. L. Evans, Trends in Multicore DSP Platforms, IEEE Signal Process. Mag., vol. 26, pp. 3849, 2009.

[39] `http://www.vhdl.org/`.

[40] `http://verilog.org/`.

[41] `http://www.accellera.org/community/systemc/`.

[42] J. Agron, Domain-specific language for HW/SW Co-design for FPGAs, Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 5658 LNCS, pp. 262284, 2009.

[43] A. T. and J. T. Moritz Schmid, Frank Hannig, High-Level Synthesis Revised: Generation of FPGA Accelerators from a Domain-Specific Language using the Polyhedron Model, Parallel Comput. Accel. Comput. Sci. Eng., vol. 25, pp. 497506, 2014.

[44] N. George, H. Lee, D. Novo, T. Rompf, K. J. Brown, A. K. Sujeeth, M. Odersky, K. Olukotun, and P. Ienne, Hardware system synthesis from Domain-Specific Languages, in 2014 24th International Conference on Field Programmable Logic and Applications (FPL), 2014, pp. 18.

[45] F. Stock, A. Koch, and D. Hildenbrand, FPGA-Accelerated Color Edge Detection Using a Geometric-Algebra-To-Verilog Compiler.

[46] P. specification Xilinx, Zynq-7000 All Programmable SoC Overview Zynq-7000, vol. 190, pp. 123, 2016.

[47] `Virtex-5:http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf`.

[48] (DS150), Virtex 6 Family, Xilinx, 2012.

[49] `http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf`.

[50] Xilinx, UltraScale Architecture and Product Overview, pp. 129, 2015.

[51] (UG369),Virtex-6 FPGA DSP48E1 Slice, 2012.

[52] Xilinx,Bringing Ultra High Productivity to Mainstream Systems and Platform Designers Vivado Design Suite HLx Editions.

[53] (UG631),ISE Design Suite 14: Release Notes , Installation , and Licensing, 2013.

[54] `http://www.xilinx.com/products/design-tools/vivado`.

[55] `http://www.xilinx.com/products/design-tools/vivado/verification-and-debug.html`.

[56] `http://www.xilinx.com/products/intellectual-property/`.

[57] (UG683), EDK Concepts, Tools, and Techniques A Hands-On Guide to Effective Emedded System Design, 2013.

[58] (UG897), Model-Based DSP Design Using System Generator, pp. 1173, 2015.

[59] `https://www.xilinx.com/products/intellectual-property/ila.html`.

[60] (UG1023), SDAccel Development Environment - User Guide, vol. 1023, 2015.

[61] `http://www.khronos.org/opencl/`.

[62] `http://sine.ni.com/nips/cds/view/p/lang/en/nid/11834`.

[63] `http://www.ni.com/compactrio/`.

[64] `https://www.altera.com/products/fpga/stratix-series.html`.

[65] `https://www.altera.com/products/fpga/arria-series.html`.

[66] `https://www.altera.com/products/fpga/cyclone-series.html`.

[67] A. Davidson, A New FPGA Architecture and Leading-Edge FinFET Process Technology Promise to Meet Next-Generation System Requirements, no. June, pp. 110, 2015.

[68] https://www.altera.com/products/fpga/features/dsp/stratix-v-dsp-block.html.

[69] https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html.

[70] https://www.altera.com/products/design-software/embedded-software-developers/nios-ii-eds.html.

[71] https://www.altera.com/products/design-software/model---simulation/dsp-builder.html.

[72] https://www.altera.com/products/design-software/embedded-software-developers/opencl/overview.html.

[73] K. Pauwels, M. Tomasi, J. Diaz Alonso, E. Ros, and M. M. Van Hulle, A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features, Comput. IEEE Trans., vol. 61, no. 7, pp. 9991012, 2012.

[74] http://bluespec.com/.

[75] J. Bachrach, H. Vo, and K. Asanovic, Chisel Manual, pp. 112, 2012.

[76] http://www.xilinx.com/products/technology/dsp.html/.

[77] https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html.

[78] https://github.com/drichmond/riffa-development.

[79] J. Korinth, D. de la Chevallerie, and A. Koch, An Open-Source Tool Flow for the Composition of Reconfigurable Hardware Thread Pool Architectures, in 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, 2015, pp. 195198.

[80] https://maltevesper.github.io/JetStream/.

[81] M. Vesper, D. Koch, K. Vipin, and S. A. Fahmy, JetStream: An open-source high-performance PCI Express 3 streaming library for FPGA-to-Host and FPGA-to-FPGA communication, in 2016 26th International Conference on Field Programmable Logic and Applications (FPL), 2016, pp. 19.

[82] K. Ambrosch, M. Humenberger, W. Kubinger, and A. Steininger, SAD-based stereo matching using FPGAs, in Embedded Computer Vision, Springer, 2009, pp. 121138.

[83] C. Colodro-Conde, F. J. Toledo-Moreo, R. Toledo-Moreo, J. J. Martnez-lvarez, J. Garrigs Guerrero, and J. M. Ferrndez-Vicente, Evaluation of stereo correspondence algorithms and their implementation on FPGA, J. Syst. Archit., vol. 60, no. 1, pp. 2231, 2014.

[84] `http://www.cvlibs.net/datasets/kitti/`.

[85] `http://vision.middlebury.edu/stereo/eval3/`.

[86] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation, in 2010 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, 2010, pp. 93101.

[87] J. Hofmann, J. Korinth, and A. Koch, A Scalable High-Performance Hardware Architecture for Real-Time Stereo Vision by Semi-Global Matching, in 2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2016, pp. 845853.

[88] D. Zha, X. Jin, and T. Xiang, A real-time global stereo-matching on FPGA, Microprocess. Microsyst., vol. 47, pp. 419428, 2016.

[89] T. Kryjak, M. Komorkiewicz, and M. Gorgon, Real-time background generation and foreground object segmentation for high-definition colour video stream in FPGA device, J. Real-Time Image Process., vol. 9, no. 1, pp. 6177, Nov. 2012.

[90] B. Johnson, N. Thomas, and J. S. Rani, An FPGA Based High throughput Discrete Kalman Filter Architecture for Real-Time Image Denoising, 2017 30th Int. Conf. VLSI Des. 2017 16th Int. Conf. Embed. Syst., pp. 5560, 2017.

[91] Z. Chai, X. Shao, Y. Zhang, W. Yang, and Q. Wu, Accelerating image boundary detection by hardware parallelism, Microprocess. Microsyst., vol. 38, no. 5, pp. 458469, Jul. 2014.

[92] C. Li, Y. Bi, F. Marzani, and F. Yang, Fast FPGA prototyping for real-time image processing with very high-level synthesis, J. Real-Time Image Process., pp. 118, 2017.

[93] T. Shimizu, Y. Tomita, H. Matsumura, M. Sugimura, H. Yamasaki, D. Thach, T. Miyoshi, T. Baba, Y. Watanabe, and A. Ike, An FPGA-accelerated partial image matching engine for massive media data searching systems, in 2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits), 2016, vol. 2016Septe, pp. 12.

[94] L. Puglia, M. Vigliar, and G. Raiconi, Real-Time Low-Power FPGA Architecture for Stereo Vision, IEEE Trans. Circuits Syst. II Express Briefs, vol. 7747, no. c, pp. 11, 2017.

[95] F. Fons, M. Fons, and E. Cant, Run-time self-reconfigurable 2D convolver for adaptive image processing, Microelectronics J., vol. 42, no. 1, pp. 204217, Jan. 2011.

[96] R. Marzotto, P. Zoratti, D. Bagni, A. Colombari, and V. Murino, A real-time versatile roadway path extraction and tracking on an FPGA platform, Comput. Vis. Image Underst., vol. 114, no. 11, pp. 11641179, 2010.

[97] A. Hajirassouliha, T. P. B. Gamage, M. D. Parker, M. P. Nash, A. J. Taberner, and P. M. F. Nielsen, FPGA Implementation of 2D Cross-Correlation for Real-Time 3D Tracking of Deformable Surfaces, Int. Conf. Image Vis. Comput. New Zeal. (IVCNZ 2013), pp. 352357, 2013.

[98] Y. K. Lim, L. Kleeman, and T. Drummond, Algorithmic methodologies for FPGA-based vision, Mach. Vis. Appl., vol. 24, no. 6, pp. 11971211, Dec. 2012.

[99] S. Jin, J. Cho, X. Dai Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, FPGA design and implementation of a real-time stereo vision system, Circuits Syst. Video Technol. IEEE Trans., vol. 20, no. 1, pp. 1526, 2010.

[100] C. Georgoulas and I. Andreadis, FPGA based disparity map computation with vergence control, Microprocess. Microsyst., vol. 34, no. 7, pp. 259273, 2010.

[101] P. Zicari, S. Perri, P. Corsonello, and G. Cocorullo, Low-cost FPGA stereo vision system for real time disparity maps calculation, Microprocess. Microsyst., vol. 36, no. 4, pp. 281288, 2012.

[102] M. K. Jawed, Intelligent vision processor (PhD Thesis), The University of Auckland, 2013.

[103] C. Ttofis, S. Hadjitheophanous, A. S. Georghiades, and T. Theocharides, Edge-Directed Hardware Architecture for Real-Time Disparity Map Computation, IEEE Trans. Comput., vol. 62, no. 4, pp. 690704, 2013.

[104] C. Ttofis and T. Theocharides, High-quality real-time hardware stereo matching based on guided image filtering, in Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, 2014, pp. 16.

[105] J. S. Montrym, D. R. Baum, D. L. Dignam, and C. J. Migdal, InfiniteReality: A Real-Time Graphics System, Proc. 24th Annu. Conf. Comput. Graph. Interact. Tech., pp. 293302, 1997.

[106] `http://www.nvidia.com/object/gpu-applications.html`.

[107] NVidia, GPU Application Catalog, gpu-apps-catalog-mar14-digital-fnl-hr, 2014.

[108] E. Lindholm, J. Montrym, S. Oberman, and J. Montrym, NVidia Tesla: a unified graphics and computing architecture computing architecture., IEEE micro, pp. 3955, 2008.

[109] NVidia, Fermi Architecture, NVIDIA Fermi Compute Architecture Whitepaper, 2015.

[110] NVidia, Kepler Architecture, NVIDIA Kepler GK110 Architecture Whitepaper, 2015.

[111] `http://devblogs.nvidia.com/parallelforall/maxwell-most-advanced-cuda-gpu-ever-made/`.

[112] NVIDIA Tesla P100 (Whitepaper).

[113] `https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/`.

[114] NVIDIA, Whitepaper: NVIDIA GeForce GTX 980, pp. 132, 2014.

[115] D. Chaikalis, N. P. Sgouros, and D. Maroulis, A real-time FPGA architecture for 3D reconstruction from integral images, J. Vis. Commun. Image Represent., vol. 21, no. 1, pp. 916, 2010.

[116] Y.-H. Seo, H.-J. Choi, J.-S. Yoo, and D.-W. Kim, An architecture of a high-speed digital hologram generator based on FPGA, J. Syst. Archit., vol. 56, no. 1, pp. 2737, 2010.

[117] D. Cortie and J. Pillans, Using a custom-FPGA architecture to extend the scale of atomistic magnetic spin simulations, J. Comput. Sci., vol. 2, no. 3, pp. 279285, 2011.

[118] S. Jin, D. Kim, T. T. Nguyen, D. Kim, M. Kim, and J. W. Jeon, Design and Implementation of a Pipelined Datapath for High-Speed Face Detection Using FPGA, IEEE Trans. Ind. Informatics, vol. 8, no. 1, pp. 158167, Feb. 2012.

[119] P. Zicari, Efficient and high performance FPGA-based rectification architecture for stereo vision, Microprocess. Microsyst., vol. 37, no. 8, pp. 11441154, Nov. 2013.

[120] F. Stock, A. Koch, and D. Hildenbrand, FPGA-accelerated color edge detection using a Geometric-Algebra-to-Verilog compiler, in 2013 International Symposium on System on Chip (SoC), 2013, pp. 16.

[121] `http://www.nvidia.com/object/tesla-servers.html`.

[122] `http://www.nvidia.com/object/imaging_comp_vision.html`.

[123] J. Fang, A. L. Varbanescu, and H. Sips, A comprehensive performance comparison of CUDA and OpenCL, in Parallel Processing (ICPP), 2011 International Conference on, 2011, pp. 216225.

[124] `http://www.nvidia.com/object/cuda_home_new.html`.

[125] `https://developer.nvidia.com/cuda-gpus`.

[126] `https://developer.nvidia.com/cuda-tools-ecosystem`.

[127] `https://developer.nvidia.com/cuBLAS`.

[128] `https://developer.nvidia.com/cuFFT`.

[129] `https://developer.nvidia.com/NPP`.

[130] `https://developer.nvidia.com/nvidia-nsight-visual-studio-edition`.

[131] `https://developer.nvidia.com/nsight-eclipse-edition`.

[132] `http://www.mathworks.com.au/discovery/matlab-gpu.html`.

[133] `http://www.mathworks.com.au/products/parallel-computing/`.

[134] `http://www.nvidia.com/object/tesla-matlab-accelerations.html`.

[135] `http://arrayfire.com`.

[136] `http://www.arrayfire.com/docs/modules.htm`.

[137] B. Zhang, S. Xu, F. Zhang, Y. Bi, and L. Huang, Accelerating Matlab code using GPU: A review of tools and strategies, in Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2011 2nd International Conference on, 2011, pp. 18751878.

[138] Introduction to GPU Computing with LabVIEW: http://www.ni.com/white-paper/14077/en/.

[139] GPU Analysis Toolkit: http://sine.ni.com/nips/cds/view/p/lang/en/nid/210829.

[140] N. Weber and M. Goesele, Adaptive GPU Array Layout Auto-Tuning, in Proceedings of the ACM Workshop on Software Engineering Methods for Parallel and High Performance Applications - SEM4HPC 16, 2016, pp. 2128.

[141] NVIDIA, Cuda C Programming Guide Programming Guides.

[142] D. Castao-Dez, D. Moser, A. Schoenegger, S. Pruggnaller, and A. S. Frangakis, Performance evaluation of image processing algorithms on the GPU, J. Struct. Biol., vol. 164, no. 1, pp. 153160, 2008.

[143] A. Wedel, T. Brox, T. Vaudrey, C. Rabe, U. Franke, and D. Cremers, Stereoscopic scene flow computation for 3D motion understanding, Int. J. Comput. Vis., vol. 95, no. 1, pp. 2951, Oct. 2011.

[144] M. Gong, Y. Zhang, and Y.-H. Yang, Near-real-time stereo matching with slanted surface modeling and sub-pixel accuracy, Pattern Recognit., vol. 44, no. 1011, pp. 27012710, Oct. 2011.

[145] W. Griffin, Y. Wang, D. Berrios, and M. Olano, Real-Time GPU Surface Curvature Estimation on Deforming Meshes and Volumetric Data Sets, IEEE Trans. Vis. Comput. Graph., vol. 18, no. 10, pp. 16031613, 2012.

[146] J. I. Agulleiro, F. Vzquez, E. M. Garzn, and J. J. Fernndez, Hybrid computing: CPU+GPU co-processing and its application to tomographic reconstruction., Ultramicroscopy, vol. 115, pp. 10914, Apr. 2012.

[147] N. Karpinsky and S. Zhang, Holovideo: Real-time 3D range video encoding and decoding on GPU, Opt. Lasers Eng., vol. 50, no. 2, pp. 280286, Feb. 2012.

[148] N. Zhang, H. Wang, and J. Cr, A Near Real-Time Color Stereo Matching Method for GPU, Third Int. Conf. Adv. Commun. Comput. (INFOCOMP 2013), pp. 2732, 2013.

[149] R. Kalarot, Real time stereo on GPU with application to precision 3D tracking, The University of Auckland, 2014.

[150] Y. Zhang, P. Zhou, Y. Ren, and Z. Zou, GPU-accelerated large-size VHR images registration via coarse-to-fine matching, Comput. Geosci., vol. 66, pp. 5465, May 2014.

[151] M. Gates, M. T. Heath, and J. Lambros, High-performance hybrid CPU and GPU parallel algorithm for digital volume correlation, Int. J. High Perform. Comput. Appl., vol. 29, no. 1, pp. 92106, Feb. 2015.

[152] D. Akgn and P. Erdomu, GPU accelerated training of image convolution filter weights using genetic algorithms, Appl. Soft Comput., vol. 30, pp. 585594, 2015.

[153] L. Zhang, T. Wang, Z. Jiang, Q. Kemao, Y. Liu, Z. Liu, L. Tang, and S. Dong, High accuracy digital image correlation powered by GPU-based parallel computing, Opt. Lasers Eng., vol. 69, pp. 712, 2015.

[154] M. Lastra, J. Carabao, P. D. Gutirrez, J. M. Bentez, and F. Herrera, Fast fingerprint identification using GPUs, Inf. Sci. (Ny)., vol. 301, pp. 195214, 2015.

[155] A. Li, A. Kumar, Y. Ha, and H. Corporaal, Correlation ratio based volume image registration on GPUs, Microprocess. Microsyst., vol. 39, no. 8, pp. 9981011, Nov. 2015.

[156] J. Jiang and T. Fogal, A Lightweight H.264-based Hardware Accelerated Image Compression Library, pp. 99100, 2016.

[157] R. Vergne, P. Barla, G. Bonneau, R. W. Fleming, and J. L. Universit, Flow-Guided Warping for Image-Based Shape Manipulation, vol. 35, no. 4, pp. 112, 2016.

[158] F. Garcia, L. Ubeda-Medina, and J. Grajal, Real-time GPU-based image processing for a 3-D THz radar, IEEE Trans. Parallel Distrib. Syst., vol. 9219, no. c, pp. 113, 2017.

[159] A. R. Brodtkorb, T. R. Hagen, and M. L. Stra, Graphics processing unit (GPU) programming strategies and trends in GPU computing, J. Parallel Distrib. Comput., vol. 73, no. 1, pp. 413, 2013.

[160] Titan: https://www.olcf.ornl.gov/titan/.

[161] S. Asano, T. Maruyama, and Y. Yamaguchi, Performance Comparison of FPGA, GPU and CPU in Image Processing, 19th Int. Conf. F. Program. Log. Appl. (FPL 2009), pp. 126131, 2009.

[162] W. Liu, P. P. Pokharel, and J. C. Principe, Correntropy: Properties and Applications in Non-Gaussian Signal Processing, vol. 55, no. 11, pp. 52865298, 2007.

[163] J. Fowers, G. Brown, J. Wernsing, and G. Stitt, A performance and energy comparison of convolution on GPUs, FPGAs, and multicore processors, ACM Trans. Archit. Code Optim., vol. 9, no. 4, pp. 121, Jan. 2013.

[164] J. Choi and R. A. Rutenbar, Video-Rate Stereo Matching Using Markov Random Field TRW-S Inference on a Hybrid CPU+FPGA Computing Platform, IEEE Trans. Circuits Syst. Video Technol., vol. 26, no. 2, pp. 385398, 2016.

[165] M. Birk, M. Balzer, N. V. Ruiter, and J. Becker, Evaluation of performance and architectural efficiency of FPGAs and GPUs in the 40 and 28nm generations for algorithms in 3D ultrasound computer tomography, Comput. Electr. Eng., vol. 40, no. 4, pp. 11711185, May 2014.

[166] https://opencv.org/

[167] http://pointclouds.org/

[168] https://opencv.org/platforms/cuda.html

[169] https://docs.opencv.org/2.4/modules/ocl/doc/ocl.html

[170] http://pointclouds.org/documentation/tutorials/gpu_install.php

[171] https://www.raspberrypi.org/products/

[172] https://en.wikipedia.org/wiki/Raspberry_Pi

[173] https://www.raspberrypi.org/blog/facial-recognition-opencv-on-the-camera-board/

[174] https://link.springer.com/article/10.1007/s11265-017-1267-1

[175] https://github.com/DT42/BerryNet

[176] M. Daga, A. M. Aji, and W. C. Feng, On the efficacy of a fused CPU+GPU processor (or APU) for parallel computing, Proc. - 2011 Symp. Appl. Accel. High-Performance Comput. SAAHPC 2011, pp. 141149, 2011.

[177] https://newsroom.intel.com/news-releases/intel-completes-acquisition-of-altera/

[178] https://www.altera.com/solutions/acceleration-hub/acceleration-stack.html

[179] https://www.altera.com/products/boards_and_kits/dev-kits/altera/acceleration-card-arria-10-gx.html

[180] https://01.org/OPAE

[181] Y. LeCun, K. Kavukcuoglu, and C. Farabet, Convolutional networks and applications in vision, ISCAS 2010 - 2010 IEEE Int. Symp. Circuits Syst. Nano-Bio Circuit Fabr. Syst., pp. 253256, 2010.

[182] Y. Lecun, Y. Bengio, and G. Hinton, Deep learning, Nature, vol. 521, no. 7553, pp. 436444, 2015.

[183] A. Krizhevsky, I. Sutskever, and G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, Adv. Neural Inf. Process. Syst., pp. 19, 2012.

[184] A. Esteva et al., Dermatologist-level classification of skin cancer with deep neural networks, Nature, vol. 542, no. 7639, pp. 115118, 2017.

[185] T. Orlowska-Kowalska and M. Kaminski, FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system, IEEE Trans. Ind. Informatics, vol. 7, no. 3, pp. 436445, 2011.

[186] A. Caulfield et al., A Cloud-Scale Acceleration Architecture, 2016 49th Annu. IEEE/ACM Int. Symp. Microarchitecture, pp. 11, 2017.

[187] M. Gharbi, J. Chen, J. T. Barron, S. W. Hasinoff, and F. Durand, Deep Bilateral Learning for Real-Time Image Enhancement, vol. 36, no. 4, 2017.

[188] P. H. Pham, D. Jelaca, C. Farabet, B. Martini, Y. LeCun, and E. Culurciello, NeuFlow: Dataflow vision processing system-on-a-chip, Midwest Symp. Circuits Syst., pp. 10441047, 2012.

[189] T. Chen et al., DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning, Proc. 19th Int. Conf. Archit. Support Program. Lang. Oper. Syst. - ASPLOS 14, pp. 269284, 2014.

[190] Z. Du et al., ShiDianNao: Shifting Vision Processing Closer to the Sensor, Isca, pp. 92104, 2015.

[191] N. P. Jouppi et al., In-Datacenter Performance Analysis of a Tensor Processing Unit, ACM SIGARCH Comput. Archit. News, vol. 45, no. 2, pp. 112, 2017.