12

Towards the Optimal Hardware Architecture for Computer Vision

Alejandro Nieto, David López Vilarino and Víctor Brea Sánchez
Centro de Investigación en Tecnoloxías da Información (CITIUS)
University of Santiago de Compostela
Spain

1. Introduction

Computer Vision systems are experiencing a large increase in both range of applications and market sales (BCC Research, 2010). From industry to entertainment, Computer Vision systems are becoming more and more relevant. The research community is making a big effort to develop systems able to handle complex scenes focusing on the accuracy and the robustness of the results. New algorithms provide more advanced and comprehensive analysis of the images, expanding the set of tools to implement applications (Szeliski, 2010).

Although new algorithms allow to solve new problems and to approach complex situations with a high degree of accuracy, not all the algorithms are adequate to be deployed in industrial systems. Parameters like power consumption, integration with other system modules, cost and performance limit the range of suitable platforms. In most cases, the algorithms must be adapted to achieve a trade-off solution and to take advantage of the target platform.

Conventional PC-based systems are constantly improving performance but their use is still limited to areas where portability, power consumption and integration are not critical. In case of highly complex algorithms, with an irregular execution flow, complex data representation and elaborated patterns to access to data, a significant gain is not achieved when moving to an *ad hoc* hardware design. In this case, a high-end CPU and a GPU with general-purpose capabilities (GPGPU) is a flexible and very powerful combination that will outperform other options (Castano-Díez et al., 2008).

However, when a conventional system does not meet the requirements of the application, a more aggressive planning is needed. For instance, migrating the application to a dedicated device such as a DSP, an FPGA or a custom chip (Shirvaikar & Bushnaq, 2009). At this point, the designers have to consider alternatives as to reduce operating range, accuracy and robustness of the results, or to remove expensive operations in order to simplify the hardware that will be implemented (Kolsch & Butner, 2009). PC-based systems enable a great flexibility at cost of performance so pure software-based algorithms hardly match pure hardware implementations. This is a serious limitation because it can compromise the efficiency of the application. This is the reason why the industry is making great efforts to develop novel architectures that enable greater flexibility to adapt the algorithms without compromising the quality of the results.

Computer Vision applications are often divided into three stages: low, medium and high level processing. Low level operations are quite simple and repetitive but applied over a large set of data with a very reduced data dependent program flow, so massive parallelism is essential for performance. In the medium level stage, temporal and task parallelism is key as the data set is smaller and the program flow is quite data-dependent. High level operations are performed over complex data representations but reduced data sets and with high precision as requirement. An efficient Computer Vision system must deal with all these stages. The selection of the computation model will determine the performance of the device in each one of the stages. So, which is the optimal paradigm for such applications? Increase the number of concurrent instructions? Increase the number of data elements processed simultaneously? A combination of both?

Taking into account the application requirements, a suitable platform to build the system must be selected. Besides general parameters (performance, cost and integration), there is a set of factors that restricts the computing paradigms and the devices that can be selected. For instance, a critical parameter is the way in which the data are transferred to the device because I/O operations are one of the bottlenecks in high-performance systems. Data type and representation will affect the computational units. The program flow will constrain the inner connections between these units and the storage elements.

This chapter addresses an analysis of different computing paradigms and platforms oriented to image processing. Previously, a representative set of Computer Vision algorithms covering the three levels of processing is reviewed. This study will lead us to observe the algorithms in terms of a set of common characteristics: operations, data type, program flow, etc. This is critical to design new hardware architectures in order to maximize performance. The analysis from the hardware point of view will highlight the best features of the most used computing paradigms in order to establish a relationship between the type of operation, data, programming model and hardware architecture. An efficient architecture for Computer Vision must combine all the selected features. The analysis of the characteristics of the different algorithms will lead us naturally to an optimized general-purpose hardware architecture for Computer Vision.

2. The problem of computer vision

Traditionally, Computer Vision (CV) applications include building blocks from three computation levels: low-, mid- and high-level vision computing. The type of operations, the data representation and the flow execution of programs depend deeply on the considered level of this hierarchy. Nevertheless, current CV-algorithms are composed of many different processing steps regarding the type of data and the way these are computed which makes difficult to classify them only in one subgroup. Following, a rather rough classification of widely used CV-algorithms is made, keeping in mind the data domain and the complexity of the involved operations.

2.1 Low-level vision

After image acquisition, some preprocessing steps are often required. These are intended to provide reliable input data for subsequent computing stages. Some typical operations are

noise reduction, color balancing, geometrical transformation, etc. Most of these operations are based on point or near-neighborhood operations. Point operations are performed at pixel-level in such a way that the output only depends on the value of any individual pixels from one or several input images. With this type of operation it is possible to modify the pixel intensity to enhance parts of the image, by increasing contrast or brightness. Equally, simple pixel-to-pixel arithmetic and Boolean operations also enable the construction of operators as *alpha blending*, for image combination or color space conversion. Neighborhood operations take also into account the value of adjacent pixels. This operation type is the basis of filtering, binary morphology or geometric transformation. They are characterized by simple operations, typically combining weighted sums, Boolean and thresholding processing steps.

After preprocessing stages, useful information has to be extracted from the resulting images. Common operations are edge detection, feature extraction or image segmentation. Edges are usually defined as step discontinuities in the image signal so finding local maxima in the derivative of the image or zero-crossings in the second derivative are suitable to detect boundaries. Both tasks are usually performed by the convolution of the input image with spatial filtering masks that approximate a first or second derivative operator.

Feature points are widely used for subsequent computing steps in multiple CV-applications. Basically, a feature represents a point in the image which differs from its neighborhood. One of the benefits of local features is the robustness against occlusion and the ability to manage geometric deformations between images when dealing with viewpoint changes. In addition, they improve accuracy when, in the same scene, objects are at different planes, (i.e. at different scales). One of the most popular techniques is that proposed by Harris (Harris & Stephens, 1988) to detect corners. It is widely used due to its strong invariance to rotation, image noise and no large illumination changes. It uses the local auto-correlation function, which describes the gradient distribution in a local neighborhood of each image point to detect the location of the corners. Using the locally averaged moment matrix from the image gradients, corners will be located at the maximum values. Another frequently used technique is the Scale-Invariant Feature Transform (SIFT) (Lowe, 2004). SIFT localizes extrema both in space and scale. Using the *Difference of Gaussians* as scale-space function, the images are filtered with Gaussian kernels of different sizes (scales). This is performed for different image sizes (octaves). The response of each filter is subtracted from the immediately following in the same octave. The interest points are scale-space extrema so local maxima and minima are extracted by comparing the neighborhood points in the same, the previous and the posterior scales. To improve accuracy, a sub-pixel approximation step is done, interpolating the location of the feature inside the scale-space structure. The number of octaves and scales can be tuned to meet the system requirements. SIFT provides invariance against scale, orientation and affine distortion, as well as partial occlusion and illumination changes. Other algorithms were proposed to improve accuracy or performance like, the *Speeded Up Robust Features* (SURF) (Bay et al., 2006) or the *Gradient Location and Orientation Histogram* (GLOH) (Mikolajczyk & Schmid, 2005). This kind of detectors are quite complex and their performance can be low even using custom hardware. For this reason, less reliable algorithms are still in use, as Harris Corner Detector, FAST (Trajkovi & Hedley, 1998) or the Smallest Univalue Segment Assimilating Nucleus (SUSAN) (Smith & Brady, 1997) corner detectors, because of their efficiency under controlled situations and their low hardware requirements.

Segmentation refers to the process of separating the data into several sets according to certain characteristics. There are several techniques to carry out this task, either based on boundaries or regions (Pal & Pal, 1993) (Haralick & Shapiro, 1985). Nevertheless, most of them rely on near-neighborhood operations. Particular attention deserves the clustering methods like the popular k-means which partitions the data set into several clusters according to a proximity criterion defined by a distance function. These methods are not restricted to image data. N-dimensional sets of abstract data can also be partitioned. Furthermore, information about the scene or domain can be introduced (number and characteristics of the target clusters). Therefore, they might be classified either as a low or mid-level vision stage.

2.2 Mid-level vision

Mid-level CV stages usually operate on images from previous processing steps, often binary images, and produce a lower amount of data but with a higher concentration of information. Some common operations are object classification and scene reconstruction.

One of the goals of Computer Vision is to recognize objects in a scene. Based on object location, pose or 2D/3D spatial relations between the objects, the algorithms have to be able to analyze the scene and its content. This involves issues such as dealing with object models, classifiers and the ability to integrate new information in the models. In the literature, a large amount of techniques can be found, usually classified as global methods, more intended for object detection and local feature-based methods for object recognition. In all of them good image registration is essential for both accuracy and performance (Zitova & Flusser, 2003). As for the global methods, common techniques are based on *template-based matching*, which employs a convolution mask or template to measure the similarity between an object patch and the template. In this sense, *normalized cross-correlation* (NCC), *sum of squared differences* (SSD) or *sum of absolute differences* (SAD) are widely used. As for local methods, local feature descriptors play an important role. Roughly speaking, a descriptor is an abstract characterization of a feature point based on its environment. One of the most popular techniques is the proposed in second part of the SIFT algorithm based on stacked orientation histograms which associate a high dimension vector to each keypoint. In order to reduce the amount of false positives and negatives during the matching stage the search area is limited by using strategies like the *nearest neighbor search* (NNS) which attempts to find the nearest points of a given one in a vector space. An *indexing structure* allows to search for features near a given feature rapidly. This is the case of the *K-dimensional trees* , which organize points in a k-dimensional space in such a way that each node has at most two child nodes.

Scene reconstruction consists of the generation of scene models starting from their parts. There are different techniques to reconstruct one or several objects in a scene. To build a 3D model, coordinates of scene points have to be calculated from the objects. If the location of the camera is known, 3D coordinates of a scene point can be determined from its projection on image planes of different viewpoints. The whole process starts with feature extraction and matching. Using geometric consistency tests it is possible to eliminate wrong matches. There are different solutions to estimate the fundamental matrix, as the *RANdom SAmple Consensus* (RANSAC). Once the matches between images are consistent, camera pose and scene geometry is reconstructed using *Structure from Motion* methods and refined with *Bundle adjustment* techniques (Triggs et al., 2000).

2.3 High-level vision

The high-level stage often starts from an abstract representation of the information. This stage is highly application dependent but due to the variety of operations, data structures, memory access patterns and program flow characteristics are often only compatible with a general purpose processor. High-level processing is characterized by the use of a small set of data to represent knowledge about the application domain. More complex data structures are needed to store and process this information efficiently, making the operations and the memory access patterns more elaborated. This, together with the inherent complexity of decision making, makes the program flow very variable.

Robust pattern recognition, object identification, complex decision making or system adaptation are some of the benefits of integrating Artificial Intelligence methods with Computer Vision. Otherwise, the system would be limited to a predetermined set of actions. Machine learning makes computers capable of improving automatically with experience. This way it is possible to generalize the behavior from unstructured information with techniques such as neural networks, decision trees, genetic algorithms, regression models or support vector machines. Machine learning has emerged as a key component of intelligent computer vision systems, contributing to a better understanding of complex images (Viola & Jones, 2001) (Oliver et al., 2000).

Data mining is the process of analyzing data using a set of statistical techniques in order to summarize into segments of useful information. This makes possible analyze data from different dimensions or angles, categorizing and summarizing the identified relationships, making evident hidden relationships or patterns between events. On the contrary of Machine learning, data mining focuses on discovering hidden patterns instead of generalizing known patterns using the new data.

It is very difficult to establish a classification of tasks and operations for high-level processing. Some of the performed tasks fall within the scope of the measurement of application specific parameters such as size and pose of objects, fault detection and monitoring specific events such as traffic situations, for example. The algorithms and technologies are very diverse and most of them lie in statistical analysis and artificial intelligence domains.

As it was previously mentioned some tasks that initially fit into the low or medium level stages due to its context actually are more like high-level operations by the type of operations performed. This is related with the commonly used bottom-up image analysis, which starts from raw data to extend the knowledge of the scene. However, new approaches include feedback to perform top-down analysis. This way, low- and mid-level stages can be controlled with general knowledge of the image, improving the results.

3. Computing platforms

Given the wide range of algorithms and applications of Computer Vision, it is clear that it does not exist a unique computing paradigm or an optimal hardware platform. The type of operations, the complexity of data structures and especially the data access patterns greatly determine parameters such as the range of application, performance, power consumption or cost. This section presents some of the most prominent platforms in image processing, focusing on their strengths and weaknesses.

3.1 Computing paradigms

The Flynn's taxonomy (Flynn, 1972) classifies the computer architectures in four big groups according to the number of concurrent instructions and data sets processed. Image processing tasks perform more or less efficiently depending on the selected paradigm. In order to develop a Computer Vision application it is crucial to exploit the spatial (data) or temporal (task) parallelism to meet trade-offs among performance, power consumption or cost.

SISD

Single Instruction Single Data (SISD) refers to the conventional computing model. A single processing unit executes a sequence of instructions on a unique data stream. Most modern computers are placed under this category and, although only one processor and one memory element are present, those which are able to pipeline their data-path are generally classified under the SISD category as they are still *serial computers*.

This paradigm performs better when spatial and temporal parallelism are hard to exploit. As seen previously, high-level image processing fit the SISD paradigm because most tasks are sequential, with a complex program flow and strong dependences between data. As processing is done sequentially, most optimizations aims to enhance the access between the memory and the arithmetic unit. Memory and processor speed are the main constraints. Furthermore, some kind of parallelism can be exploited when pipelining the data-path. Data allocation, pre-fetching and reducing stalls in the pipeline are some of the possible optimizations.

SIMD

SIMD (*Single Instruction Multiple Data*) computers have an unique control unit and multiple processing units. This control unit sends the same instruction to all processing units, which operate over different data streams. This paradigm focuses on exploiting the spatial parallelism. It is also possible to pipeline the data-path or to employ several memories to store the data in order to increase the bandwidth. SIMD computers are commonly specific-purpose, intended to speed-up certain critical tasks.

One of the drawbacks of SIMD machines is data transference. A network is required to both supply data to each processing unit and share data among them. Its size grows with the number of connected nodes so SIMD architectures have a practical limitation. Another restriction is data alignment when gathering and scattering data into SIMD units. This results in a reduction of flexibility in practical implementations. It is needed to determine the correct memory addresses and reordering data adequately, affecting performance. In addition, as this paradigm exploits spatial parallelism, program flow is heavily limited because all units execute the same instruction. Additional operations are needed to enable at least simple flow control tasks.

Low-level image processing benefits greatly of SIMD units. As it was described previously, most operations are quite simple but repetitive over the whole set of data. In addition, certain tasks of mid and high-level processing can also take advantage of SIMD units when using in conjunction with others paradigms. The simplicity of the arithmetic units and the

memory access patterns make feasible to design efficient units, which is crucial to increase the parallelism. Memory bandwidth and data distribution among the processors is also key for performance.

Two types of SIMD accelerators can be distinguished based on the number of processing units: fine-grain and coarse-grain processors, where the major difference is the number of processing units. While the first includes a large amount of very simple processors with a rigid network, the second features major flexibility although with a much lower parallelism. When using in low-level image processing, fine-grain processor arrays match with massively parallel operations such as filters or morphological operations. Using a processor-per-pixel scheme and local communications, neighborhood operations are completed in just a few instructions. On the contrary, when the parallelism level is lower a configuration as *vector processor* is usually preferable. By reducing communications and increasing core complexity, they are much more flexible and efficient not only for low-level operations but also for other processing stages.

MIMD

Multiple Instruction Multiple Data (MIMD) refers to architectures where several data streams are processed using multiple instruction streams. MIMD architectures have several processing units executing different instructions to exploit task parallelism. Processors perform independently and asynchronously. MIMD systems are classified depending on the memory architecture.

In *Shared Memory Systems*, all processors have access to an unique memory. Connection hierarchy and latencies are the same for all processors. This scheme eases data transference among processors although simultaneous access must be taken into account to avoid data hazards. Scalability is also reduced because it is hard to increase the memory bandwidth at the same rate as the number of processors.

If each processor has its own and private memory it is possible to upscale more easily as memory and processors are packet as a unit. This scheme is known as *Distributed Memory System*. In addition, local memory access is usually faster. The major disadvantage is the access to data which are located outside the private memory because dedicated buses and a *message passing system* to communicate with the processors are needed. This can result in high access times and an increase of hardware requirements.

In a *Distributed Shared Memory System*, the processors have access to a common shared memory but without a shared channel. Each processor is provided with local memory which is interconnected with other processors through a high-speed channel. All processors can access to different banks a global address space. Access to memory is done under the schemes as *Non-Uniform Memory Access* (NUMA), which takes less time to access the local memory than to access the remote memory of other processor. This way scalability is not compromised.

Very long Instruction Word (VLIW) and *superscalar* architectures are also classified within MIMD paradigm because they exploit instruction-level parallelism, executing multiple instructions in parallel. *Pipelining* also executes multiple instructions but splitting them in independent steps to keep all the units of the processor working at a time.

Mid-level image processing and some operations of the other processing levels of Computer Vision can exploit MIMD processors. Operations are relatively simple, with data-dependent program flow. Temporal and task parallelism are easier to exploit than spatial parallelism although a reduced degree is usually present in this type of algorithms. Each MIMD processing element can include SIMD units. This way it is possible to process complex tasks more efficiently, from kernel operations as when pre-processing images concurrently in multi-view vision systems to high level tasks as multiple object recognition and tracking. In general, any set of tasks with weak dependences between them to reduce internal communications can take advantage of this paradigm.

MISD

There is one more paradigm, *Multiple Instruction Simple Data* (MISD), which achieves higher parallelism than SISD executing different instructions over the same data set employing several computing units.

Systolic arrays are regular n-dimensional arrays of simple cores with nearest-neighbors interconnections. Each core operates on the input data and shares the result to its neighbor, flowing the data synchronously usually with different flow in different directions. They are employed for tasks such as image filtering or matrix multiplication. Pipelined architectures belong to this type, as they are considered an one-dimensional systolic array, but they are commonly considered an improved version of the other aforementioned paradigms.

This paradigm is rarely used for Computer Vision as the other paradigms match better and offer higher performance and flexibility when dealing with real Computer Vision problems.

Remarks

Low-level Computer Vision entails the largest processing times in most applications. Data sets are usually very large and the kind of operations simple and repetitive. However, operations are inherently massively parallel and the data access patterns are regular. It is feasible to exploit these features to design very optimized SIMD custom hardware accelerators or to migrate the algorithms to existing hardware.

It is harder to extract parallelism in the mid-level stage because operations involve more complex data-flow. In addition, the data set is smaller so the benefits of including dedicated units to speed-up the computation are lower than expected. Despite this, hybrid processors (SIMD-MIMD) able to exploit both spatial and temporal parallelism can overcome this limitation.

Finally, the amount of data involved in the high-level stage is usually small so it is rarely necessary to sacrifice precision in order to get better performance. Moreover, unlike in previous stages, the disparity in the type of data makes the use of floating-point often a requirement. Another characteristic of this stage is the program flow, far more complex, which may even consume more computation time than the arithmetic operations. The kind of computation performed at this stage is so varied that the best option is often a general purpose SISD processor.

3.2 Current devices

There are different possibilities to implement the aforementioned computation paradigms. There is not an unique and direct correspondence between a paradigm and its hardware implementation. On the one hand, it can be designed a dedicated hardware which follows the original conception. On the other hand, the paradigm can be emulated both in hardware or software.

Microprocessors

Microprocessors, SISD machines, are the most straightforward devices to develop a Computer Vision application. Their main advantage is their versatility, the ability to perform very different tasks for a low cost. They can perform any type of data processing, although its efficiency, measured in parameters such as cost, power or integration capabilities, is not always optimal because of their general-purpose condition. The large variety of available technologies, libraries, support and programs cut down the *cold start*, enabling to get the system ready for development in a short time. Developers can focus on the problem itself instead of technical issues (Bradski & Kaehler, 2008).

Basically, they are composed by a main memory and a processing unit which includes the arithmetic and the control modules. From this basic structure more optimized microprocessors can be designed. From caches to cut down the memory access times, tightly coupled high-speed memory controllers or specialized units for critical tasks, the variability of architectures is as large as the amount of fields in the market (Singhal, 2008). However, despite the evolution of the industry pure SISD microprocessors do not offer adequate performance for a large set of tasks. That is why a wide range of accelerator modules have been included, as specific-purpose arithmetic units and sets of instructions or co-processors. The inclusion of SIMD units is decisive for tasks such as video encoding and decoding, but any data-intensive algorithm can take advantage of them (Franchetti et al., 2005).

As it will be discussed later, the advances in the semiconductor industry allows to increase the integration density so it is possible to include more processing power on the same Silicon area. This has led to abandon the race for speed (to increase the working frequency) to more efficient systems where energy consumption is vital and parallelism is the way to overcome the limitations of Moore's Law (Naffziger, 2009). Most of modern processors are multi-core and the number of cores is expected to grow in the near future. Programming languages and techniques as well as image processing algorithms have to be adapted to this new reality (Chapman, 2007).

Microprocessors are employed in a wide range of applications, from developing and testing algorithms such as autonomous driving (Urmson et al., 2008) to final platforms as medical image reconstruction (Chu & Chen, 2009). Even its use in restrictive stand-alone devices is also viable such as autonomous flight (Meier et al., 2011). Microprocessors stand out in high-level tasks, as the latest stages of image retrieval (Deselaers et al., 2008) and scene understanding (Li et al., 2009), where handling image databases, storing and communicating data are fairly complex to implement them in specific purpose devices. Video surveillance tasks can take advantage of these features for image processing (Ahmed & Terada, 2010) and event control in complex distributed systems (Chen et al., 2008).

Mobile processors have become a benchmark in innovation and development after the explosion of the mobile market. As discussed below, they integrate several general purpose cores, graphics processing units and other co-processors on a single chip keeping power consumption very low. The applications they can address are increasingly complex (Taylor et al., 2009) (Wither et al., 2011) (Ren et al., 2010).

Graphics Processing Units

A *Graphics Processing Unit* (GPU) is a specialized co-processor for graphic processing to reduce the workload of the main microprocessor in PCs. They implement highly optimized graphic operations or *primitives*. Current GPUs provide a high processing power and exploit the massively spatial parallelism of these operations. Because of their specialization, they can perform operations faster than a modern microprocessor even at lower clock rates. GPUs have hundreds of independent processing units working on floating point data. Memory access is critical to avoid processing downtimes, both in bandwidth and speed.

Their high processing power makes GPUs an attractive device for non related graphic tasks. *General Purpose GPU* (GPGPU) is a technique to perform general computation not related to graphics on these devices (Owens et al., 2007) (Che et al., 2008). This makes possible to use its specialized and limited pipeline to perform complex operations over complex data types. In addition, it eases memory management and data access. Flow control, as looping or branching, is restricted as in other SIMD processors. Modern GPUs architectures as (Seiler et al., 2008) or (Lindholm et al., 2008) have added support for these operations although slightly penalizing throughput. Word size is also a limitation in GPGPU techniques. It was reduced to increase the integration density as graphic operations do not usually require high precision. However, since this was a serious limitation for scientific applications, large word-sizes support was added later (Thall, 2006).

GPUs are effective when using *stream processing*, a paradigm related to SIMD (Rixner et al., 1998). A set of operations (*kernel*) is applied to each element of a set of data (*stream*). The flexibility is reduced to increase the parallelism and to lower the communication requirements when involving hundreds of processing elements. Otherwise, providing data to hundreds of processors would be a bottleneck. Processors are usually pipelined in a way that results pass from one arithmetic unit to the next one. This way, the locality and concurrency are better exploited, reducing communication requirements because most of the data are stored on-chip.

The use of GPUs to speed-up the computing has greatly increased, specially after the optimization of libraries and functions that mask low-level technical difficulties. Most image processing kernels and algorithms were adapted to work in GPGPUs, obtaining significant improvements. Basic image processing (Podlozhnyuk, 2007), FFT transforms (Govindaraju & Manocha, 2007), feature extraction (Heymann et al., 2007) or stereo-vision (Lui & Jarvis, 2010), all of them computationally expensive, benefit greatly of the massively parallelism of GPU. They can be used also to emulate other computing paradigms frequently used in low level vision (Dolan & DeSouza, 2009). However, some authors argue that the gap between GPUs and CPUs is not as large as it seems if key optimizations are carried out (Lee et al., 2010).

Digital Signal Processors

A *Digital Signal Processor* (DSP) is a microprocessor-based system with a set of instructions and hardware optimized for intensive data applications. They are specially useful for real-time processing of analog signals but they offer a high throughput in any data intensive application. DSP market is well established and offers a large range of devices, optimized for each particular task (Schneiderman, 2010). Apart from attached processors, to assist a general purpose host microprocessor, DSPs are often used in embedded systems including all necessary elements and software.

They are able to exploit parallelism both in instruction execution and data processing. In a von Neumann architecture, instruction and data share the same memory space. However, DSP applications usually require several memory accesses to read and write data per instruction. To exploit concurrency, many DSPs are based on a Harvard architecture, with separate memories for data and instructions. Many modern devices are based on *Very Long Instruction Word* (VLIW) architectures so they are able to execute several instructions simultaneously (Lin et al., 2008). Compilers are fundamental to find the parallelism in the instructions and a large improvement can be obtained after an efficient placement of data and programs in memory. Superscalar processors and pipelined data-paths also improve the overall performance although this is done by hardware. To operate, they include specialized hardware for intense calculation such as *multiply-accumulate* operation, which is able to produce a result in one clock cycle. Although many DSPs have floating-point arithmetic units, fixed-point units fit better in battery-power devices. Formerly, floating-point units were slower and more expensive but this gap is getting smaller and smaller. They also include zero-overhead looping, rounding and saturated arithmetic or dedicated units for address management (Talavera et al., 2008; Texas Instruments, 2002; Wang et al., 2010).

DSPs are usually designed for intensive data processing so performance can be penalized in mixed tasks. However, current all-in-one devices are able to handle complete applications efficiently. In addition, DSPs are not only available as independent devices but also as part of integrated circuits such as FPGAs or SoCs.

DSPs have a large tradition on image processing tasks. As optimized versions of conventional processors, they were used to accelerate the most expensive operations. These operations were related mainly with low-level image processing (Baumgartner et al., 2009), where parallelism and data access enable a large performance increase. Stereo vision (Lin & Chiu, 2008), Fourier transform (Sun & Yu, 2009) or video matching and tracking (Shah et al., 2008) are some samples. However, higher level algorithms also suit for DSPs, specially in industrial tasks (Suzuki et al., 2007) (Neri et al., 2005). Nowadays, DSPs are able to handle large sets of operations efficiently both for co-processing (Rinnerthaler et al., 2007) or standalone (Bramberger & Rinner, 2004).

Field Programmable Gate Arrays

A *Field Programmable Gate Array* (FPGA) is a device with user-programmable hardware logic. It is made of a large set of *logic cells* connected together through a network. Both elements are programmable in such a way that the logic cells emulate combinational functions and the

network permits to join them to build more complex functions. In addition, the *program* can be rewritten as many times as needed.

The main advantage of FPGAs is their high density of interconnections between cells, which provides a very high flexibility. This network has a complex hierarchy with optimizations for specific functions. It provides specialized lines to propagate clock or reset signals across all the FPGA or to build buses with high *fan-out* within acceptable time delays. With these very basic elements it is possible to build highly complex modules as arithmetic units, controllers or even embedded microprocessors. In addition to simple elements such as 1-bit *flip-flops*, FPGAs have a large set of embedded modules. Large memory elements, DSP arithmetic units, networking and memory controllers or even embedded microprocessors are available on modern FPGAs (Leong, 2008).

These devices are an excellent mechanism to build proof-of-concept prototypes. On the one hand due to their flexibility any computing paradigm can be implemented restricted mainly for the number of available cells. On the other hand, thanks to the re-programmability it is possible to debug and test on real hardware. Even more, nowadays some final products are implemented exclusively on FPGAs instead of migrating the design to custom integrated circuits. The achieved performance can be tens of times higher with lower power consumption than standard PC-based approaches (Fasang, 2009). FPGAs are widely employed as co-processors in personal computers such as GPUs or as accelerators in specific purpose devices as high-capacity network systems (Djordjevic et al., 2009) or high-performance computing (Craven & Athanas, 2007). Nowadays, it is possible to embed full systems on a single FPGA.

One of the major disadvantages of FPGAs is the set-up time, still higher than in pure-software approaches. Traditional FPGA programming is done with HDL languages, forcing to design at a very low level. High-level languages, such as C extensions, are more friendly for software engineers, although the control over the design is much lower (Coussy et al., 2010). The FPGA-based designs can be exported and distributed as *IP Cores* because theses languages are platform-independent unless very specific features of a given FPGA are employed. This way, *non-recurring engineering* costs (NRE) are cut-down. Therefore FPGAs are in an intermediate stage between software and hardware. Algorithms are programmed by software and *compiled* to a hardware architecture, so an careful hardware/software codesign is fundamental (Moreno et al., 2010). They are able to exploit both spatial and temporal parallelism very efficiently. Since logic cells are independent many arithmetic units can process concurrently, with custom routing between them. In addition, the memory subsystem can be tuned to exploit the on-chip memory banks, reducing the access to the external memories.

Regarding Computer Vision, FPGAs are widely used both in industry and research. They offer a high degree of flexibility and performance to handle many different applications. Most compute-intensive algorithms were migrated to FPGAs: stereo vision (Jin et al., 2010), geometric algebra (Franchini et al., 2009), optical flow (Martineau et al., 2007), object recognition (Meng et al., 2011) or video surveillance (Nair et al., 2005) (Salem et al., 2009) to name some examples. Low and mid-level image processing stages, based on SIMD/MIMD paradigms can be efficiently implemented. However, high-level processing, although it could be possible to implement on an FPGA, fits better on conventional processors. Nevertheless, FPGAs can use external processors connected through high-speed off-chip communications

or include general-purpose processors. There are available soft-core (emulated) (Lysecky & Vahid, 2005) and hard-core (Veale et al., 2006) embedded processors. While the first offer a large degree of configuration, adapting all parameters of the design to the particular needs of the applications, the last feature better performance. This way FPGAs are able to implement all the stages of a complete application (Shi, 2010).

Application-specific integrated circuits

An *Application-Specific Integrated Circuit* (ASIC) is a device designed for a particular task instead of for general purpose functionality. In this case, designers have to work at the very bottom level of design so this process is long and error-prone. As there are elements present in almost all ICs, a set of *libraries* is usually provided making easier system-design. This way it is possible to work at different levels of abstraction. *Full Custom* designs require a greater effort because it is necessary to design both functionality and physical layout. However, it allows better optimizations and performance. *Standard Cells* allow to focus on the logic operation instead on the physical design, splitting the process into two parts. Physical design is usually done by the manufacturer, who provides from simple logic gates to more complex units such as *Flip-Flops* or adders. Apart from simple units, third party manufacturers provide more complex modules for specific functions, known as *IP Cores*. This way, they can be used as subcomponents in a large design. There is a large variety of cores to tackle all needs, as it happens with the FPGAs, and there is available from IO controllers (RAM, PCI-Express, ethernet) to arithmetic cores (signal processing, video and audio decoding) or even complete microprocessors.

The IC-level design allows to build embedded systems, *System-on-Chip*, efficiently. It is possible to include all the elements of the system on a single chip, even if there are digital, analog and mixed-signal modules. Nowadays, custom ASICs are the unique alternative for complex SoCs although FPGAs size grows with each generation and are becoming a viable alternative. IC design leads to high costs, both in design and manufacturing when production volume is low. However, some applications still need ICs because the alternatives do not match with performance, power consumption or size (Kuon & Rose, 2007).

It is true that some of the devices previously mentioned are in some way ASICs. However, the design of a custom architecture for a concrete algorithm provides the best possible results. Many custom chips were designed and built instead of mapping them in a programmable device, for specific algorithms (Kim et al., 2008), domain applications (Stein et al., 2005) or complete general purpose SoCs (Khailany et al., 2008a).

Because of this flexibility a large set of *exotic* devices can be found in the market or in the specialized literature. While some aims to address very specific tasks or novel computing paradigms, others are taking their first steps on the market after that technology has allowed its viability. *Pixel-parallel Processor Arrays* are the natural platform for low-level vision. They are massively parallel SIMD processors laid down on a 2D grid with a processor-per-pixel correspondence and local connections among neighbors. Each processor, very simple, can also include an image sensor to eliminate the IO bottleneck. Some representative examples are (Foldesy et al., 2008; Garrido et al., 2008; Lopich & Dudek, 2008). There are also approaches closer to the biological vision, as (Koyanagi et al., 2001) or (Constandinou et al., 2004). More information is available in (Zarándy, 2011). *Massively Parallel Processor Arrays* (MPPAs)

provide hundreds to thousands of processors. They are encapsulated and independent and have their own program and memories. They work in MIMD mode but also include internal improvements as pipelines, superscalar capabilities or SIMD units. Some examples are (Bell et al., 2008; Butts et al., 2007; Duller et al., 2003).

3.3 Discussion

As described previously in this section, there is a wide range of hardware devices suitable for Computer Vision. Depending on the application requirements, a compromise between performance, cost, power consumption and development time is needed. Commercial applications are heavily constrained by the time-to-market so suboptimal solutions are preferable if the development cycle is shorter. This way, software-based solutions are usually better from the commercial point of view.

As discussed before, the large amount of highly optimized libraries make PCs (conventional microprocessors) the first choice both as development and production platform. Multi-core and SIMD programming are key for performance, although this can significantly increase the development time. One of the benefits of choosing a PC as platform is that a GPU is included *"at no cost"*. This is, most application requires some kind of graphical display so including a GP-capable GPU provides a much greater benefit with a very low cost/performance ratio, even using a low-cost card. The combined use of CPU-GPU has proved to be very effective, although very restricted in terms of form factor and specially in power consumption. Only if these parameters are very constrained DSP-based solutions are preferable. This is the case of mobile applications, although low-power microprocessors are taking advantage in this field. As these are extensively used, development kits, compilers and libraries are very optimized, helping to cut down time-to-market and related costs.

However, if the aforementioned devices do not provide acceptable results, it will be required to go into the hardware. As application developers, we need to search for *exotic* devices such as MPPAs or dedicated image processors. In contrast to the previous devices, these are not normally industry standards therefore a greater effort during development is needed. However, we are still under the software coverage. On the contrary, if the requirements are very strict or if the production volume is very high, a custom chip is the unique alternative to reach the desired performance or to lower the cost per unit. FPGAs are an excellent platform for testing before manufacturing the final design in a custom chip. As they are reconfigurable, different architectures can be evaluated before sending to fabric. On the other hand, some authors claim than software development is the bottleneck in the current ASIC development. The *software stage* can not start until a device where to do tests has been built. Although emulators (both functional an cycle-accurate) are used, their performance is very low, resulting in very large test cycles and poor feedback for those programmers at the hardware control layer. This is why FPGAs are widely used as proof-of-concept devices, as they enable software development cycles many months before a test chip was built.

Tab. 1 shows performance comparison of a computationally intensive algorithm as SURF (Bay et al., 2006) for different platforms. PCs offer uneven performance if heavily use multithreading programming (Zhang, 2010) or a straightforward implementation (Bouris et al., 2010). GPUs feature very large performance at the expense of high power consumption. However, FPGA-based implementation delivers the best performance in terms of speed and

power consumption. Tab. 2 shows a comparison between a low-power CPU and GPU, compared to an standard laptop CPU. Authors conclude that optimization is critical and a carefully analysis of low-level operations must be performed but the achieved performance is quite close to standard conventional processors. Complex algorithms which include a higher abstraction level as Viola-Jones detector (Viola & Jones, 2001) are also candidate for mobile platforms. In (Aby et al., 2011), a Beagleboard xM board with a Texas Instruments DM3730 SoC (ARM Cortex-A8 and TMS320C64X DSP) achieves around $0.5x$ speed-up compared with a conventional Intel 2.2 GHz processor, both using openCV (Bradski & Kaehler, 2008) library. In (Arth & Bischof, 2008), a DSP-based embedded system for object recognition achieves up to 4fps, including SIFT-based feature detection and description and object recognition. Although with lower performance than other approaches, the major advantage is that the whole application fits in a single device reducing also power consumption.

Some operations, as the Fourier transform, are computationally very expensive and yet required in many applications, including low-power, low-cost or high performance devices. Optimized libraries for both CPU (Takahashi, 2007) and GPU (Ogata et al., 2008) aim to exploit SIMD units and multitasking, achieving high performance with regard to straightforward implementations. In particular, Fourier transform operation is very suitable for FPGAs an ASICs if performance and power consumption is critical. In (He & Guo, 2008), a FFT core design for FPGAs is proposed, consuming less than 1W in the worst case and lowering manufacturing cost more than $x15$ compared with a DSP implementation. In (Guan et al., 2009), a more aggressive approach is done, developing an *application-specific instruction set processor* (ASIP). With very little hardware overhead and a consumption of few tenths of a watt, outperforms standard software and DSP implementations more than $x800$ and $x5$ times respectively. However, these designs have larger development cycles, as (Pauwels et al., 2011) depicts. In this work, some low-level operations (phase-based optical flow, stereo and local image features) are compared both on FPGA and GPU. Tab. 3 summarizes some of the results of this work. As authors conclude, high-performance or low-cost implementations should be done on CPUs with GPU co-processing. GPUs overcome FPGAs in terms of absolute performance due to their memory throughput. However, if a standalone platform is needed, an FPGA board should meet the requirements or establish the basis for testing and validating an ASIC design.

Finally, general-purpose custom designs converge form factor, power consumption and performance. For instance, SCAMP processor (Dudek & Hicks, 2005) exploits the massively spatial parallelism of low-level operations, integrating processing units and sensors in a processor-per-pixel fashion. As it is an analog design, the integration density and the performance is very high, keeping power consumption under 240 mW. Current digital solutions also offer similar performance and many advantages as faster development and array scalability. ASPA processor (Lopich & Dudek, 2007) includes novel techniques to increase performance, specially on global operations, without sacrificing the other trade-offs. Hardware-oriented algorithms are also key to take advantage of custom hardware. Tab. 4 shows a comparative between different specific-purpose processors executing an active contour algorithm for retinal-vessel tree extraction. The algorithm is designed for focal-plane processing arrays. The *cycles-per-pixel* parameter illustrates how specific platforms are able to lower hardware requirements and even to increase performance. As well as other custom designs depicted in this chapter, they are not suitable to handle a whole Computer Vision

application as they are intended to reduce the workload of the main processor in the more computational expensive tasks. Approaches as (Khailany et al., 2008b) or (Fijany & Hosseini, 2011) can completely embed highly complex applications without compromise its efficiency.

	Device	Performance	Power (W)
Bouris et al. (2010)	Intel Core 2 Duo 2.4 GHz	$<$ 7 fps	N/S
Zhang (2010)	Intel Core 2 Duo P8600 2.4 GHz	33 fps	25
Terriberry et al. (2008)	nVidia GeForce 880 GTX	56 fps	200
Bouris et al. (2010)	Xilinx Virtex 5XC5VFX130T	70 fps	$<$ 20

Table 1. Summary of different SURF (Bay et al., 2006) implementations on different platforms for images of 640 $\times$ 640 px.

Device	Un-optimized	Optimized
Intel Core 2 Duo Merom 2.4GHz	9.03 fps	16.37 fps
Intel Atom 1.6GHz	2.59 fps	5.48 fps
GMA X3100 GPU 500MHz	1.04 fps	5.75 fps

Table 2. Performance of SIFT (Lowe, 2004) implementations in low-power devices for images of 640 $\times$ 640 px. See (Murphy et al., 2009) for details.

Device	Power (W)	Cost ($)	Time-to-Market (months)
nVidia GeForce GTX 280	236	N.S.	2 (1 persons)
nVidia GeForce GTX 580	244	499	2 (1 persons)
Xilinx Virtex4 xc4vfx100	7.2	2084	15 (2 persons)
Xilinx Virtex5 xc5vlx330t	5.5	12651	12 (2 persons)

Table 3. Main GPU and FPGA costs for optical flow, stereo and local image features implementation. See (Pauwels et al., 2011) for complete details and performance results.

Device	Cores (used/aval.)	Performance (s)	Cycles-per-pixel
Intel Core i7 940 2.93GHz	4/4 (8 threads)	13.7	357993
SCAMP-3 Visual Processor	16384/16384	0.230	8950
Xilinx Spartan-3 sc3s4000-5	90/90	1.349	14070
Xilinx Virtex-6 xc6vlx240t-1	192/192	0.080	3425
Ambric Am2045 MPPA	125/360	0.0087	742

Table 4. On-chip retinal vessel-tree extraction (hardware-oriented algorithm) on different platforms. See (Nieto et al., 2011) for complete performance analysis. *Note: the algorithm was slightly adapted for SCAMP-3 and substantially lightened for Ambric Am2045. The FPGA implementation was carried out faithfully the original algorithm. Virtex-6 results are an estimation.*

Looking ahead

The progress of new technologies, marked by Moore's Law allows increasingly integration density. More hardware resources, with higher clock frequencies, are available for the designer. However, although the ultimate goal is to increase the performance, other parameters come into play. Nowadays, one of the critical trade-offs is power consumption, directly related with energy efficiency and power dissipation, some of the most decisive limitation design constraints (Kim et al., 2003).

Power consumption is driven by two sources, *dynamic* and *static*. Static consumption is a result of the leakage current and it refers when all inputs are held so the circuit is not changing state. On the contrary, the dynamic term refers to the circuit switching at a given frequency. This power is dominated in today CMOS circuits, being directly proportional to frequency. This is one of the capital reasons why the semiconductor industry moves from a *race for frequency* to a *race for parallelism*. In recent years, the industry is making a big effort to increase the parallelism of most devices to keep the performance increase rate. Apart from more arithmetic units, leading architectures integrate more systems previously contained on separated circuits, as microcontrollers or GPUs. To achieve these results, it is still necessary to scale down the transistors. In this sense, the advent of emerging technologies like CMOS-3D (Philip Garrou, 2008) will permit to integrate heterogeneous functions on the same monolithic solution more easily. A vision-oriented ASIC could integrate the image acquisition stage to an eventual processor. At the same time, more conventional solutions as PCs or FPGAs would yield large parallelization using this and other advances such as the Tri-Gate technology (Intel, 2011). However, this involves problems as the increment of leakage currents, thereby increasing static power consumption, not negligible at all nowadays (Koch, 2005) (Kim et al., 2003). In addition, new manufacturing methods are more expensive because the yield is lower and more time is needed to discount the investments. Or equivalently, it is necessary to sell more devices to continue growing at the rate set by Moore's Law.

Conventional microprocessors are in the leading edge of evolution. There is a large market which justifies large investments in R&D to meet the growing needs of consumers, especially by large increase in media consumption. This way, it is now possible to find low-cost multicore microprocessors. It is expected that the current evolution towards a greater number of cores will be maintained but increasingly including more elements previously located on external chips, reducing the bottleneck when communicating with off-chip elements (Singhal, 2008). New parallel computing techniques need to be developed to take advantage of the available multithreading capabilities.

PCs also benefit of GPU capabilities. GPU performance grows at a higher rate than microprocessors. As they are very specialized devices, although featuring general purpose computing, the technical improvements in the semiconductor industry are clearly more beneficial. As discussed previously, there are available hardware resources to increase the parallelism and enhance the datapath pipeline. Leading GPUs have more than 1000 processing units and high speed and bandwidth memories. It is also possible to combine multi-core GPUs to work together, achieving a very large throughput. Still, their major disadvantage is being the power consumption. New architectures are taking advantage of the fixed-function hardware to improve area usage and power efficiency. GPU design will focus entirely on improving GPGPU computing (Brookwood, 2010a).

DSPs are also moving to multicore architectures. As specialized microprocessors, they can take advantage of all the improvements in the consumer market, both in hardware and software improvements such as compilers or other optimization techniques. Although competitors are strong, DSP will continue to be used because they lead to compact circuit boards, lower power consumption and cost, if the appropiate device is selected based on the application requirements. In addition, the benefits of the extensive experience in DSP development, with shorter time-to-market thanks to the very optimized compilers and

libraries. This is specially relevant in embedded applications, to take advantage of the multi-core capabilities of modern DSPs. This way it is possible to integrate several DSP cores, each one optimized for a specific task, on a single chip (Friedmann, 2010). Low-power devices which still keeping reasonable performance are fundamental in handled and portable devices, where traditional microprocessors are not suitable.

Microprocessors and GPUs tend to converge on a single chip. Apart from the obvious benefits of integration, reducing cost, size, power consumption, the performance will increase because the reduction of off-chip communications. In addition, architectures as AMD Fusion integrate in the same units 3D acceleration, parallel processing and other functions of GPUs (Brookwood, 2010b). On the other hand, mobile microprocessors are becoming more important. These microprocessors embed very low-power GPUs and auxiliary DSP units for co-processing in the same chip (NVIDIA, 2011).

Programmable systems, not only FPGAs, are able to get the same performance as recent past ASICs, keeping time-to-market and non-recurring engineering costs lower compared to custom ICs. As discussed previously, FPGAs are between software and hardware solutions. Modern FPGAs experienced a large increase in hardware resources, both in dedicated units and logic cells. High-level programming languages are another major reason why FPGAs are becoming increasingly competitive, specially when dealing with complex FPGAs and to maintain and keep the designs portable (Singh, 2011). Nowadays these devices can address complete SoCs, integrating memory and IO controller natively. Manufacturer roadmaps show their inclusion in a very near future and it is expected a big leap in performance and flexibility (DeHaven, 2010).

4. Summary and conclusions

The large variety of Computer Vision applications makes difficult to classify them into tight categories. As a result, it is extremely difficult to design a unique hardware architecture which handle efficiently all processing stages of any Computer Vision algorithm. In the literature there are available several studies where different platforms are tested under the same conditions (Asano et al. (2009); Baumgartner et al. (2009); Kisacanin (2005); Wnuk (2008)). They show that to tune-up is key for performance and that new parallel computing techniques are a requirement to exploit parallel devices. However, the increase of the market makes investment in new platforms that implement different algorithms a necessity.

The most accessible platform is a Personal Computer equipped with a GP-capable GPU. As test or final platform, it cuts down developing time and costs. GPUs give enough performance for most intensive tasks, while using the CPU multimedia extensions it is possible to meet the requirements in the other stages. In addition, they include all necessary elements for user IO, communication, storage and information display. The availability of models is large enough to select the adequate platform according to the application trade-offs. When CPU performance is not adequate, DSPs are a serious alternative. In addition, it becomes almost mandatory when dealing with embedded devices without compromising performance, where power consumption and form factor are very restrictive. They are widely used for prototyping custom ICs but FPGA-based applications have their own niche. Integration and high flexibility besides a large number of available IP Cores allow to drop NRE

costs. Although all devices described in this chapter are ASICs, they were not conceived for an unique application. To lower costs, the manufacturer expands their range of application although it is possible to find families specialized in specific tasks. But there are available devices very specific for critical tasks, where the requirements are very tight and any other device complies with them. Flexibility is complete and there is not restriction to employ cutting-edge technologies which are not available in commercial devices until a near future.

Almost all Computer Vision applications need to face all processing stages in a lesser or a greater degree. Generally, this leads to implement efficient mechanisms to tackle massively spatial parallelism, mixed spatial and temporal parallelism and sequential processing. Each stage matches with a level of processing so all mechanisms have to be implemented in most applications. Low-level stages benefit of massively parallelism with simple data distribution systems as operations. When the data abstraction level grows, during mid-level tasks, more information about the problem is required by the algorithms increasing their complexity. This leads to complex architectures, where information distribution and sharing makes difficult to exploit spatial parallelism, although it is usually present. Task-parallel architectures are able to exploit better their possibilities. Low and mid-level processing stages can be implemented in pure hardware solutions because they often implement kernel operations. However, high-level is closer to software and designers can take advantage of this to build complex systems easier by using general purpose processors. In addition, the device which performs the image processing related tasks needs to communicate or to control other devices. This is not strictly related with the Computer Vision domain but it is clearly a requirement in the final solution. In this case, the use of a general purpose processor is beneficial because it allows easier control and it increases the flexibility of the whole system.

Although it is almost impossible to develop a system able to run all operations in an optimal way due to the rich nature of the Computer Vision applications, it is desirable to provide the capability to perform any operation. The design must be scalable to adapt it to the specific needs of each application. This way, a product ranging from low to high-end devices can be easily built. The internal architecture should be also modular, so that from a basic outline more features could be added without dramatic changes. In general, a high-end microprocessor is a requirement to manage complex operations and communications between the system and the external components of the complete system. An on-chip SIMD-MIMD hybrid co-processor would tackle the most expensive computation, reconfiguring its internal interconnections according to the current task. Embedded high-speed memory controllers are also key to reduce the data-access bottleneck. All these elements, together, are able to face efficiently most of the situations described throughout this chapter.

5. Acknowledgments

This work is funded by Xunta de Galicia under the projects 10PXIB206168PR and 10PXIB206037PR and the program Maria Barbeito.

6. References

Aby, P., Jose, A., Jose, B., Dinu, L., John, J. & Sabarinath, G. (2011). Implementation and optimization of embedded face detection system, *Signal Processing, Communication,*

Computing and Networking Technologies (ICSCCN), 2011 International Conference on, IEEE, pp. 250–253.

Ahmed, A. & Terada, K. (2010). A general framework for multi-human tracking, *Journal of Software* 5(9): 966–973.

Arth, C. & Bischof, H. (2008). Real-time object recognition using local features on a dsp-based embedded system, *Journal of Real-Time Image Processing* 3(4): 233–253.

Asano, S., Maruyama, T. & Yamaguchi, Y. (2009). Performance comparison of fpga, gpu and cpu in image processing, *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, IEEE, pp. 126–131.

Baumgartner, D., Roessler, P., Kubinger, W., Zinner, C. & Ambrosch, K. (2009). Benchmarks of low-level vision algorithms for dsp, fpga, and mobile pc processors, *Embedded Computer Vision* pp. 101–120.

Bay, H., Tuytelaars, T. & Van Gool, L. (2006). Surf: Speeded up robust features, *Computer Vision–ECCV 2006* pp. 404–417.

BCC Research (2010). Machine Vision: Technologies and Global Markets, *Report IAS010C*, BCC Research.

Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Bao, L., Brown, J. et al. (2008). Tile64-processor: A 64-core soc with mesh interconnect, *Solid-State Circuits Conference, 2008. ISSCC 2008. Digest of Technical Papers. IEEE International*, IEEE, pp. 88–598.

Bouris, D., Nikitakis, A. & Papaefstathiou, I. (2010). Fast and efficient fpga-based feature detection employing the surf algorithm, *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, IEEE, pp. 3–10.

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV*, O'Reilly.

Bramberger, M. & Rinner, B. (2004). An embedded smart camera on a scalable heterogeneous multi-dsp system, *In Proceedings of the European DSP Education and Research Symposium (EDERS 2004*, Citeseer.

Brookwood, N. (2010a). Amd fusion family of apus – enabling a superior, immersive pc experience, *AMD white paper* .

Brookwood, N. (2010b). Amd fusion family of apus – enabling a superior, immersive pc experience, *AMD white paper* .

Butts, M., Jones, A. & Wasson, P. (2007). A structural object programming model, architecture, chip and tools for reconfigurable computing, *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, IEEE, pp. 55–64.

Castano-Díez, D., Moser, D., Schoenegger, A., Pruggnaller, S. & Frangakis, A. S. (2008). Performance evaluation of image processing algorithms on the GPU, *Journal of Structural Biology* 164(1): 153 – 160.

Chapman, B. (2007). The multicore programming challenge, *Lecture Notes in Computer Science* 4847: 3.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. & Skadron, K. (2008). A performance study of general-purpose applications on graphics processors using cuda, *Journal of Parallel and Distributed Computing* 68(10): 1370–1380.

Chen, W., Chen, P., Lee, W. & Huang, C. (2008). Design and implementation of a real time video surveillance system with wireless sensor networks, *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, IEEE, pp. 218–222.

Chu, C. & Chen, S. (2009). Parallel implementation for cone beam based 3d computed tomography (ct) medical image reconstruction on multi-core processors, *World Congress on Medical Physics and Biomedical Engineering, September 7-12, 2009, Munich, Germany*, Springer, pp. 2066–2069.

Constandinou, T., Georgiou, J. & Toumazou, C. (2004). Towards a bio-inspired mixed-signal retinal processor, *Circuits and Systems, 2004. ISCAS'04. Proceedings of the 2004 International Symposium on*, Vol. 5, IEEE, pp. V–493.

Coussy, P., Takach, A., McNamara, M. & Meredith, M. (2010). An introduction to the systemc synthesis subset standard, *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ACM, pp. 183–184.

Craven, S. & Athanas, P. (2007). Examining the viability of fpga supercomputing, *EURASIP Journal on Embedded systems* 2007(1): 13–13.

DeHaven, K. (2010). Extensible processing platform ideal solution for a wide range of embedded systems, *Xilinx White Paper* .

Deselaers, T., Keysers, D. & Ney, H. (2008). Features for image retrieval: an experimental comparison, *Information Retrieval* 11(2): 77–107.

Djordjevic, I., Arabaci, M. & Minkov, L. (2009). Next generation fec for high-capacity communication in optical transport networks, *Journal of Lightwave Technology* 27(16): 3518–3530.

Dolan, R. & DeSouza, G. (2009). Gpu-based simulation of cellular neural networks for image processing, *Proceedings of the 2009 international joint conference on Neural Networks*, IEEE Press, pp. 2712–2717.

Dudek, P. & Hicks, P. (2005). A general-purpose processor-per-pixel analog simd vision chip, *Circuits and Systems I: Regular Papers, IEEE Transactions on* 52(1): 13–20.

Duller, A., Panesar, G. & Towner, D. (2003). Parallel processing-the picochip way, *Communicating Processing Architectures* pp. 125–138.

Fasang, P. (2009). Prototyping for industrial applications [industry forum], *Industrial Electronics Magazine, IEEE* 3(1): 4–7.

Fijany, A. & Hosseini, F. (2011). Image processing applications on a low power highly parallel simd architecture, *Aerospace Conference, 2011 IEEE*, IEEE, pp. 1–12.

Flynn, M. (1972). Some computer organizations and their effectiveness, *Computers, IEEE Transactions on* 100(9): 948–960.

Foldesy, P., Zarándy, Á. & Rekeczky, C. (2008). Configurable 3d-integrated focal-plane cellular sensor–processor array architecture, *International Journal of Circuit Theory and Applications* 36(5-6): 573–588.

Franchetti, F., Kral, S., Lorenz, J. & Ueberhuber, C. (2005). Efficient utilization of simd extensions, *Proceedings of the IEEE* 93(2): 409–425.

Franchini, S., Gentile, A., Sorbello, F., Vassallo, G. & Vitabile, S. (2009). An embedded, fpga-based computer graphics coprocessor with native geometric algebra support, *Integration, the VLSI Journal* 42(3): 346–355.

Friedmann, A. (2010). Enabling small cells with tiâ€™s new multicore soc, *Texas Instruments White Paper* .

Garrido, S., Listán, J., Alba, L., Utrera, C., Rodríguez-Vázquez, S., Domínguez-Castro, R., Jiménez-Espejo, F. & Romay, R. (2008). The Eye-RIS CMOS Vision System, *Analog circuit design: sensors, actuators and power drivers; integrated power amplifiers from wireline to RF; very high frequency front ends* pp. 15–32.

Govindaraju, N. & Manocha, D. (2007). Cache-efficient numerical algorithms using graphics hardware, *Parallel Computing* 33(10-11): 663–684.

Guan, X., Lin, H. & Fei, Y. (2009). Design of an application-specific instruction set processor for high-throughput and scalable fft, *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 1302–1307.

Haralick, R. M. & Shapiro, L. G. (1985). Image segmentation techniques, *Computer Vision, Graphics, and Image Processing* 29(1): 100 – 132.

Harris, C. & Stephens, M. (1988). A combined corner and edge detector, *Alvey vision conference*, Vol. 15, Manchester, UK, p. 50.

He, H. & Guo, H. (2008). The realization of fft algorithm based on fpga co-processor, *Second International Symposium on Intelligent Information Technology Application*, IEEE, pp. 239–243.

Heymann, S., Maller, K., Smolic, A., Froehlich, B. & Wiegand, T. (2007). Sift implementation and optimization for general-purpose gpu, *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, Citeseer.

Intel (2011). Intel 22nm 3-d tri-gate transistor technology, *Intel Documentation* .

Jin, S., Cho, J., Dai Pham, X., Lee, K., Park, S., Kim, M. & Jeon, J. (2010). Fpga design and implementation of a real-time stereo vision system, *Circuits and Systems for Video Technology, IEEE Transactions on* 20(1): 15–26.

Khailany, B., Williams, T., Lin, J., Long, E., Rygh, M., Tovey, D. & Dally, W. (2008a). A programmable 512 gops stream processor for signal, image, and video processing, *Solid-State Circuits, IEEE Journal of* 43(1): 202 –213.

Khailany, B., Williams, T., Lin, J., Long, E., Rygh, M., Tovey, D. & Dally, W. (2008b). A programmable 512 gops stream processor for signal, image, and video processing, *Solid-State Circuits, IEEE Journal of* 43(1): 202–213.

Kim, K., Kim, J.-Y., Lee, S., Kim, M. & Yoo, H.-J. (2008). A 211 gops/w dual-mode real-time object recognition processor with network-on-chip, *Solid-State Circuits Conference, 2008. ESSCIRC 2008. 34th European*, pp. 462 –465.

Kim, N., Austin, T., Baauw, D., Mudge, T., Flautner, K., Hu, J., Irwin, M., Kandemir, M. & Narayanan, V. (2003). Leakage current: Moore's law meets static power, *Computer* 36(12): 68–75.

Kisacanin, B. (2005). Examples of low-level computer vision on media processors, *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, IEEE, pp. 135–135.

Koch, G. (2005). Discovering multi-core: extending the benefits of mooreâ€™s law, *Technology* 1.

Kolsch, M. & Butner, S. (2009). Hardware Considerations for Embedded Vision Systems, *Embedded Computer Vision, Springer* pp. 3–26.

Koyanagi, M., Nakagawa, Y., Lee, K., Nakamura, T., Yamada, Y., Inamura, K., Park, K. & Kurino, H. (2001). Neuromorphic vision chip fabricated using three-dimensional integration technology, *Solid-State Circuits Conference, 2001. Digest of Technical Papers. ISSCC. 2001 IEEE International*, IEEE, pp. 270–271.

Kuon, I. & Rose, J. (2007). Measuring the gap between fpgas and asics, *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 26(2): 203 –215.

Lee, V., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P. et al. (2010). Debunking the 100x gpu vs. cpu myth:

an evaluation of throughput computing on cpu and gpu, *ACM SIGARCH Computer Architecture News*, Vol. 38, ACM, pp. 451–460.

Leong, P. (2008). Recent trends in fpga architectures and applications, *Electronic Design, Test and Applications, 2008. DELTA 2008. 4th IEEE International Symposium on*, IEEE, pp. 137–141.

Li, L., Socher, R. & Fei-Fei, L. (2009). Towards total scene understanding: Classification, annotation and segmentation in an automatic framework, *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, IEEE, pp. 2036–2043.

Lin, C. & Chiu, Y. (2008). The dsp based catcher robot system with stereo vision, *Advanced Intelligent Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on*, IEEE, pp. 897–903.

Lin, T., Liu, C., Tseng, S., Chu, Y. & Wu, A. (2008). Overview of itri pac project-from vliw dsp processor to multicore computing platform, *VLSI Design, Automation and Test, 2008. VLSI-DAT 2008. IEEE International Symposium on*, IEEE, pp. 188–191.

Lindholm, E., Nickolls, J., Oberman, S. & Montrym, J. (2008). Nvidia tesla: A unified graphics and computing architecture, *Micro, IEEE* 28(2): 39–55.

Lopich, A. & Dudek, P. (2007). Global operations in simd cellular processor arrays employing functional asynchronism, *Computer Architecture for Machine Perception and Sensing, 2006. CAMP 2006. International Workshop on*, IEEE, pp. 18–23.

Lopich, A. & Dudek, P. (2008). Aspa: Focal plane digital processor array with asynchronous processing capabilities, *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, IEEE, pp. 1592–1595.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints, *International journal of computer vision* 60(2): 91–110.

Lui, W. & Jarvis, R. (2010). Eye-full tower: A gpu-based variable multibaseline omnidirectional stereovision system with automatic baseline selection for outdoor mobile robot navigation, *Robotics and Autonomous Systems* 58(6): 747–761.

Lysecky, R. & Vahid, F. (2005). A study of the speedups and competitiveness of fpga soft processor cores using dynamic hardware/software partitioning, *Proceedings of the conference on Design, Automation and Test in Europe-Volume 1*, IEEE Computer Society, pp. 18–23.

Martineau, M., Wei, Z., Lee, D. & Martineau, M. (2007). A fast and accurate tensor-based optical flow algorithm implemented in fpga, *Applications of Computer Vision, 2007. WACV'07. IEEE Workshop on*, IEEE, pp. 18–18.

Meier, L., Tanskanen, P., Fraundorfer, F. & Pollefeys, M. (2011). Pixhawk: A system for autonomous flight using onboard computer vision, *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, pp. 2992–2997.

Meng, H., Appiah, K., Hunter, A. & Dickinson, P. (2011). Fpga implementation of naive bayes classifier for visual object recognition, IEEE Computer Vision and Pattern Recognition.

Mikolajczyk, K. & Schmid, C. (2005). A performance evaluation of local descriptors, *IEEE transactions on pattern analysis and machine intelligence* pp. 1615–1630.

Moreno, F., Lopez, I. & Sanz, R. (2010). A design process for hardware–software system co-design and its application to designing a reconfigurable fpga, *Digital System Design - Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, IEEE, pp. 556–562.

Murphy, M., Keutzer, K. & Wang, H. (2009). Image feature extraction for mobile processors, *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, IEEE, pp. 138–147.

Naffziger, S. (2009). Microprocessors of the future: Commodity or engine growth?, *Solid-State Circuits Magazine, IEEE* 1(1): 76 –82.

Nair, V., Laprise, P. & Clark, J. (2005). An fpga-based people detection system, *EURASIP journal on applied signal processing* 2005: 1047–1061.

Neri, C., Baccarelli, G., Bertazzoni, S., Pollastrone, F. & Salmeri, M. (2005). Parallel hardware implementation of radar electronics equipment for a laser inspection system, *Nuclear Science, IEEE Transactions on* 52(6): 2741 –2748.

Nieto, A., Brea, V., Vilarino, D. & Osorio, R. (2011). Performance analysis of massively parallel embedded hardware architectures for retinal image processing, *EURASIP Journal on Image and Video Processing* 2011: 10.

NVIDIA (2011). Variable smp – a multi-core cpu architecture for low power and high performance, *NVIDIA Corporation white paper* .

Ogata, Y., Endo, T., Maruyama, N. & Matsuoka, S. (2008). An efficient, model-based cpu-gpu heterogeneous fft library, *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, IEEE, pp. 1–10.

Oliver, N., Rosario, B. & Pentland, A. (2000). A bayesian computer vision system for modeling human interactions, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22(8): 831 –843.

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., KrALger, J., Lefohn, A. E. & Purcell, T. J. (2007). A survey of general-purpose computation on graphics hardware, *Computer Graphics Forum* 26(1): 80–113.

Pal, N. R. & Pal, S. K. (1993). A review on image segmentation techniques., *Pattern Recognition* pp. 1277–1294.

Pauwels, K., Tomasi, M., Alonso, J., Ros, E. & Van Hulle, M. (2011). A comparison of fpga and gpu for real-time phase-based optical flow, stereo, and local image features, *IEEE Transactions on Computers* .

Philip Garrou, Chistopher Bower, P. R. (2008). *Handbook of 3D Integration*, Wiley-VCH.

Podlozhnyuk, V. (2007). Image convolution with cuda, *NVIDIA Corporation white paper, June* 2097(3).

Ren, F., Huang, J., Terauchi, M., Jiang, R. & Klette, R. (2010). Lane detection on the iphone, *Arts and Technology* pp. 198–205.

Rinnerthaler, F., Kubinger, W., Langer, J., Humenberger, M. & Borbély, S. (2007). Boosting the performance of embedded vision systems using a dsp/fpga co-processor system, *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, IEEE, pp. 1141–1146.

Rixner, S., Dally, W., Kapasi, U., Khailany, B., López-Lagunas, A., Mattson, P. & Owens, J. (1998). A bandwidth-efficient architecture for media processing, *Proceedings of the 31st annual ACM/IEEE international symposium on Microarchitecture*, IEEE Computer Society Press, pp. 3–13.

Salem, M., Klaus, K., Winkler, F. & Meffert, B. (2009). Resolution mosaic-based smart camera for video surveillance, *Distributed Smart Cameras, 2009. ICDSC 2009. Third ACM/IEEE International Conference on*, IEEE, pp. 1–7.

Schneiderman, R. (2010). Dsps evolving in consumer electronics applications [special reports], *Signal Processing Magazine, IEEE* 27(3): 6–10.

Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R. et al. (2008). Larrabee: a many-core x86 architecture for visual computing, *ACM Transactions on Graphics (TOG)* 27(3): 1–15.

Shah, S., Khattak, T., Farooq, M., Khawaja, Y., Bais, A., Anees, A. & Khan, M. (2008). Real time object tracking in a video sequence using a fixed point dsp, *Advances in Visual Computing* pp. 879–888.

Shi, Y. (2010). Smart cameras for machine vision, *Smart Cameras* pp. 283–303.

Shirvaikar, M. & Bushnaq, T. (2009). A comparison between DSP and FPGA platforms for real-time imaging applications, Vol. 7244, SPIE, p. 724406.

Singh, D. (2011). Higher level programming abstractions for fpgas using opencl, *ALTERA* .

Singhal, R. (2008). Inside intel® next generation nehalem microarchitecture, *Hot Chips*, Vol. 20.

Smith, S. M. & Brady, J. M. (1997). Susanâ€š a new approach to low level image processing, *International Journal of Computer Vision* 23: 45–78.

Stein, G., Rushinek, E., Hayun, G. & Shashua, A. (2005). A computer vision system on a chip: a case study from the automotive domain, *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, p. 130.

Sun, T. & Yu, Y. (2009). Memory usage reduction method for fft implementations on dsp based embedded system, *Consumer Electronics, 2009. ISCE'09. IEEE 13th International Symposium on*, IEEE, pp. 812–815.

Suzuki, K., Ikeda, H., Ishimaru, K., Suzuki, J., Adachi, F. & Wang, X. (2007). New image retrieval system utilizing image directory on gigabit network for distributing industrial product information, *Industry Applications, IEEE Transactions on* 43(4): 1099 –1107.

Szeliski, R. (2010). *Computer vision: Algorithms and applications*, Springer-Verlag New York Inc.

Takahashi, D. (2007). Implementation and evaluation of parallel fft using simd instructions on multi-core processors, *Innovative architecture for future generation high-performance processors and systems, 2007. iwia 2007. international workshop on*, IEEE, pp. 53–59.

Talavera, G., Jayapala, M., Carrabina, J. & Catthoor, F. (2008). Address generation optimization for embedded high-performance processors: A survey, *Journal of Signal Processing Systems* 53(3): 271–284.

Taylor, S., Rosten, E. & Drummond, T. (2009). Robust feature matching in 2.3μs, *IEEE CVPR Workshop on Feature Detectors and Descriptors: The State Of The Art and Beyond*.

Terriberry, T., French, L. & Helmsen, J. (2008). Gpu accelerating speeded-up robust features, *Proc. Int. Symp. on 3D Data Processing, Visualization and Transmission (3DPVT)*, Citeseer, pp. 355–362.

Texas Instruments (2002). Tms320c6000 programmer's guide, *White Paper* .

Thall, A. (2006). Extended-precision floating-point numbers for gpu computation, *ACM SIGGRAPH 2006 Research posters*, ACM, pp. 52–es.

Trajkovi, M. & Hedley, M. (1998). Fast corner detection, *Image and Vision Computing* 16(2): 75–87.

Triggs, B., McLauchlan, P., Hartley, R. & Fitzgibbon, A. (2000). Bundle adjustment: A modern synthesis, *in* B. Triggs, A. Zisserman & R. Szeliski (eds), *Vision Algorithms: Theory and Practice*, Vol. 1883 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 153–177.

Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M., Dolan, J., Duggins, D., Galatali, T., Geyer, C. et al. (2008). Autonomous driving in urban environments: Boss and the urban challenge, *Journal of Field Robotics* 25(8): 425–466.

Veale, B., Antonio, J., Tull, M. & Jones, S. (2006). Selection of instruction set extensions for an fpga embedded processor core, *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, p. 8 pp.

Viola, P. & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features, *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 1: 511.

Wang, M., Wang, Y., Liu, D., Qin, Z. & Shao, Z. (2010). Compiler-assisted leakage-aware loop scheduling for embedded vliw dsp processors, *Journal of Systems and Software* 83(5): 772–785.

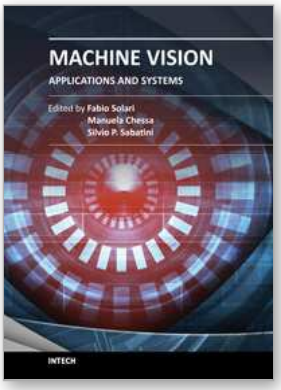
Wither, J., Tsai, Y. & Azuma, R. (2011). Mobile augmented reality: Indirect augmented reality, *Computers and Graphics* 35(4): 810–822.

Wnuk, M. (2008). Remarks on hardware implementation of image processing algorithms, *International Journal of Applied Mathematics and Computer Science* 18(1): 105–110.

Zarándy, Ã. (2011). *Focal-plane sensor-processor chips*, Springer Verlag.

Zhang, N. (2010). Computing optimised parallel speeded-up robust features (p-surf) on multi-core processors, *International Journal of Parallel Programming* 38(2): 138–158.

Zitova, B. & Flusser, J. (2003). Image registration methods: a survey, *Image and Vision Computing* 21(11): 977 – 1000.



Machine Vision - Applications and Systems
Edited by Dr. Fabio Solari

ISBN 978-953-51-0373-8
Hard cover, 272 pages
Publisher InTech
Published online 23, March, 2012
Published in print edition March, 2012

Vision plays a fundamental role for living beings by allowing them to interact with the environment in an effective and efficient way. The ultimate goal of Machine Vision is to endow artificial systems with adequate capabilities to cope with not a priori predetermined situations. To this end, we have to take into account the computing constraints of the hosting architectures and the specifications of the tasks to be accomplished, to continuously adapt and optimize the visual processing techniques. Nevertheless, by exploiting the low?cost computational power of off?the?shell computing devices, Machine Vision is not limited any more to industrial environments, where situations and tasks are simplified and very specific, but it is now pervasive to support system solutions of everyday life problems.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821