electronics

MDPI

Review

Hardware Architectures for Real-Time Medical Imaging

Eduardo Alcaín [1], Pedro R. Fernández [2], Rubén Nieto [2], Antonio S. Montemayor [1], Jaime Vilas [2], Adrian Galiana-Bordera [2,3], Pedro Miguel Martinez-Girones [2,3], Carmen Prieto-de-la-Lastra [2,3], Borja Rodriguez-Vila [2,3], Marina Bonet [2], Cristina Rodriguez-Sanchez [2], Imene Yahyaoui [2], Norberto Malpica [2,3], Susana Borromeo [2], Felipe Machado [2,†] and Angel Torrado-Carvajal [2,3,*,†]

1 Department of Computer Science and Statistics, Universidad Rey Juan Carlos, 28933 Madrid, Spain; eduardo.alcain.ballesteros@gmail.com (E.A.); antonio.sanz@urjc.es (A.S.M.)
2 Department of Electronics, Universidad Rey Juan Carlos, 28933 Madrid, Spain; pedro.barbosa@urjc.es (P.R.F.); ruben.nieto@urjc.es (R.N.); j.vilas@alumnos.urjc.es (J.V.); adrian.galiana@urjc.es (A.G.-B.); pedromiguel.martinez@urjc.es (P.M.M.-G.); carmen.prieto.lastra@urjc.es (C.P.-d.-l.-L.); borja.rodriguez.vila@gmail.com (B.R.-V.); marina.bsanz@urjc.es (M.B.); cristina.rodriguez.sanchez@urjc.es (C.R.-S.); imene.yahyaoui@urjc.es (I.Y.); norberto.malpica@urjc.es (N.M.); susana.borromeo@urjc.es (S.B.); felipe.machado@urjc.es (F.M.)
3 Medical Image Analysis and Biometry Laboratory, Universidad Rey Juan Carlos, 28933 Madrid, Spain
* Correspondence: angel.torrado@urjc.es; Tel.: +34-91-488-4902
† Contributed equally to this work.

Abstract: Medical imaging is considered one of the most important advances in the history of medicine and has become an essential part of the diagnosis and treatment of patients. Earlier prediction and treatment have been driving the acquisition of higher image resolutions as well as the fusion of different modalities, raising the need for sophisticated hardware and software systems for medical image registration, storage, analysis, and processing. In this scenario and given the new clinical pipelines and the huge clinical burden of hospitals, these systems are often required to provide both highly accurate and real-time processing of large amounts of imaging data. Additionally, lowering the prices of each part of imaging equipment, as well as its development and implementation, and increasing their lifespan is crucial to minimize the cost and lead to more accessible healthcare. This paper focuses on the evolution and the application of different hardware architectures (namely, CPU, GPU, DSP, FPGA, and ASIC) in medical imaging through various specific examples and discussing different options depending on the specific application. The main purpose is to provide a general introduction to hardware acceleration techniques for medical imaging researchers and developers who need to accelerate their implementations.

Keywords: biomedical imaging systems; hardware acceleration; medical imaging; medical image analysis; parallel architectures

Citation: Alcaín, E.; Fernández, P.R.; Nieto, R.; Montemayor, A.S.; Vilas, J.; Galiana-Bordera, A.; Martinez-Girones, P.M.; Prieto-de-la-Lastra, C.; Rodriguez-Vila, B.; Bonet, M.; et al. Hardware Architectures for Real-Time Medical Imaging. *Electronics* **2021**, *10*, 3118. https://doi.org/10.3390/electronics10243118

Academic Editor: Paris Kitsos

Received: 15 October 2021
Accepted: 6 December 2021
Published: 15 December 2021

1. Introduction

Medical imaging is a set of techniques and methods that noninvasively acquire a variety of images of the body's internal aspects by means of several effects and interactions with different tissues, revolutionizing modern diagnostic imaging, radiology, and nuclear medicine [1,2]. Currently, there are a wide range of imaging modalities, i.e., ultrasound (US), conventional radiology, arthroscopy, computed tomography (CT), magnetic resonance imaging (MR), bone scintigraphy, positron emission tomography (PET), and combined technologies such as PET/MR, that provide complementary information, being the radiologists who determine the most appropriate examination in a specific clinical situation [3]. For example, CT images provide information on the densities of different tissues; MR images can provide anatomical, functional, perfusion or diffusion information depending on the different sequences used; and PET images provide metabolic information, perfusion

or receptor occupation and binding by detecting the concentration of radiotracers within the body.

Medical imaging applications are very demanding systems. Earlier prediction and treatment have been driving the acquisition of higher image resolutions as well as the fusion of different modalities, raising the need for sophisticated software/hardware systems for medical image registration, storage, analysis, and processing [4,5]. They demand complex computations and real-time processing of the images, whose number, size, resolution, and bit depth tend to increase with the evolution of the technology scenario, and given the new clinical pipelines and the huge clinical burden of hospitals, these systems are often required to provide both highly accurate and real-time processing of large amounts of imaging data. This is crucial because it is directly related to the uncomfortable patient experience and generally higher clinical costs, hence the importance of achieving faster and better imaging studies. Additionally, lowering the prices of each part of imaging equipment, as well as its development and implementation, and increasing their lifespan is crucial to grant more affordable and accessible healthcare in general [6,7]. Given these circumstances, the development and implementation of novel hardware architectures have been an essential requirement when dealing with imaging technologies and real-time image processing in healthcare applications.

In fact, the most common and flexible platform to code or develop on is probably the personal computer or PC. As such, many applications are initially developed and refined for Central Processing Units (CPUs). While their computing capabilities might not match up to more specialized platforms, CPUs must not be underestimated. Their computational performance has increased drastically since the dawning of this technology, either by increasing their clock frequency or by increasing the number of cores. Currently, most CPUs achieve a compromise between the number of cores and clock frequency, allowing different combinations of performance and power-draw to be achieved. This trend has also enabled multi-processing using threads.

In this respect, to reduce the computation time, it is usually thought of first using hardware with high computational power. Although the processing power of CPUs in PCs continues to increase, this option may not be the best choice depending on the specific applications. This is particularly relevant when it comes to designing a system that requires low power consumption and high performance.

In the literature, the most discussed hardware accelerators for computer vision and image processing algorithms can be grouped into Graphics Processing Units (GPUs), Digital Signal Processors (DSP), and Field Programmable Gate Arrays (FPGAs) [8]. In this review, Application Specific Integrated Circuits (ASICs) are also included, as they have been widely used for the implementation of certain medical imaging equipment parts.

Despite being a more specialized technology, the GPUs platforms fit perfectly in the medical imaging niche, offering a way to speed up certain computational tasks and algorithms (compared to CPUs) and still maintain a certain amount of flexibility. As such, GPUs are often used when CPUs fail to meet the needs of a specific application, but another specialized platform cannot be used either.

While usually not the fastest platform, DSPs specialize in digital signal processing, and as such, are the best performers in that field. They barely offer any flexibility, so it is common to use them combined with other hardware solutions in complex designs where fast digital signal processing is needed, yet it is not the only kind of processing performed.

FPGAs are integrated circuits capable of post-fabrication reconfiguration, both interconnect and hardware functionality, using hardware description languages (HDLs). This type of device is based on an array of logic blocks, such as lookup tables (LUTs), flip flops and logic gates, connected through programmable interconnects along with input/output ports. Therefore, FPGAs allow custom hardware design with the flexibility to make modifications due to design errors or improvements. Thus, it is recommended to use FPGAs when the system specifications demand high performance, usually with real-time

processing. Indeed, they are used to avoiding becoming involved in the complexities and lower flexibility of ASIC design (as it will be described now).

ASICs are devices made specifically to fulfill the required functionality; thus, as its name suggests, they are integrated circuits tailored for specific applications. Consequently, ASICs provide smaller form factors, higher performance, and less power consumption since they are manufactured to custom design specifications. Furthermore, their implementation offers a significantly lower cost per unit for very high-volume designs. However, designing an ASIC is a long, complex, and high-risk endeavor that requires dedicating many resources.

Nowadays, modern hardware architectures have evolved to provide competitive clock rates for use in high-capacity imaging equipment. In addition, all architectures have benefited from increased logic density, as well as other features such as integration of dedicated memory and high-speed serial input/output. As such, all modern hardware architectures may offer competitive solutions in the image processing space depending on system requirements and flexibility.

Several reviews on hardware accelerators can be found in the state-of-the-art. However, none of these reviews offers a general overview of all the hardware technologies in the field of medical imaging. While the most comprehensive reviews analyze the different hardware technologies [9–11], they do it in the field of artificial intelligence and/or deep learning algorithms in general.

On the other hand, those reviews focused on medical image processing generally do not cover all the hardware possibilities but only one of them. Different surveys can be found focused on the use of GPU for medical imaging in general [12,13], but also more specifically on medical image reconstruction [14], segmentation [15] or registration [16,17]. Similar results appear related to the use of FPGA, both in general medical image processing [18] and in specific applications [19]. A specific review on photon counting ASICs for spectroscopic X-ray imaging, with emphasis on the CT medical imaging application, is presented in [20]. To the best of our knowledge, there are no specific reviews on DSP technologies focused on medical image processing.

The present paper offers an introduction on the different hardware accelerators and their applications on medical imaging that will be of high interest for those researchers and developers in the field of medical imaging that are concerned about using different hardware architectures to speed up their implementations. In the next section, a brief presentation explaining the evolution of each one of the technologies is offered, followed by several examples of their application in medical imaging. In the discussion, a comparison of the technologies is detailed in depth, highlighting strengths and drawbacks, and general guidelines for selecting the proper technology for every need are presented.

2. Evolution of Hardware Architectures and Their Applications in Real-Time Medical Imaging

In this section, the evolution of the different architectures (CPUs, GPUs, DSPs, FPGAs and ASICs) and their use and performance in medical imaging through several specific examples are explained.

2.1. Central Processing Units (CPUs)

2.1.1. Evolution of CPU Architectures

Consumer CPU architectures have traditionally powered their performance by increasing the frequency clock as well as including parallel capabilities since the 1990s. In the Intel x86 family, the Single Instruction Multiple Data (SIMD) introduction was coined MultiMedia eXtensions (MMX) in 1996 for the release of the Intel Pentium MMX processor [21]. They were composed of eight 64-bit wide registers for integer data with an instruction set of 57 operations. They were focused on the early multimedia needs, mainly in the video game industry and imagery from the Internet irruption. They could execute one instruction

in 64-bit integer data (such as 8 independent bytes or 2 32-bits words) simultaneously, saving loop iterations in integer data processing (such as images) [22].

With the 3D video game industry in mind, in 1998, AMD released an extension for supporting floating-point operations on the MMX registers called 3DNow! technology [23]. In 1999, Intel released the Pentium III processor with new 128-bit floating-point registers for 4 single precision 32-bit floating-point data called SSE (Streaming SIMD Extensions) and an instruction set of 70 operations. After several updates of the SSE instruction set for packing different integer and floating-point data configurations in the 128-bit registers, Intel launched the Advanced Vector eXtensions (AVX) in 2011 [24]. AVX registers double the 128-bit wide SSE registers to 256-bits, adding extra updates to the instruction set, which were again renewed in 2014 with the AVX2 SIMD technology. Initially for the Intel Xeon Phi HPC platform, an update of 256 to 512-bit wide registers was released with the AVX-512 extensions [25].

All these SIMD facilities were added for parallel processing and were especially suited for image processing tasks. An optimized compiler configuration would try to automate the use of these instructions and registers, though sometimes a good data layout is needed to ease the automatic vectorization.

For many years, Moore's law and Dennard scaling have delivered greater performance because a greater number of transistors per chip can be packed, and they could perform faster with less power consumption almost every two years [26]. However, the physical limits have been reached, and obtaining a constant free speedup increase with new computer generation has slowed down considerably [27,28]. In the past two decades, it has been seen how CPUs have evolved from single-core architectures (only one processor in die to process instructions) to multicore architectures combining several independent cores in a single die. Nowadays, multicore architectures have become very popular, and desktop CPUs as well as high-end computing machines improve their computational performance by means of parallel resources.

On the other hand, as stated by Amdahl's Law, the performance of parallel computing is limited by its serial components [29,30]. Although multicore CPUs offer outstanding instruction execution speed with reduced power consumption, optimizing performance of individual processors and then incorporating them by interconnection between processors and access to shared resources on a single die is a non-trivial task [31,32].

In this scenario, concurrent primitives such as mutex, locks, and monitors, etc., could be employed for synchronization of computing threads and avoidance of race conditions in shared memory problems. However, their use becomes extremely difficult when scaling from two to more computing threads. This situation led to the development of solutions from the low-level POSIX Threads (Pthreads) [33] to higher level open proposals such as the Open Multi-Processing specification (OpenMP) [34], or proprietary solutions such as Intel Threading Building Blocks (TBB) [35]. Moreover, simultaneous multithreading technologies such as 2-way Intel Hyper-threading or 4-way IBM BlueGene can also spread the parallel capabilities of a desktop processor. Based on this, if an average quad-core (four cores) processor is being used, it is possible to run 4, 6 or up to 8 threads that would perform optimally (with gains up to 30% [36]). Likewise, if a hexa-core (six cores), octa-core (eight cores) or deca-core (ten cores) processor is being used, 12, 16 or 20 threads could be run up with a good performance, respectively. Currently, it is even possible to find processors with up to 128 cores (ARM Ampere Altra, single threaded cores) or with 64 cores and 128 threads (AMD Ryzen). Thus, OpenMP is an example of how to parallelize the computation successfully, efficiently, and easily, thus, speeding up CPU-based implementations.

Figure 1 shows the most significant milestones over time on CPUs that have contributed significantly to CPU-based hardware acceleration for image processing.

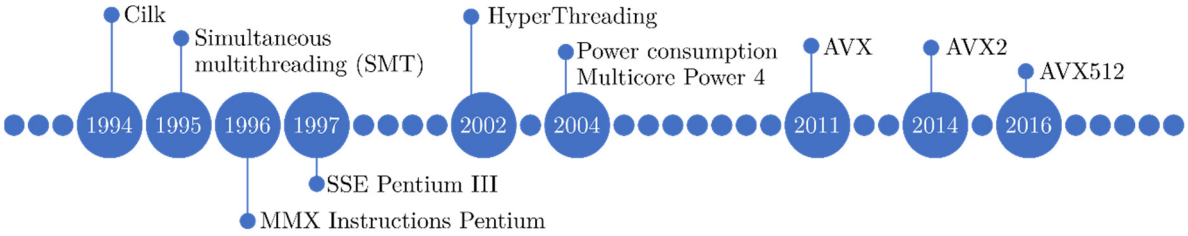

Figure 1. Most significant milestones over time on CPU-based hardware acceleration for image processing.

2.1.2. CPU Architectures in Medical Imaging

Medical imaging has been historically tied to CPU computing, as it has been the more accessible and flexible architecture to prototype and deploy algorithms. In this sense, several studies have considered and studied the effect of multicore architectures and parallel processing in several medical imaging applications.

Membarth et al. performed a comprehensive assessment of the use of different frameworks for multicore architectures to single-core implementations in an X-ray image registration problem [37]. They compared OpenMP, Cilk++, TBB, RapidMind and OpenCL as parallelization languages for multicore processors and provided with usability, performance and overhead estimations, showing a speedup of ~6 times for an octa-core processor and up to ~18 times when using 24 cores compared to a single-core processor sequential implementation. In addition, the study performed by S. Ekström demonstrated how the combination of a GPU with a CPU can accelerate the image registration process by parallelizing the tasks instead of only using a GPU or a CPU [38]. Thus, they performed the matching algorithm in the GPU while the registration optimization was computed in the CPU. The performance of their method was evaluated on brain images. They sped up the process by a factor of four and eight compared to the GPU-only and the advanced normalization tools (ANTs) implementations, respectively.

Kegel et al. showed how the use of multicore architectures improved the computational performance of the list-mode Ordered Subset Expectation Maximization (OSEM) algorithm for 3D PET image reconstruction [39]. In this study, the authors compared the use of Pthreads, OpenMP and TBB, demonstrating a speedup of parallel implementations for a single subset iteration of up to approximately five times on a dual quad-core processor.

Kalamkar et al. showed the speedup obtained by using multicore systems to improve the performance of the Non-Uniform Fast Fourier Transform (NUFFT) in iterative 3D non-Cartesian MRI reconstruction [40]. The high performance of their algorithm implementation relied on an efficient SIMD utilization rate and high parallel efficiency, demonstrating a speedup of more than four times on a 12-core processor compared to the best implementation in that time.

Saxena et al. compared the use of multicores in a method for kidney segmentation from abdominal images [41]. The authors showed how some segmentation tasks could be more efficient when each core is responsible for an individual region, demonstrating a speedup of ~2–4 times on a quad-core processor. Moreover, several groups have proven the efficiency and acceleration of fuzzy c-means and fuzzy c-means based segmentations when they are implemented on hybrid CPU-GPU designs. This way, promising results have been obtained in different parts of the body, such as brain and breast [42,43].

CPUs are accessible hardware architectures. Hence, the use of CPUs is much more widespread in the clinic than that of GPUs due to their lower complexity and cost. For that reason, Vaze et al. designed a CNN and implemented it on a CPU system for real-time ultrasound segmentation [44]. They reduced the computation time by 9 and the memory requirements by 420 compared to the traditional U-net method. Thus, images were processed at 30 fps, enabling real-time applications suitable for ultrasound imaging in the clinical environment.

All these publications confirm that the use of multicore architectures increases the CPU performance up to a point where serial component restrictions limit the performance

growth, even lowering it. The combination of GPU with CPU designs also reduces the computation time in medical image processing. Moreover, the accessibility of CPUs compared to other architectures makes them convenient for implementations in the clinic, even for deep learning applications.

2.2. Graphics Processing Units (GPUs)

2.2.1. Evolution of GPU Architectures

At the same time CPUs were evolving into more efficient architectures. GPUs evolved dramatically to allow tackling problems other than computer graphics and video games. The GPU architecture has fundamental differences in comparison with CPU architecture. First, GPUs possess more Arithmetic Logic Units (ALU), making it possible to compute more operations, but on the contrary, they lose flow control [45]. Second, GPUs are equipped with small caches, focused on producing a better bandwidth output instead of reducing latency [46].

To use this hardware, algorithms must be adapted to this new paradigm. Two parts in this CPU–GPU computing ecosystem can be distinguished: host and device. The host is the CPU that controls the device computation, whereas the device is the graphics hardware that processes the data. The instructions to accomplish a specific procedure executed are encoded in kernels.

The first generation of GPUs was designed as simpler peripherals to display information on monitors. After this, the second generation incorporated dedicated memory and processors to render effects. They were configurable but not programmable. At the same time, some 3D OpenGL [47], and DirectX [48] libraries were released. The third generation came with two specific graphic processors allowing the implementation of two stages: vertex shaders and fragment shaders through a rendering pipeline [49]. In the next generation, a unified device architecture was created after merging these two kinds of processors into a single scalar processor.

Programming the GPU for general purpose applications using this graphics context was called GPGPU (general-purpose computation on GPUs). Although a wide range of functions had been added, the programming was still complicated. There was an effort in the software direction to create a model to better manipulate the resources and let the programmers map more problems without describing them in terms of graphics (GPU Computing).

Since then, there has been a constant evolution both in hardware and in software. There are currently several GPU manufacturers: AMD/ATI, Intel, and NVIDIA, to name the most relevant ones. AMD stream computing technology came to the market a long time after NVIDIA introduced CUDA (originally Compute Unified Device Architecture) [50]. As a result, NVIDIA has far more applications available for CUDA than AMD/ATI (HIP programming language) does for its competing stream technology [51].

Each generation includes more processors for computation, increasing parallelism and scalability, for example, from NVIDIA Tesla C870 (2007) with a processor size of 90 nm and 681 million transistors to Tesla A100 (2020), which includes 54.2 billion transistors with a processor size of 7 nm [52]. These graphics cards deliver 0.345 and 19.5 TFlops, respectively. Furthermore, there has also been an effort to reduce development time costs using unified memory space (Unified Virtual Addressing in NVIDIA or Heterogeneous Unified Memory Access for AMD). Global memory in the graphic card (DRAM technology) is conceptually organized into a sequence of byte segments and connected to the host through a high-speed IO bus slot, typically a PCI-Express and, in current high-performance systems, NVLink for NVIDIA [53]. Large differences can be shown, from Tesla C870 (PCIe 1.0 x16) 76.80 GB/s to Tesla A100 (NVLink 3rd) 1555.80 GB/s. The memory cache system has been improved, imitating the evolution in CPU, and in the new graphic cards, the complexity of the caches allows having no alignment memory access patterns with fewer penalizations [54].

This evolution has also been on the software side to exploit spatial locality with efficient representations, i.e., the multiplication of the vector-matrix with new ways to

store the spare matrix [55], highlighting the importance of the low occupancy for specific problems using instruction-level parallelism [56], creating more sophisticated libraries for parallel primitives such as reduction, sort, etc., for example, Thrust [57] or low-level primitives to improve the codification of collaborative algorithms on the GPU side [58].

The development of highly optimized libraries for computation has increased every year and helped to tackle new problems in a more productive way. Apart from libraries for Fourier transform (FFT), basic linear algebra (BLAS) or random number generation (rand), to name a few, there also exists the possibility of integrating Matlab with native kernels, which allows reducing the time of the computation for those expensive functions [59].

As it has happened to other fields in computer science, there has been a common effort to standardize the GPU Computing model. Although the standard OpenCL was released in 2008 [60], it is still a work in progress. The OpenCL idea promotes multi-platform adoption rather than being bound to a single vendor, and it is a collaboration among software vendors, computer system designers (including designers of mobile platforms) and microprocessors (embedded, accelerator, CPU and GPU) manufacturers. AMD's ROCm (Radeon Open Compute) is an open software platform to provide a heterogeneous ecosystem for computing on GPU with portability and flexibility [61]. Furthermore, there are also some libraries to alleviate the difficulties in programming the algorithms in the parallel landscape using GPUs. For example, OpenACC exposes a transparent way to parallelize code such as OpenMP, which is very convenient for a large community in research [62]. With the emerge of machine learning approaches, the use of GPUs for training and inferring has increased. This is the reason why there are also many libraries to make the most out of the GPUs with machine learning frameworks, i.e., Caffe [63], Tensorflow [64] or PyTorch [65].

Figure 2 shows the most significant milestones over time on different technologies (libraries, GPUs, frameworks, etc.) that have contributed significantly to the development of applications requiring GPU-based hardware acceleration for image processing.

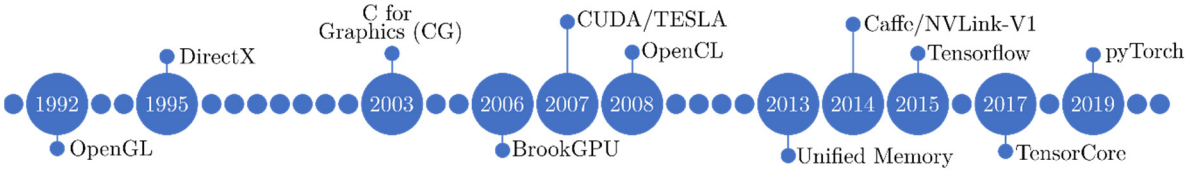

Figure 2. Most significant milestones over time on GPU-based hardware acceleration for image processing.

2.2.2. GPU Architectures in Medical Imaging

Many research fields have progressed and included GPU-based acceleration techniques for parallel programming with very significant results. In such scenario, medical imaging has been one of the most demanding fields for GPU computation, with GPUs found in nearly all imaging modalities, bringing high computation capabilities to the edge equipment in several applications [12,13,15]. Here, a few examples where GPUs played a crucial role in achieving faster computations and leading to real-time implementations are provided.

In 2010, Johnson et al. proposed an iterative GPU-based fat and water decomposition with an echo asymmetry and least squares reconstruction (IDEAL) scheme. They approximated the fat and water parameters and compared the Brent method with the search for the golden section to optimize the unknown parameter of MR field inhomogeneity (psi) in the IDEAL equations. They stated that their algorithm was more robust to the ambiguities of fatty water using a modified planar extrapolation of the psi method. Their experiments showed that the grease-water rebuild time of their GPU deployment methods could be rapidly and strongly reduced by a factor of 11.6 on a GPU compared to CPU-based rebuild [14,66].

Herraiz et al. presented a GPU-based implementation of the OSEM iterative reconstruction algorithm for 3D PET image reconstruction [67]. In this study, the authors

compared a GPU implementation using CUDA against a single-core CPU implementation, demonstrating a speedup factor of up to 72 times on a NVIDIA Tesla C1060 compared to an Intel Core i7.

Alcaín et al. proposed a very fast implementation of an algorithm for modality propagation/synthesis based on a groupwise patch-based approach and a multi-atlas dictionary [68,69]. They proposed and implemented an accelerated version of the patch-based modality propagation algorithm to compare the benefits of using multicore CPUs and CUDA-based GPU models, using both a global memory and a shared memory version. The evaluation of these GPU-based implementations demonstrated how the use of these techniques gained up to 15.9 times of speedup against a multicore CPU solution and up to about 75 times against a single-core CPU solution.

Punithakumar et al. compared an implementation of an image registration algorithm on the GPU versus the CPU. The goal of their research is to develop computationally efficient approaches for deformable image registration to find the point correspondence between image slices with the delineated cardiac right ventricle. The proposed approach offered a computational performance improvement of about 19 times compared to the CPU implementation while maintaining the same level of segmentation accuracy [70].

Florimbi et al. developed a multi-GPU support system that provides accurate brain cancer delimitation based on hyperspectral imaging constrained to providing a real-time response to avoid prolonging the surgery [71]. Their most efficient implementation showed the ability to classify images in less than three seconds.

Torti et al. presented a parallel pipeline for skin cancer detection that exploits hyperspectral imaging [72]. They showed how adopting multicore and many-core technologies, such as OpenMP and CUDA paradigms, and combining them led to a significant reduction in computational times, showing that a hybrid parallel approach can classify hyperspectral images in less than 1 s.

Zachariadis et al. introduced a novel implementation of B-spline interpolation (BSI) on GPUs to accelerate the computation of the deformation field in non-rigid image registration algorithms for Image Guided Surgery. Its implementation of BSI on GPUs minimizes the data that must be moved between memory and processing cores during input mesh loading and takes advantage of the GPUs large on-chip register file to reuse input values. They succeeded in reducing computational complexity and increasing accuracy. They evaluated the method on liver deformation caused by pneumoperitoneum, i.e., inflation of the abdomen. They managed to improve BSI performance by an average of 6.5 times and interpolation accuracy by 2 times compared to three state-of-the-art GPU implementations. Through preclinical validation, they were able to demonstrate that their optimized interpolation accelerates a non-rigid image registration algorithm, which is based on the free-form deformation (FFD) method, by up to 34%. Thus, the study showed the achievement of significant performance and accuracy gains with the novel parallelization scheme presented that makes effective use of the GPU resources. They showed that the method improves the performance of real medical imaging registration applications used in practice [73].

Milshteyn et al. proposed a fast, patient-specific workflow for online specific absorption rate supervision using a fast electromagnetic (EM) solver [74]. The MARIE® package used in their approach solves the EM fields in the patient accelerated on an NVIDIA Tesla P100 GPU, and the EM simulations required an average and standard deviation of 290.3 ± 67.3 s.

Another example of 3D image registration is proposed by Brunn et al. [75], who presented an implementation of a mixed-precision Gauss–Newton–Krylov solver for diffeomorphic two-image registration. The work extended the publicly available CLAIRE library to GPU architectures. Their algorithms managed to significantly reduce the execution time of the two main computational kernels of CLAIRE: derivative computation and sparse data interpolation. First, they implemented highly optimized, mixed-precision GPU kernels for the evaluation of sparse data interpolation; second, they replaced the first-order derivatives based on the fast Fourier transform (FFT) with optimized eighth-order finite differences;

and finally, they compared them with state-of-the-art CPU and GPU implementations. They showed that it is possible to record 256^3 clinical images in less than 6 s on a single NVIDIA Tesla V100. This is a speedup of more than 20× over the current version of CLAIRE and more than 30 times over existing GPU implementations.

With the development and improvement of new modern approaches, such as deep learning (DL) research in medical imaging, more efficient and improved approaches that were not previously feasible are being developed thanks to the evolution of GPUs. In this sense, training a DL model takes hours or even days, depending on its complexity and the amount of training data; however, presenting a new input to the already trained model provides results in seconds, which has led to a decrease in time acquisition and processing for several approaches.

Martinez-Girones et al. recently showed how the use of DL architectures took around 90 h for training a DL model used for head and neck MR-based pseudo-CT synthesis, but around 1 min to reconstruct the whole pseudo-CT volume in a NVIDIA RTX 2080Ti [76]. These results remark an affordable way to generate images of specific modalities from other different ones in real-time.

These works settle how the use of GPU architectures has enabled promising results in medical imaging acquisition, processing and enhancement in the past ten years due to their high efficiency in low time rates and the heterogeneity of their applications. Due to this fact, GPUs are powering the next generation of medical image algorithms, having a significant impact in medical imaging modalities such as CT, MRI, PET, SPECT, and US, etc. and, consequently, opening the way to innovative medical imaging applications.

2.3. Digital Signal Processors (DSPs)

2.3.1. Evolution of DSPs Architectures

While theories concerning digital signal processing date back as far as the 1960s, it was during the late 1970s that Speak and Spell™ from Texas Instruments first proved that DSPs could operate in real-time and be cost effective [77,78]. In fact, DSPs are a class of microprocessors whose architectures have been optimized for numeric processing operations and algorithms, with this way being faster, cheaper, and more energy efficient than usual microprocessors while still being reprogrammable.

DSPs are designed to measure, filter and/or compress continuous signals. They can obtain data and execute instructions simultaneously with low latency and power consumption. As a result, these devices typically do not require a demanding power system or cooling system, making them suitable for use in portable devices. Furthermore, this characteristic makes them less risky—in terms of their re-programmability and lifespan—and more cost-effective than ASICs for small productions [79,80]. Their evolution has been motivated by the custom algorithms they execute, as every feature is designed to increase the performance of the heavy computations carried out by the processor.

The main improvements performed over the years in the architectures of DSP processors are mostly related to parallelization of processes, enhancing memory access, and reducing the number of clock cycles per operation in the most common operations, multiply and accumulate (MAC) [8,81]. DSPs are specifically designed for digital signal processing, and particularly in image analysis, one of the common calculation algorithms is filtering, where multiply and accumulate (MAC) operations are commonly used.

In this way, DSP processors evolved to achieve a single clock cycle MAC, first introduced in 1982 by Texas Instruments. This feature is so important that modern DSPs have at least one dedicated MAC unit, and the number of MACs per second has become a measuring unit in the field. Despite these efforts, the MAC block still lies in most DSP critical paths to this day, impacting both their overall speed and power draw [82].

To be able to perform such specific operations as fast as possible, the architectures implement several execution units to parallelize the operations. Multicore chips have been critical since the early 1990s, when Texas Instruments launched the TMS320C40, the first multicore DSP capable of parallel processing [83]. As stated above, DSP processors excel at

MAC-intensive operations. Due to this, adding multiple cores sharply increases system performance through parallelization. At the same time, adding multiple-core architecture is the simplest way to achieve the above-mentioned parallelization [8,78]. These platforms also have at least three data buses for each computational unit so, with the same instruction, and in the same clock cycle, the data sample and the filter coefficient can be fetched, and the MAC result can be stored in the memory [84].

Each manufacturer offers different families of DSPs for general and specialized use. Typically, the general-purpose ones allow filtering, correlation, convolution, and FFT operations [85]. However, those for specialized use are more oriented to a more specific use in audio or video processing. For example, Texas-Instruments has designed various DSP families with different ranges of processing power and functionalities: ultra-low-power DSPs (cheap but low performance), optimized power DSPs (portable and mobile devices), and Open Multimedia Application Platforms (OMAP) [86–89]. Other processors are Digital Media Processors (DMP), which are designed for multimedia applications such as image and video capture as well as their processing. DMPs can perform more complex tasks at the expense of higher power consumption, i.e., hardware image and video codecs (MPEG, H.264 and JPEG) and hardware accelerators for video processing [90]. There are also multicore DSPs that are optimized for computationally complex tasks and high-performance computing (HPC) [91], which allow them to perform tasks in parallel. The maximum computational performance in multicore DSPs is obtained if the task can be fully parallelized so that the threads run on different cores simultaneously. However, this is often not feasible in practice, and advanced parallel programming techniques are required to optimize the computational speed of these DSPs [8].

Another factor to consider is the type of arithmetic calculation support (fixed point or floating-point operation). Floating-point operational support in DSPs makes algorithm implementation easier and increases precision compared to fixed-point units. In contrast, fixed point units can perform operations with fewer bits and higher speed.

Current multicore DSP systems can combine DSP cores with other types of cores, such as GPU or microcontroller cores. These are known as "heterogeneous" platforms. The most common type of heterogeneous platform combines CPU cores (ARM) and DSP cores. The former handles user interaction, protocol processing as well as controlling the platform. The latter handles compute-intensive tasks [78,92]. Furthermore, since a single MAC operation requires accessing two memory reads to obtain the data, plus the multiplication, the addition and the writing of the result in the memory in the same clock cycle, special long instructions have been developed, and DSP processors have their own specialized instruction sets. Furthermore, for better memory access, they incorporate direct memory access (DMA).

Some known manufacturers have developed a subgroup of processors known as Crossover MCUs ('crossover' embedded processors), which guarantee high performance (>400 MHz multicore CPU) and low consumption. These devices combine the best of the microcontroller world with high-performance DSPs that are ideal for machine learning and artificial intelligence applications. The price is affordable compared to other hardware platforms [93]. For instance, NXP offers an i.MX RT family based on a Cortex-M7/M33, supporting RTOS and providing 2D graphics support, a camera interface, and high-performance audio support [93]. These devices are application processors built with an MCU core, architected to deliver high performance and functional capabilities of applications processors but with the ease-of-use and real-time low-power operation of traditional MCUs. Further, crossover processors are designed to reduce system cost by eliminating the need for flash, external DDR memory, and power management ICs. Some of these elements appear on general embedded designs for mass-market applications such as metering, medical equipment, and IoT gateways.

As a result, modern DSP systems have a small power draw compared to other solutions, while also excelling at handling digital signal processing. Multicore solutions also

offer the ease-of-use attributes associated with microcontrollers, one of the drawbacks traditional DSP solutions have suffered from.

Figure 3 shows the most significant milestones over time that have contributed significantly to the development of imaging applications requiring DSP-based hardware acceleration systems.

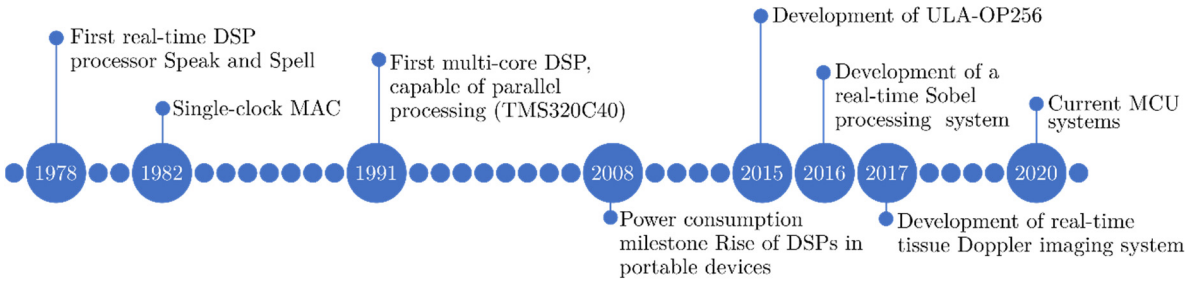

Figure 3. Most significant milestones over time on DSP-based hardware acceleration for image processing.

2.3.2. DSPs in Medical Imaging

There are several methods needed in medical imaging that depend on digital signal processing (DSP), such as convolution, discrete Fourier transform (DFT), fast Fourier transform (FFT), finite impulse response (FIR) and infinite impulse response (IIR) filters, FFT recursive and non-recursive digital filters, FFT processing, random signal theory, adaptive filters, upsampling and downsampling, etc. [94].

The DSPs' rising performance has allowed them to undertake intense real-time tasks, which, coupled with their inherent proficiency in arithmetic operations, has made them a desirable platform for certain medical imaging algorithms implementation.

Recursive and non-recursive digital filters are mainly used to acquire interest signals and block unwanted signals, i.e., noise. In general, low pass, high pass, band pass and band reject filters are implemented for filtering functions. It is common to employ these filters using DSPs for biomedical engineering fields such as MRI, ultrasound, CT, X-ray or PET imaging, as well as the analysis and processing of signals derived from these or genetic signals [94].

Berg et al. [95] proposed an optimization-based image registration algorithm using a least-squares data term and implemented it on an embedded distributed multicore digital signal processor (DSP) architecture. All relevant parts were optimized, ranging from mathematics, algorithmics, and data transfer to hardware architecture and electronic components. They evaluated the performance using histological slices of cancer tissue.

Chandrashekara and Sreedevi implemented contrast-limited adaptive histogram equalization on a TMS320C6713T DSP processor to improve contrast in brain magnetic resonance images [96]. They showed promising results for several image quality metrics and subjective visualization.

DSPs have often been used together with another accelerator. Liang et al. described a digital magnetic resonance imaging spectrometer based on a DSP along with an FPGA [97]. The DSP was utilized as the pulse programmer on which a pulse sequence is executed as a subroutine.

Ali et al.'s white paper and Pailoor et al.'s application report provided a description of portable ultrasound imaging equipment with emphasis on the signal processing strategies [98,99]. They reported how DSPs, oftentimes, can be complemented by a reduced instruction set computer (RISC) processor. This way, the DSP can perform the more demanding algorithmic and real-time control tasks, even though ensuring a low energy consumption, while the RISC processor handles the tasks the DSP core is less suited for, making up for its downsides without adding another device.

In this line, Boni et al. described the 256-channel ULtrasound Advanced Open Platform (ULA-OP 256), intended to afford high performance in a small size [100]. This research

scanner integrates several homogeneous multicore DSP and FPGA devices, where DSPs handle real-time operations (i.e., demodulation, filtering) as well as arithmetic operations. The ULA-OP 256 was successfully used in several consequent real-time image processing applications, such as the ones carried out by Ricci et al. [101], performing real-time blood velocity vector measurement over a 2-D region; Ramalli et al. [102], presenting a real-time implementation of the tissue Doppler imaging modality; and Tortoli et al. [103], comparing vector Doppler methods with conventional spectral Doppler approaches and achieving real-time estimations.

Arulkumar et al. proposed an ALU-based FIR filter for Biomedical Image Filtering application [104]. The ALU design operation includes accumulation, subtraction, displacement, multiplication and filtering. The FIR filter is designed to perform retina image filtering. This process allows DSP uses for improved visualization of the medical field. They concluded that the design and analysis of the ALU-based FIR filter provides an efficient result in the way of achieving the factors such as static and dynamic power, delay area utilization, MSE and PSNR.

On another note, Beddad and Hachemi showed encouraging performance results when implementing several medical image algorithms (such as Fuzzy C-Means or Level sets) for brain tumor detection in MR images [105].

All these works confirm that the use of DSPs is especially envisioned when designing specialized devices, particularly as their low power draw makes them exceptionally well-suited for portable devices. Additionally, there are many tools available for programming C/C ++ code for DSP, and the libraries available include common general-purpose algorithms for computer vision. In general, the development time for a simple task in single-core DSP is relatively short; nevertheless, developing optimized code using parallel programming techniques for multicore DSPs becomes hard and requires advanced programming skills. Moreover, free libraries have been developed to help programmers with optimized basic functions. Some of these libraries are: DSPLIB and MSP-DSPLIB (including functions for some digital signal processing tasks, such as FFT and convolution), IMGLIB and MATHLIB (basic math operations and basic practical calculations).

2.4. Field Programmable Gate Arrays (FPGAs)

2.4.1. Evolution of FPGA Architectures

FPGAs were introduced in 1985; however, a variety of Programmable Logic Devices (PLDs) appeared earlier [106,107]. The first PLDs were just Programmable Read-Only Memories (PROMs) that, rather than being used as computer memories, were used to implement simple logical functions. These devices integrate in the same integrated circuit a non-programmable memory decoder consisting of AND gates and a programmable array consisting of OR gates. Using the ROM input addresses as independent signals of a Boolean function and the memory outputs interfaced to the programmable array as a result of this Boolean function, it is possible to make independent logic circuits from the input signals. However, in some cases, they may be larger and slower than dedicated logic circuits and only a part of the capacity is used, which provides an inefficient use of the device.

New kind of PLDs appeared later in the 1970s to reduce the size of the PROMs or increase the configurability, such as Programmable Logic Array (PLA) and Programmable Array Logic (PAL) [108], which allow programming the AND gate array or the memory decoder, and the PALs also allow programming the output logic, reducing the number of components. The PROM, PLA and PAL are classified as Simple PLDs (SPLD).

In the 1980s, more advanced PLDs appeared, which were called Complex PLDs (CPLD) [109]. The architecture of CPLDs consists of various SPLDs blocks surrounded by a programmable interconnection matrix. This allows the development of larger and more complex logic circuits, reducing cost, size and increasing design reliability. Although the internal structure of a CPLD varies from vendor to vendor, the general structure is characterized by the following blocks: SPLD blocks, such as PROM, PLA or PAL; input and output (I/O) blocks that connect the I/O to the SPLDs; and the programmable interconnect

matrix responsible for interconnecting the inputs and outputs of the SPLDs, as well as the I/O blocks.

CPLDs hierarchical structure, combining registered SPLDs with a programmable interconnect matrix, produced a big leap forward in the capabilities of programmable logic devices. Nevertheless, CPLDs were not yet suitable for complex circuits that require many flip flops. At that moment, there was still a big gap between the circuits that could be designed using PLDs and ASICs, leading to the development of FPGAs.

The most common FPGA architecture is called island-style FPGA architecture, in which the logic blocks are surrounded by a grid of routing interconnect, such as connection boxes or switch boxes. The smallest unit of logic block is the Configurable Logic Block, which consists of a LUT, a flip flop and a multiplexer [107]. LUTs are composed of SRAM cells that store the function outputs and a multiplexer that selects the output value from the inputs, and it can implement any function from its inputs.

FPGAs accomplish the goal of implementing complex digital circuits on a single chip that was previously only achievable using ASICs. At the same time, its high configurability allows for fast design and modification times. However, this flexibility comes at a cost since most of the FPGA area is dedicated to the programmable routing interconnections, which not only implies much higher use of silicon but also higher power consumption and slower speeds than ASICs [110].

On the other hand, the flexibility of FPGAs is the main advantage compared to ASICs, since if the design had to be modified due to design errors or improvements, the FPGA could just be reprogrammed, but a custom digital circuit would need to be redesigned from scratch [111]. In addition, since nowadays FPGAs can be reprogrammed on-site within seconds, their design costs are significantly lower than ASICs'. Thus, ASIC devices are only economical when high production volumes are needed or where the ASICs performance cannot be matched by an FPGA [112].

The FPGAs have evolved from the homogeneous structure to a more heterogeneous architecture with a wide range of specific blocks. Nowadays, FPGAs contain embedded memory blocks (block RAM), multipliers (DSP), and even embedded processors cores, known as System-on-Chip FPGA (SoC FPGA) [113]. In addition, current multiprocessor system-on-chip (MPSoC) systems also include real-time processors, GPUs or AI modules [114,115] on the same die. These devices are growing more and more complex; therefore, the use of high-level synthesis tools allows further abstraction using C, C++ or SystemC to transform algorithm descriptions into register-transfer level code, allowing to speed up the hardware design process on the FPGA.

Semiconductor manufacturers are exploring alternative architectures, including specific accelerators for vector processing (DSPs, GPUs) and programmable logic (FPGA). Xilinx has introduced in recent years a new heterogeneous architecture called Adaptive Compute Acceleration Platform (ACAP), which is expected to be a solution for the current applications [116,117].

ACAP is a hybrid platform that integrates an FPGA, together with a programmable processor and software programmable accelerator engines (DSP and GPU) prepared for vector computation. In addition, a communication network (NoC) interconnects all the parts, which makes it possible to establish an efficient data exchange channel. This new architecture allows performing acceleration interfaces for Artificial Intelligence and Machine Learning applications, as well as Automotive Driver Assist System and 5G wireless communications, among others. Therefore, this new architecture can be an alternative for future medical image processing applications, since it includes all the architectures described in this article (DSPs, GPUs, FPGA and CPU) and the manufacturers are tending to integrate more elements in the same chip.

Therefore, FPGAs are now widely used in the implementation of image processing applications. This is especially focused on real-time applications, where latency and power consumption are important parameters of design. For instance, an FPGA built into a smart

camera can do much of the processing of the captured image, allowing the camera to deliver a stream of processed output data rather than a sequence of raw images.

Unfortunately, implementing an algorithm to an FPGA can produce disappointing results because many image processing algorithms have been optimized for a serial processor execution and for a specific image dimension. Generally, it could be necessary to transform the algorithm to efficiently exploit the parallelism and resources available in the FPGA.

Figure 4 presents the most significant milestones over time that have contributed significantly to the development of imaging applications using FPGA-based acceleration systems.

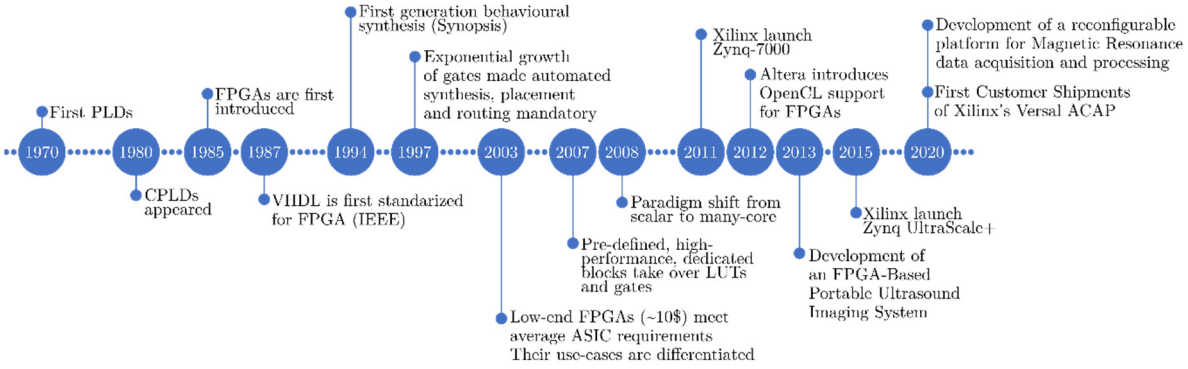

Figure 4. Most significant milestones over time on FPGA-based hardware acceleration for image processing.

2.4.2. FPGA in Medical Imaging

The evolution of FPGAs has motivated an increase in the use of these devices, whose architecture allows the development of hardware solutions optimized for complex tasks, such as 3D MRI image segmentation [118], 3D discrete wavelet transform [119], tomographic image reconstruction [18,120], or PET/MRI systems [121,122]. The developed solutions can perform intensive computation tasks with parallel processing, are dynamically reprogrammable, and have a low cost, all while meeting the hard real-time requirements associated with medical imaging.

Moreover, its reduced size and single chip capabilities make them the perfect platform to implement portable testing devices [123] or even surgical assisting imaging devices [124]. Configurable hardware solutions offer a compromise between the flexibility of software and the high processing speed of ASICs, at a lower cost than DSPs, and as reconfigurable devices, new algorithms can be implemented. Furthermore, parallelism and pipelining techniques become possible, increasing the FPGAs performance, even at slower clock rates. This makes FPGAs capable of performing more than 33 million operations per second [125], which is the number of operations per second needed to execute a single operation in real-time 768 $\times$ 576 color video at 25 frames per second.

FPGAs have been designed to perform image registration of medical images as well. Specific processes as affine transformations or mutual information calculations can be accelerated during real-time image registration, implementing the algorithms in FPGAs [126,127]. Nevertheless, recently, Mondal and Banerjee proposed an efficient and complete hardware-based image registration [128]. They used angular and radial projections without converting the image into polar coordinates. Therefore, they adaptively adjusted the number of samples according to the angle and the radial length in parallel. Moreover, they sped up the process by avoiding the calculation of geometric transformations in each iteration, which also reduces the amount of hardware resources that are required. Hence, they presented a high-speed hardware architecture able to perform the whole image registration process.

Gebhardt et al. proposed a system based on FPGAs to solve the electromagnetic interference problems to which PET detectors and the MRI radio frequency (RF) subsystem

in multimodal PET-MRI systems are exposed, resulting in a deterioration of the signal-to-noise ratio (SNR) [129]. The methods presented employed an FPGA that replaces clock frequency and phase shifting of the digital silicon photomultipliers (dSiPM) used in PET modules. A reduction in RF interference was achieved based on the principles of coupling and decoupling of the EM field from the RF receive coils. For this purpose, the clock frequencies were modified, and the clock phase relationships of the digital circuits were changed.

Zhou et al. solved the problem of detecting nuclei in high-resolution histological images by implementing a generalized Gaussian Laplacian algorithm on FPGA. The results demonstrated a significant improvement in processing time without loss of detection accuracy [130].

In addition, real-time image processing can be implemented in MRI normal/abnormal immediate classification, lowering the false-negative rates and increasing tumor detection by double reading the images [131]. FPGAs are ideal devices to implement functions and operations, as convolutions, or even neural networks for machine learning and DL, due to parallelism and pipelining where they outperform CPUs and GPUs [132]. Several proposals have been presented for MR image reconstruction. Li and Wyrwicz designed a single FPGA chip with a 2D Fast Fourier Transform (FFT) core to reconstruct multi-slice images without the need of additional hardware components [133]. More recently, different groups have presented FPGA designs intended to accelerate the MRI reconstruction through sensitivity encoding (SENSE) [134,135]. Thus, Inam et al. obtained results that were comparable to those acquired with conventional CPU-based implementations but achieved lower computational time [135]. Furthermore, FPGAs have been used for other MRI applications, such as spectrometry. Liang et al. use a DSP along with an FPGA-based design to build a digital magnetic resonance imaging spectrometer [136]. The FPGA oversaw the gradient control, RF generation and RF receiving.

According to other imaging modalities, innovative applications have been developed for CT with field programmable gate arrays. Choi et al. showed an FPGA design with several block RAMs to make 3D CT reconstruction faster and reduce the radiation exposure [137]. Likewise, FPGAs can be used to accelerate data acquisition of ultrafast X-ray CT in real-time, as Windisch et al. demonstrated [138]. This way, real-time control of the scanning process can be enhanced, which is one of the main limitations of this modality. In addition, Goel et al. proposed a new method to speed up the COVID-19 diagnostic process. For that purpose, they developed a deep convolutional neural network to classify infected patients from CT images [139]. They tested different hardware-based implementations, including multicore CPU, many-core GPU, and even FPGA. They obtained satisfactory results with the different platforms, demonstrating a competitive optimization of the network training and inference performance with the FPGA.

Moreover, many PET and PET/MR scanners include FPGA systems to decode the signals arriving at photomultipliers [140]. Thus, several groups have implemented FPGA-only digitizers for signal digitization, processing and communication. Moreover, the latest research built these highly integrated data acquisition (DAQ) boards using single-ended memory interface (SeMI) input receivers instead of low-voltage differential signaling (LVDS) input receivers [141,142]. Furthermore, different machine learning algorithms have been implemented for photon position and arriving time estimation, scatter correction, attenuation correction and noise reduction taking advantage of FPGAs [140].

FPGAs are also introduced to speed up the acquisition of US imaging as well as to reduce the computational cost. Thus, Assef et al. designed an envelope detector for US imaging using a Hilbert Transform finite impulse response filter on an FPGA [143]. The results showed high efficiency for real-time envelope detection with a cost minimization of up to 75%. In addition, the implementation of an FPGA system allowed Wu et al. to build an intravascular US device able to perform photoacoustic and ultrasound imaging with forward- and side-viewing capability (contrary to conventional only forward-looking in-

travascular US) [144]. The system presented high-speed, low-cost, flexible programmability and compactness, making the device suitable for both research and clinical environments.

Henceforth, the scope of FPGA applications in the medical image field is becoming increasingly widespread. They enable to design-specific processes separately and in parallel, speeding-up acquisition and processing. Moreover, recent developments have proven their capability to combine tasks in order to complete whole procedures as image registration. Therefore, FPGAs offer a broad range of possibilities to improve the current imaging techniques.

2.5. Application Specific Integrated Circuits (ASICs)

2.5.1. Evolution of ASICs

The beginnings of ASICs could be traced back at least 20 years before the development of masked read-only memory (ROM). In the early 1970s, the concept of Gate Arrays and Standard Cells was introduced, with the first ASICs using diode-transistor logic and transistor-transistor logic technology. On the other hand, complementary metal oxide semiconductor (CMOS) technology enabled the commercialization of gate arrays, with the first CMOS gate arrays being developed in 1974. However, ASICs acquired a prominent place in the integrated circuit market worldwide during the 1980s, when the first low-end personal computers became commercially available using circuits based on gate-array circuits. The evolution of ASIC designs and technology can be characterized by the continuous growth and development of various ASIC design styles, which can be classified into four groups: standard-cell designs, gate-array or semi-custom design, structured design and full-custom design [145–147].

The standard cell design was the first method that allowed enhancing this technology since, until then, to design an ASIC it was necessary to choose a manufacturer and use the design tools that it provided. However, manufacturers implemented standard cells that were functional blocks with known electrical characteristics, which could easily be represented in the developed tools. Standard Cell-based design is the use of these functional blocks to achieve very high gate densities while achieving good electrical performance. Standard Cells produce a design density with comparatively lower cost and can also integrate Intellectual Property (IP) and SRAM cores in an effective way, unlike gate arrays.

The gate-array design is a method where the transistors and the other active elements are predefined, and the wafers containing these elements are kept in stock prior to metallization, while the design process defines the interconnection of the elements in the final device. Fixed costs are significantly lower, and the production cycles are shorter. It should be noted that minimal propagation delays can be achieved with this method, compared to commercially available FPGA-based solutions.

In the structured design of ASICs [148], the manufacturing cycle as well as the design cycle are reduced compared to cell-based ASICs, due to the existence of predefined metal layers, which reduces the fabrication time, and a pre-characterization of the silicon, which reduces the design time. It has lower fixed costs than chips based on standard cells or fully custom-made, the predefined metallization is used mainly to reduce the cost of the mask set, and it is also used to reduce the development cycle. In addition, the tools used for structured ASICs can substantially reduce and facilitate the design since the tool does not have to perform all the functions required for cell-based ASICs. Besides, it enables the use of IP cores, which are common for certain applications or industry segments, instead of designing these cores from scratch.

Finally, the full-custom ASIC includes custom logic cells, as well as the customization of all mask layers, making them very expensive to manufacture and to design. An example of a full-custom ASIC is a microprocessor, where many hours of design time are involved to maximize the performance of the microprocessor area. This allows to include analog circuits or optimized memory cells in the same device, obtaining more compact designs although requiring a long design time.

Since feature sizes of ASICs are shrinking and design tools have improved over the years, the overall complexity and functionality of an ASIC has grown from 5000 logic gates to more than 100 million. Current ASICs often include complete microprocessors, memory blocks, as well as other large building blocks. For this reason, ASICs are sometimes called system-on-chip (SoC) since the current trend is to integrate all or most of the modules that make up a computer into a single integrated circuit [149].

Figure 5 presents the most significant milestones over time that have contributed significantly to the development of imaging applications using ASIC-based hardware acceleration systems.

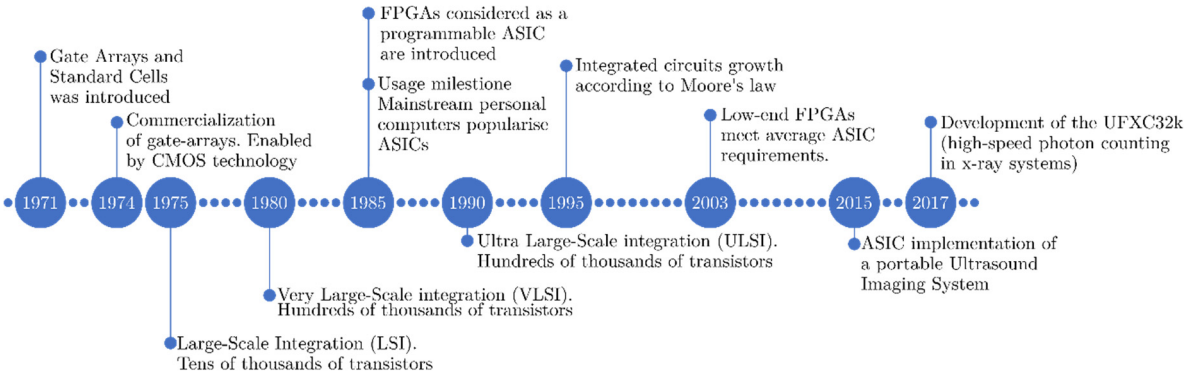

Figure 5. Most significant milestones over time on ASIC-based hardware acceleration for image processing.

2.5.2. ASICs in Medical Imaging

Despite the recent technological advances in CPUs, GPUs, FPGAs and DSPs, ASICs are still used to support a few more specific and more challenging processing tasks. Thus, ASICs have been traditionally used to support the high computational and data rate requirements in medical X-ray spectroscopy, CT scans, MRI, ultrasounds, and PET imaging systems, with the focus on the front-end receiver electronics. In this sense, several studies have proposed dedicated architectures to solve specific problems with very strict real-time constraints.

Kmon et al. presented an ultra-fast 32k channels chip for an X-ray hybrid pixel detector, which was designed to obtain a high count-rate performance [150]. Their dedicated and fast front-end circuit could be bump-bonded to a silicon pixel detector allowing for single-photon counting readout for hybrid semiconductor detectors in low-energy X-ray imaging systems. Another proposal in this field is described by Sundberg et al. in [151], where they use an already evaluated ASIC architecture capable of photon-counting [152] and propose adjusting the shaping time to counteract the increased noise that results from decreasing the power consumption.

In clinical and preclinical nuclear imaging applications, there is a rising demand for high-resolution room-temperature solid-state detectors (RTSDs), especially CZT and CdTe detectors. As the demand for SPECT systems with higher spatial resolution, energy resolution and sensitivity continue to rise, there is an increasing need to develop high-performance small-pixel CZT and CdTe detectors and corresponding high-speed readout electronics. The multi-channel readout circuitry reported in [153] is based on the pre-existing High Energy X-ray Imaging Technology (HEXITEC) ASIC [154]. Pushkar et al. offered a different solution to the same problem, although there are not many details of their SPECT ASIC [155].

Kang et al. described a novel SoC solution for a portable US imaging system that was compact and had low power consumption [156]. In this case, their solution comprised all the signal processing elements, including the transmit and dynamic receive beamformer modules, mid- and back-end processors, and color Doppler processors, in addition to an efficient architecture for hardware-based imaging methods, including dynamic delay computation, multi-beamforming, and coded excitation and compression. Kim et al.

presented an ultrasound transceiver ASIC directly integrated with an array of 12 × 80 piezoelectric transducer elements to enable next-generation ultrasound probes for 3D carotid artery imaging. It is a second-generation ASIC that employed an improved switch design to minimize clock feedthrough and charge-injection effects of high-voltage metal–oxide–semiconductor field-effect transistors, which in the first-generation ASIC caused imaging artifacts [157]. Rothberg et al. described the design of the first ultrasound-on-chip to be cleared by the FDA for 13 indications, comprising a two-dimensional array of silicon-based microelectromechanical systems ultrasonic sensors directly integrated into complementary metal–oxide–semiconductor-based control and processing electronics to enable an inexpensive whole-body imaging probe [158]. The beamformers of ultrasound imaging are conventionally implemented using FPGA. However, the minimum delay time provided by an FPGA chip is about 2 ns, which is not suitable for high-frequency US (HFUS) imaging. As a result, an ASIC is needed to provide an appropriate delay time to excite the array transducer elements in the HFUS array system. Sheng et al. proposed an all-digital transmit-beamforming integrated circuit with high resolution for high-frequency ultrasound imaging systems, not only more suitable for portable ultrasound system application but also at a lower cost [159].

Cela et al. reported a compact detector module, including a FlexToT ASIC designed for time-of-flight (TOF) PET [160]. Novel TOF sensors allowed for an additional level of detail in PET imaging, as they can add the actual time difference between the detection of photons released during an annihilation event (compared to normal detectors that only measure the direction and attenuation of photons). Their module provided a fast, low-power front-end readout for silicon photomultiplier (SiPM) arrays in scintillator-based PET detectors. An evolution of this ASIC, named HRFlexToT, was presented in 2021. It offers a linear Time-over-Threshold (ToT) with an extended dynamic range for energy measurement, low power consumption, and an excellent timing response [161]. Nemallapudi et al. [162] described the use of STiC, an ASIC with four channels dedicated to fast timing discrimination in PET using silicon photomultipliers [163], for range verification in proton therapy.

All these works confirm that the use of ASICs is still needed for specific and more challenging processing endeavors and serves as examples of very specific tasks where other hardware architectures could fail to meet the strict real-time constraints.

3. Discussion

Earlier prediction and treatment have been driving the acquisition of higher image resolutions as well as the fusion of different modalities. On the other side, with the demand for healthcare services increasing, providers require medical imaging equipment that acquires images faster and improves image quality. In this context, the rising need for sophisticated software/hardware systems for medical image acquisition and storage, as well as novel and nearly real-time medical imaging analysis tools, has led to a change of paradigm when choosing the most efficient hardware architectures while keeping healthcare affordable and accessible.

3.1. Hardware Architectures Comparison

Today, manufacturing technologies are evolving rapidly, and competition between different manufacturers has brought better development tools, which allow creating solutions in a reasonable time and equipped with the latest technological advances. However, each hardware accelerator is designed to be efficient for certain algorithms or implementations but not for others. In fact, choosing a hardware accelerator is often a compromise between several factors, including computing power, speed, development time, power consumption, and price. Therefore, this makes choosing a suitable hardware accelerator for a specific application relatively difficult.

One of the main challenges lies in finding a suitable solution for an application that requires the use of hardware accelerators. It is difficult to define an indicator that makes it easy to choose one hardware accelerator over others. A more realistic analysis indicates that

processing speed alone is not enough as a criterion of choice, especially when they belong to different families. Thus, there are other objective factors ranging from power consumption and price to more subjective ones related to the skill of the programmer and the design tools available. Figure 6 shows how each of the five hardware architectures discussed in this review excels in different parameters, so aspiring designers should consider the strengths of each one before choosing a platform to develop their prototype and, eventually, their final application. In this sense, the following definitions of the parameters shown in Figure 6 are considered for comparison purposes:

- Flexibility: Ability to adapt the platform to different scenarios and use cases while keeping performance high enough to be a competitive option (higher is better).
- Design cost: Economic cost of resources needed to produce and program the device. This includes coding easiness (higher rating means lower requirements).
- Unitary cost: Cost per manufactured unit. This cost does not include coding or implementation expenses and is influenced by common manufacturing runs (higher rating means lower cost).
- Performance: Ability to complete a given (appropriate) task (higher is better).
- Power efficiency: Power required to operate (higher rating means lower power consumption).

CPUs have been typically chosen as the most common and flexible platform to code or develop on due to their inherent flexibility and easiness to program new methods. Additionally, their computational performance has increased considerably in the past decades (Figure 6a).

GPUs offer a further dedicated technology providing a way to improve certain computational tasks and algorithms while maintaining a certain amount of flexibility (Figure 6b). However, the development and implementation of algorithms in GPU architectures has been very complex until the development of a unified architecture, which helped to have a better load-balancing in exchange for complex hardware. Furthermore, the use of GPUs by a broader public has driven the development of new functionalities as well as the creation of several highly optimized libraries such as OpenACC, Caffe, Tensorflow or PyTorch.

DSPs specialize, as mentioned earlier, in digital signal processing, and as such, are the best performers in that field. Nevertheless, they barely offer any flexibility, so they are often used together with other architectures in composite designs where fast digital signal processing is desired, yet it is not the only kind of processing performed (Figure 6c).

FPGAs have less performance, higher power consumption and unitary cost than ASICs but advantages of higher flexibility, lower design effort and time-to-market. Moreover, since FPGA design times are considerably shorter than dedicated ASICs', they are also used to prototype the ASIC design. Compared to CPUs and GPUs, FPGA performance can be higher, but with the cost of a higher design effort (Figure 6d). In addition, it is possible to simulate functionality in the early stages of the design and validation process [106], including the hardware–software interactions and communication with other devices. These features provide FPGA-based devices ideal for a large application range, from prototyping to aerospace and defense [164,165]. In general, FPGAs are used in communications, video, and image processing because of their parallelism, which allows them to meet real-time requirements.

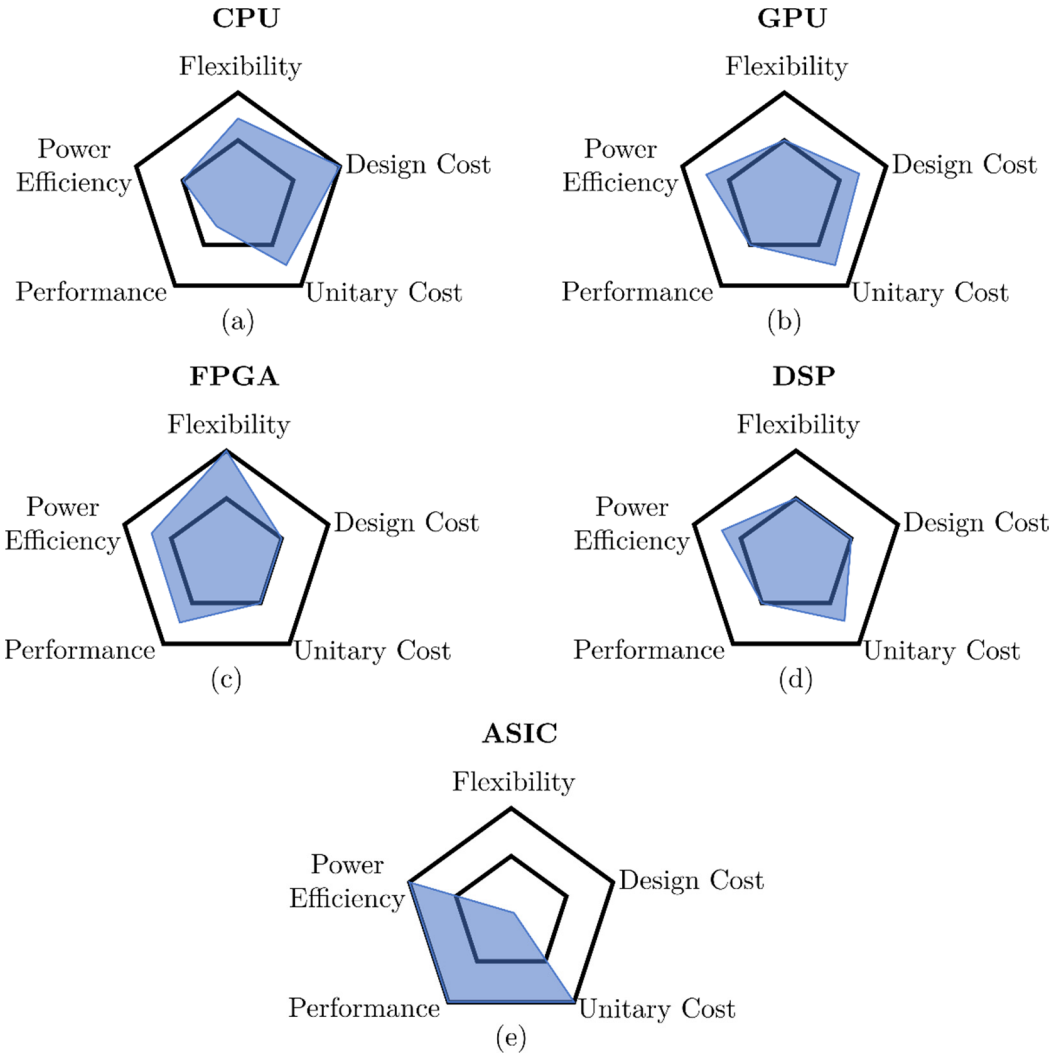

Figure 6. Summary of the comparison between the five hardware architectures discussed in this paper, based on different parameters to be considered when implementing image processing for medical imaging applications. In detail, (**a**) Central Processing Units (CPUs) are probably the most widespread technology, as every PC is equipped with one of them. Their availability, flexibility and ease-of-use make them the perfect prototyping candidates when deploying algorithms or developing code. (**b**) Graphics Processing Units (GPUs) offer more raw power than a CPU while being less flexible. (**c**) Digital Signal Processors (DSPs) are cheap and easy to use but do not perform well outside their niche. (**d**) Field Programmable Gate Arrays (FPGAs) offer a unique combination of flexibility and high performance, as they can be programmed to perform similarly to a given highly specialized platform. (**e**) Application Specific Integrated Circuits (ASICs) are extremely geared towards high-performance and low unitary cost for a high production so that the low variable costs may offset the high fixed costs.

ASICs provide smaller form factors, higher performance, and less power consumption since they are tailored for a specific application and implemented based on custom design specifications. However, ASICs have a long time to market as there is a need for the layout of masks and manufacturing steps, with its subsequent risk of lost revenue due to being late to market, implementation flexibility and future obsolescence. In summary, ASICs should only be considered by experienced engineers developing high volume-specific devices, especially as they only add up in very well tested systems/subsystems where a possible change or update does not make much sense and, above all, security is not compromised (Figure 6e).

Table 1 provides a summary comparison chart for all five hardware architectures discussed in this review, including a brief overview of the architecture, the specific processing

characteristics, programming capabilities, supported peripherals, strengths, weaknesses, and most common medical applications.

Table 1. Summary comparison chart for CPU, GPU, DSP, FPGA and ASIC hardware architectures.

	CPU	GPU	DSP	FPGA	ASIC
	Central Processing Unit	**Graphics Processing Unit**	**Digital Signal Processors**	**Field Programmable Gate Arrays**	**Application-Specific Integrated Circuit**
Overview	Traditional sequential processor, optimized for executing general computational instructions.	Designed mainly for graphical outputs; now used in a wide range of computationally heavy applications (e.g., training AI models).	Dedicated to processing digital signals such as audio. Designed to perform mathematical functions such as addition and subtraction at high speed with minimal energy consumption.	IP blocks and logic elements that can be modified or configured in the field by the designer.	A customized embedded circuit designed for a specific application.
Processing	Single and Multicore MCUs MPUs.	Thousands of identical processor cores.	A parallel architecture is suitable as it is scalable and allows performance based on the number of processors in the system. Current multicore DSPs can be combined with other structures (e.g., CPU cores (ARM) and DSP cores).	SoCs include hard or soft IP cores, and they are configured for application.	Configured for specific application, and they could include third-party IP cores.
Programming	OSes, multitude of high-level languages via APIs; OpenMP, Cilk++, TBB, RapidMind and OpenCL; assembly language.	OpenCL, HIP (AMD) and NVIDIA CUDA API allow general-purpose programming (e.g., C, C++, Python).	C, C++	Traditionally HDL (Verilog VHDL); The newer systems include C/C++ via OpenCL and Vitis.	Application specific. For example, Google Coral TPUs to work with Tensorflow tensors or CPU manufacturers such as Intel include tools with ASICs.
Peripherals	Analog and digital peripherals in MCUs include digital bus interfaces.	Very limited, e.g., only cache memory and interconnection among GPUs (multi-GPUs).	Program memory, data memory, BUS.	Transceiver blocks and configurable I/O banks.	Tailored to application. ASICs can include standard connectors such as USB, Thunderbolt, Ethernet, etc.
Strengths	High versatility, multitasking and ease of programming.	High-performance processing in specific applications, such as video processing, image and signal analysis and neural network training, deep learning.	They have an optimized architecture to perform the computational operations in digital signal processing with low cost and high performance per watt.	Application-specific configuration, and this configuration could be changed for new fields. High performance per watt. Allows parallel operation.	Tailor-made for the application with an optimal combination of performance and energy consumption.
Weaknesses	The operating system adds a large overhead. It is optimized for sequential processing with limited parallelism.	Not suitable for all programs or algorithms. It is energy-intensive and becomes hot. Problems must be adapted to take advantage of parallelism.	They barely offer any flexibility.	Long development time and difficult programming. Low performance in sequential operations. Not optimal for floating-point operations.	Long development time with a high price tag. Cannot be changed once designed.
Most common medical usage	Image reconstruction, registration, segmentation.	3D PET image reconstruction, EM simulations, Pseudo-CT synthesis with MRI.	Ultrasound filtering, image registration, image enhancement.	Optimized 3D MRI image segmentation, TC/PET/MRI image reconstruction, image registration	Time-of-flight (TOF) PET, X-ray hybrid pixel detector, CZT and CdTe detectors for SPECT, specific US solutions.

Despite having analyzed each of the different hardware architectures separately, nowadays, it is difficult to find any one of the five types of devices discussed in this paper working independently of each other in image processing pipelines. An example of this is GPUs, which also need memory and usually work together with a CPU to manage data transactions. Moreover, frequently, there is no optimal solution for a specific application but a well-suited solution for a compromise between different requirements. It is also worth bearing in mind in what way these selections line up in numerous shared applications. As exposed in Table 1, developers can often base their designs on several or all the choices either alone or, often, in combination.

On the other hand, a recent trend of using FPGAs as partially reconfigurable, highly specialized circuits has set them under the spotlight. MPSoC devices are characterized by integrating different elements on the same chip. Although they have less computational power than CPUs or GPUs, however, when combined with a programmable logic of an FPGA, they are a very flexible solution. In addition, the current microprocessors integrate SIMD units, making these devices more commonly used for real-time image processing. These platforms tend to use FPGAs to speed up algorithm implementation while leveraging their onboard CPU to perform general tasks or run software applications. Their advantages include the ability to provide high-speed customizable solutions due to the parallelism it allows in the design. The programmable logic in the same device makes it possible to design hardware accelerators combining the advantages of FPGAs together with those of multicore systems.

Semiconductor device manufacturing techniques require increasingly complex techniques to allow the integration of a greater number of transistors, functional blocks, memories and buses, all on the same chip, obtaining devices with higher computational performance [166]. This increases the cost of most CPUs, GPUs, MPSoCs or FPGAs but also allows higher performance and the ability to parallelize and speedup algorithms, as well as greater flexibility in the different fields of application.

Considering the devices in terms of their flexibility and parallelism capabilities, it can be considered that MPSoCs with FPGAs allow using the programmable logic of the FPGA to apply parallelism, while different applications can be developed on the SoC processors. Slightly less flexible are CPUs and GPUs since it is possible to parallelize very effectively using the GPU, but it is not possible to make specific hardware accelerators as it happens with FPGAs. Finally, ASICs would be the least flexible, because they are usually designed for a specific application and development; therefore, if the ASIC is designed to be flexible, it may become a very costly solution.

Taking performance into account, it can be said that MPSoC devices with FPGAs have lower performance than a solution based on ASICs. Nevertheless, depending on the processing, their performance could be higher than a solution based on CPUs or GPUs [167–169]. The flexibility provided by MPSoCs makes it possible to use a large part of the programmable logic to develop hardware accelerators, while parallel processing is performed on the microprocessor cores, implementing mixed hardware/software solutions.

Finally, it should be noted that the trend of manufacturing devices that integrate a greater number of specific elements on the same die, known as heterogeneous architecture, allows for more complete solutions; however, it requires a high level of specialization from designers to obtain optimal solutions adapted to the chosen hardware. This means that, despite having a very powerful heterogeneous architecture, sometimes not all the embedded components are used, making inefficient use of the device. However, this statement is somewhat relative since a better solution can be achieved depending on the challenge imposed by the application, the selection process carried out by the developers, and their experience in similar tasks.

Multicore units (whether heterogeneous or homogeneous) are tailored to either benefit from the added strengths of two different architectures or to offset their weaknesses. As such, they have become the golden standard in many applications and commercial products. However, to properly utilize a multicore unit, a correct understanding of its cores

is needed. To this extent, this paper focuses on these independent hardware platforms while occasionally touching on specific multicore applications.

3.2. Advantages and Disadvantages of Hardware Architectures for Real-Time Medical Imaging

A compilation of the advantages and disadvantages of the most relevant characteristics for the different architectures analyzed has been summarized in Table 1 and Figure 6. With all details about the different hardware architectures suitable for real-time medical image processing in mind, one of the main challenges in medical imaging applications lies in selecting the right image processing engine. The ability of an architecture to manipulate images according to medical imaging standards (increasingly used in different services in the hospital) in a certain time gap places a huge load on both the selected processor and the associated system. However, the specific processing requirements depend on the explicit application and, thus, general characteristics such as speed, resolution and cost will be important considerations for both the designer and the end-user.

Medical imaging requires a high level of processing power, especially in parallel computing, when real-time analysis and processing are required. In this sense, any solution or improvement that arises can be developed mainly on three main levels:

1. Lower level: It deals with pixel operation functions, differences between images and filtering using kernels. These operations are usually repetitive and exhaustive processes that tend to increase the operations number despite the simplicity of the operations.
2. Intermediate level: The segmentation, the movement estimation and the extraction or coincidence of characteristics (i.e., image registration). It requires a large use of operations and memory.
3. Top level: It deals with the interpretation. This function requires prior knowledge of the environment and even the application of Artificial Intelligence techniques.

The challenge of channeling these three tiers in real-time can be accomplished using parallel architectures and generally depends on the assigned operations of each part of the medical imaging system. Additionally, to allow a degree of flexibility and scalability in the processing system, a modular design approach is preferable, as it allows additional processors to be added if the original specification turns out to be insufficient or a system design with some degree of scalability is sought. Therefore, a parallel architecture is suitable and desired as it is scalable and will allow performance based on the number of processors in the system.

In this sense, the use of CPUs as the desired hardware platform for real-time medical imaging processing has been recognized. Indeed, this type of architecture has been considered the more accessible and flexible architecture to prototype and deploy algorithms. Some of the advantages of using CPUs are explained now:

- They are an adequate platform for prototyping due to the existence of many ubiquitous optimized libraries for tackling image processing and mathematical problems.
- There exist standards (OpenMP) to facilitate multithreading programming.
- They offer good performance in problems that require serial processing.
- They can be part of computation clusters or supercomputers.

Nonetheless, there exist several drawbacks:

- A combination of multicores and vectorization is needed to obtain the best speed up. This implies a deep knowledge of hardware and software and heavily timed optimization processes.
- Difficult to control the exclusivity of the memory, as there are more processes running on the same operating system.

However, due to the huge improvements in GPU architectures during the past decade, real-time medical imaging processing has advanced after including GPU-based acceleration techniques for parallel programming, with GPUs found in nearly all imaging modalities, bringing high computation capabilities to the edge equipment in several applications.

Additionally, with the development of novel methods such as DL research, more efficient and improved approaches are now feasible. Some of the advantages of using GPUs are:

- They offer a good performance/cost ratio.
- Coding productivity has been improved in recent years.
- GPU architectures have been adapted to cover different algorithms that were not suitable at the beginning, such as branching or bank conflicts in shared memory.
- Many high optimized libraries have been developed for many tasks such as machine learning, signal, and image processing, etc., increasing their suitability for medical imaging problems.
- More efficiency in medical image processing tasks, generally, due to GPU ecosystems being designed and optimized for working with large amounts of data.
- Manageability of cloud ecosystems with powerful GPUs, which eliminate the necessity of having local equipment.

Nonetheless, there exist several notorious drawbacks:

- Data transfer between general and dedicated memory limits some real-time applications, generating a memory transfer/exchange bottleneck.
- Difficulty to migrate code into other GPUs vendors, unless using OpenCL and HIP.
- Requirement of deep knowledge of CPU and GPU programming languages for specialized medical imaging solutions involving GPUs.
- Not all image processing tasks perform better on a GPU than on a CPU.

The rising performance of DSPs has allowed them to undertake heavier real-time tasks, which, coupled with their inherent proficiency in arithmetic operations, make them the desirable platform for implementing certain medical imaging algorithms. Some advantages of using DSP are:

- The development costs of a portable system are relatively low, helping to lower the prices of each part of the imaging equipment.
- Generally, medical image processing tasks involve sequential algorithms, and the chip architecture of DSPs is optimized to do it. Multi-core DSPs have added the ability to implement coarse-grained parallelism for algorithms with a low to medium level of complexity.
- Power consumption in DSPs is relatively low. There are DSP families specially designed for mobile and portable applications, being a great choice for portable imaging equipment, such as portable ultrasound imaging equipment.
- Development time for simple single-core DSP computer vision and image processing algorithms is generally relatively short. There are free libraries and operative systems available for imaging processing.
- DSPs used to incorporate standard communications ports (e.g., USB, SATA, RS-232 and ethernet). It facilitates storage and transmission tasks between medical devices.
- Some DSPs are compatible with audio and video codecs, making them suitable for video processing applications on portable devices.

Nevertheless, there are some known disadvantages:

- Multi-core DSPs are designed for low to medium complexity HPC applications mostly. Thus, they are not suitable for high-speed or high-data throughput applications, and their use is not recommended for imaging devices that generate great amounts of data.
- DSPs are better suited for sequential processing but not a suitable option for increasing processing speed in massively parallel imaging algorithms.
- It is generally not efficient to use DSP in conjunction with CPU in PCs. Both have a similar sequential processing nature, while CPU programming is easier and more efficient than using DSP.

DSPs also seem to be relegated as a secondary/complementary option due to the performance improvement of other architectures, such as those used in the newest FPGAs. As stated, the evolution of FPGAs has motivated an increase in the use of these devices,

as their architecture allows the development of hardware solutions optimized for more complex medical image processing tasks. Some of the advantages of using FPGAs are:

- It is possible to perform massive arithmetic operations per clock cycle, and these operations can be performed in parallel. As stated before, for the case of GPUs, this is of great interest, as a huge amount of medical image processing and analysis algorithms are inherently parallelizable.
- Different image algorithms implementations or designs are feasible using the same hardware, which provides flexibility of application.
- The designer can create prototypes quickly, simulate, measure time constraints, and modify them, making FPGAs a very interesting option for real-time medical image processing applications using High-Level Synthesis (HLS) tools, such as Vitis HLS.
- The programmable logic of current FPGAs is huge, allowing more complex algorithms to be implemented in hardware. Thus, FPGAs are excellent devices to implement functions and operations, such as convolutions, or even neural networks for machine learning and DL.
- In the mixed hardware/software architectures, it is possible to implement a hardware solution versus a software solution to achieve some algorithm acceleration.
- The integration of the recent FPGA devices includes RAM memory, ARM processors and even DSP engines on the same die. This makes it possible to develop more complex applications by developing heterogeneous architectures where a large part of the elements contained in the FPGA are used.

Nonetheless, there exist several notorious drawbacks:

- Image processing algorithms are often highly complex and computationally expensive, requiring many logical resources. Thus, it is necessary to consider the cost of FPGAs since the more programmable logic integrated into the FPGA, the more expensive the device.
- Although it is possible to create prototypes quickly, it has long design cycles to optimize the resources consumption and achieve the time constraints, as well as making sure that the hardware designs reach a balance between area occupied and parallelization. In addition, the compilation and synthesis processes can take some time, depending on the design, and are related to the designer experience

Considering the customization capabilities of the hardware, both ASICs and FPGAs can achieve a better system performance compared to other technologies analyzed throughout this work. However, as both technologies differ in the internal structure of the logic block construction, they have very different results in areas such as speed, power consumption, cost, and degrees of integration, etc. In general, designs based on ASICs are optimized using a wide variety of logic cells with different sizes and characteristics along with dedicated interconnections. In contrast, FPGAs are designed to optimize flexibility through programmable logic components and programmable interconnections. Therefore, ASICs are considerably faster than FPGAs.

On another note, in the case of more specialized and complex medical equipment, such as medical X-ray spectroscopy, CT scans, MRI, ultrasounds, and PET imaging systems, ASICs are still used to support more specific and more challenging high computational and data rate processing tasks, with an emphasis on the front-end receiver electronics. Some advantages of ASICs are:

- The chip size is reduced, resulting in ASICs with high levels of customization. Many circuits can be integrated on the same chip, making it ideal for high-speed applications.
- Libraries in the case of standard cells reduce development time and design complexity.
- Minimal signal routing and timing issues since there is minimal routing to interconnect the different circuits.
- Low power consumption. The ASICs are designed to use all parts of the circuit, which implies high energy efficiency.

However, there exist several notorious drawbacks that make it difficult to use ASICs in medical imaging applications:

- ASICs in general have a high cost and development time.
- The design of ASICs requires greater design skill and complexity, which increases the price per unit.
- ASICs have limited programming flexibility because they are custom chips created for a specific application.
- ASICs require more time to market.

To ensure programmability, many FPGA devices use pass-through transistors to connect dynamically different logic elements. This feature adds delays to the interconnection of signal paths. In contrast, ASICs provide the ability to use optimal buffered cables. Another factor that contributes to the degradation of FPGA speed is its logical granularity since LUTs with a fixed number of entries are used. Any logical function with a few more input variables will require additional LUTs, which introduces additional routing and delay. In contrast, ASICs logic functions can be adjusted during the synthesis process to meet a better time constraint.

Active routing on FPGAs also introduces additional capacitance, which is combined with large capacitances caused by the length of the fixed interconnection path. Thus, the capacitance in the FPGA signal path is generally greater than that of an ASIC. This drawback also increments power consumption dissipated during signal switching that drives such signal paths.

The logical density in an FPGA is usually lower than ASICs because the active routing device occupies an important chip area. In general, this situation is related to the increment of the cost per unit, making the ASIC design preferable for systems ready for massive production.

However, it is important to stress that algorithms, such as feature extraction and tracking, are best suited for software implementation. With the facilitation of various FPGA tools, the interaction between software and hardware can be easily verified on an FPGA platform. Minor hardware/software changes are easier and more feasible compared to ASICs-based systems. This will make it easier for hardware designers to focus primarily on the architecture and the task of logical design.

When it comes to the amount of time it takes to design and develop a system based on FPGAs versus ASICs, FPGAs are generally easier and faster to produce and can be reprogrammed in the field as needed, so there is less need for testing and verification processes such as that of ASIC's where the process is more complex, often requiring custom design and a multifaceted design flow requiring a higher specialization degree.

3.3. Concluding Remarks and Outlook to the Future

In this review, the authors discussed how modern hardware architectures have advanced to deliver competitive clock rates allowing them to be used in high-capacity imaging equipment. Additionally, all five types of hardware architectures described have benefited from increased logic density in recent years, along with other features such as integration of dedicated memory and high-speed serial input/output. Intrinsically, all of them may offer competitive solutions in the medical image processing field depending on the specific purpose/system requirements and flexibility.

Knowing all advantages and disadvantages of the different hardware architectures suitable for real-time medical image processing discussed in this review, it can be affirmed that choosing the right hardware architecture for the implementation of Real-Time Medical Image Processing is a very complex task. On the one hand, there should be a very well-defined premise of the underlying medical imaging problem and, based on that, a very good establishment of the medical needs and constraints, which, usually, are very demanding. On the other hand, the designer/developer should be aware of the advantages and drawbacks of each hardware architecture to be able to meet all the different needs

and constraints established from the medical perspective (including time constraints and cost limits).

However, as mentioned before, despite having studied each of the different hardware architectures unconnectedly, currently, it is hard to find any of them working independently in medical imaging processing workflows. In the case of medical imaging, it is important to note that solutions based on CPUs together with GPUs are the most used since they allow fast prototyping, and their performance is optimal for image processing. Moreover, the cost of these devices has been reduced in recent years, along with an increase in their performance and parallelism capabilities. Finally, it should be noted that the cost of developing applications on CPUs and GPUs is very low since these devices support multilanguage coding and there exist a multitude of optimized libraries to obtain the best performance from these devices.

For example, it would be evident that, by increasing the quality of medical images (i.e., spatial and contrast resolution), physicians could better diagnose or monitor their patients. Consequently, with the emergence of Ultra HD displays, the new challenge will be to design processors to reliably represent these high levels of image quality and clarity, which is currently a clear trend for the audiovisual electronic industry. However, basing the design only on CPUs would be intended for applications with a market life of a few years only. In medical applications, this problem may be more evident because (i) higher image resolutions and deeper (more gray levels) contrast resolutions are required for these types of applications, (ii) it is not possible to assume an unreliable representation of those resolutions, and (iii) it is not possible to adopt a product redesign/upgrade within a few years, given the tedious, expensive, and time-consuming regulatory framework adaptation.

In this sense, the use of heterogeneous architectures, including different combinations of multicore designs counting with CPUs, GPUs, DSPs, FPGAs, or even small ASICs, in medical imaging is on the rise. The case of a PET/MRI could serve as an example: specialized ASICs perform the highly demanding real-time data acquisition of TOF PET; MPSoC combining CPU, FPGA and DSP can be used for optimized 3D MRI segmentation and other DL-based algorithms that exploit the flexibility of this hardware; GPUs oversee the 3D PET image reconstruction and the MRI to CT synthesis for PET attenuation, while CPUs are used for the user interaction.

Thus, the main arising challenge for both manufacturers and designers may be to facilitate the integration of the different hardware architectures, alleviating the programming difficulties associated with the heterogeneous solutions needed in the field of medical imaging.

Author Contributions: Conceptualization, E.A., P.R.F., R.N., A.S.M., F.M. and A.T.-C.; investigation, E.A., P.R.F., R.N., A.S.M., J.V., A.G.-B., P.M.M.-G., C.P.-d.-l.-L., B.R.-V., M.B., C.R.-S., I.Y., F.M. and A.T.-C.; writing—original draft preparation, E.A., P.R.F., R.N., A.S.M., J.V., A.G.-B., P.M.M.-G., B.R.-V., M.B., F.M. and A.T.-C.; writing—review and editing, E.A., P.R.F., R.N., A.S.M., J.V., A.G.-B., P.M.M.-G., C.P.-d.-l.-L., B.R.-V., M.B., C.R.-S., I.Y., N.M., S.B., F.M. and A.T.-C.; visualization, E.A., R.N. and J.V.; supervision, A.S.M. and A.T.-C.; funding acquisition, A.S.M., N.M., S.B. and A.T.-C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by Young Researchers R&D Project Ref M2166 (MIMC3-PET/MR) financed by Community of Madrid and Rey Juan Carlos University (PI: A. Torrado-Carvajal); by RETOS-COLABORACIÓN Project RTC-2017-6218-1 (QUASAR) financed by the Ministry of Science, Innovation and Universities and the European Regional Development Fund (PIs: N. Malpica and S. Borromeo); and by Comunidad de Madrid and European Structural Funds (FOTOCAOS project, Y2018/EMT-5062) as well as the support of the Spanish government, Ministerio de Ciencia, Innovación y Universidades, RTI2018-098743-B-I00 (MICINN/FEDER) (PI: A. S. Montemayor) and Puente Project Ref M2421 (RESOL) financed by Rey Juan Carlos University (PIs: I. Yahyaoui and M.C. Rodriguez-Sanchez).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Iglehart, J.K. The New Era of Medical Imaging. *N. Engl. J. Med.* **2006**, *355*, 1502. [CrossRef]
2. Munn, Z.; Jordan, Z. The patient experience of high technology medical imaging: A systematic review of the qualitative evidence. *Radiography* **2011**, *17*, 323–331. [CrossRef]
3. Suetens, P. *Fundamentals of Medical Imaging*, 3rd ed.; Cambridge University Press: Cambridge, UK, 2017.
4. Beyer, T.; Pichler, B. A decade of combined imaging: From a PET attached to a CT to a PET inside an MR. *Eur. J. Nucl. Med. Mol. Imaging* **2009**, *36*, 1–2. [CrossRef] [PubMed]
5. Cal-Gonzalez, J.; Rausch, I.; Sundar, L.K.S.; Lassen, M.L.; Muzik, O.; Moser, E.; Papp, L.; Beyer, T. Hybrid imaging: Instrumentation and data processing. *Front. Phys.* **2018**, *6*. [CrossRef]
6. Saini, S.; Seltzer, S.E.; Bramson, R.T.; Levine, L.A.; Kelly, P.; Jordan, P.F.; Chîango, B.F.; Thrall, J.H. Technical cost of radiologic examinations: Analysis across imaging modalities. *Radiology* **2000**, *216*, 269–272. [CrossRef]
7. Sistrom, C.L.; McKay, N.L. Costs, charges, and revenues for hospital diagnostic imaging procedures: Differences by modality and hospital characteristics. *J. Am. Coll. Radiol.* **2005**, *2*, 511–519. [CrossRef]
8. HajiRassouliha, A.; Taberner, A.J.; Nash, M.P.; Nielsen, P.M.F. Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms. *Signal Process. Image Commun.* **2018**, *68*, 101–119. [CrossRef]
9. Talib, M.A.; Majzoub, S.; Nasir, Q.; Jamal, D. A Systematic Literature Review on Hardware Implementation of Artificial Intelligence Algorithms. *J. Supercomput.* **2021**, *77*. [CrossRef]
10. Mittal, S. Vibhu A survey of accelerator architectures for 3D convolution neural networks. *J. Syst. Archit.* **2021**, *115*, 102041. [CrossRef]
11. Alzubaidi, L.; Zhang, J.; Humaidi, A.J.; Al-Dujaili, A.; Duan, Y.; Al-Shamma, O.; Santamaría, J.; Fadhel, M.A.; Al-Amidie, M.; Farhan, L. Review of Deep Learning: Concepts, CNN Architectures, Challenges, Applications, Future Directions. *J. Big Data* **2021**, *8*. [CrossRef]
12. Eklund, A.; Dufort, P.; Forsberg, D.; LaConte, S.M. Medical image processing on the GPU–Past, present and future. *Med. Image Anal.* **2013**, *17*, 1073–1094. [CrossRef]
13. Shi, L.; Liu, W.; Zhang, H.; Xie, Y.; Wang, D. A survey of GPU-based medical image computing techniques. *Quant. Imaging Med. Surg.* **2012**, *2*, 188.
14. Wang, H.; Peng, H.; Chang, Y.; Liang, D. A survey of GPU-based acceleration techniques in MRI reconstructions. *Quant. Imaging Med. Surg.* **2018**, *8*, 196–208. [CrossRef]
15. Smistad, E.; Falch, T.L.; Bozorgi, M.; Elster, A.C.; Lindseth, F. Medical image segmentation on GPUs—A comprehensive review. *Med. Image Anal.* **2015**, *20*, 1–18. [CrossRef]
16. Fluck, O.; Vetter, C.; Wein, W.; Kamen, A.; Preim, B.; Westermann, R. A survey of medical image registration on graphics hardware. *Comput. Methods Programs Biomed.* **2011**, *104*, e45–e57. [CrossRef]
17. Shams, R.; Sadeghi, P.; Kennedy, R.; Hartley, R. A survey of medical image registration on multicore and the GPU. *IEEE Signal Process. Mag.* **2010**, *27*, 50–60. [CrossRef]
18. Passaretti, D.; Joseph, J.M.; Pionteck, T. Survey on FPGAs in medical radiology applications: Challenges, architectures and programming models. In Proceedings of the 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 9–13 December 2019; pp. 279–282. [CrossRef]
19. Moses, J.; Selvathi, D. A survey on FPGA implementation of medical image registration. *Asian J. Inf. Technol.* **2014**, *13*, 46–52.
20. Ballabriga, R.; Alozy, J.; Bandi, F.N.; Campbell, M.; Egidos, N.; Fernandez-Tenllado, J.M.; Heijne, E.H.M.; Kremastiotis, I.; Llopart, X.; Madsen, B.J.; et al. Photon Counting Detectors for X-Ray Imaging with Emphasis on CT. *IEEE Trans. Radiat. Plasma Med. Sci.* **2021**, *5*, 422–440. [CrossRef]
21. Gepner, P.; Kowalik, M.F. Multi-core processors: New way to achieve high system performance. In Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), Bialystok, Poland, 13–17 September 2006; pp. 9–13.
22. Mittal, M.; Peleg, A.; Weiser, U. MMX Technology Architecture Overview. *Intel Technol. J.* **1997**, *1*. Available online: https://www.intel.com/content/dam/www/public/us/en/documents/research/1997-vol01-iss-3-intel-technology-journal.pdf (accessed on 15 October 2021).
23. Oberman, S.; Favor, G.; Weber, F. AMD 3DNow! technology: Architecture and implementations. *IEEE Micro* **1999**, *19*, 37–48. [CrossRef]
24. Lomont, C. Introduction to intel advanced vector extensions. *Intel White Pap.* **2011**, 1–21.
25. Jeffers, J.; Reinders, J.; Sodani, A. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*; Morgan Kaufmann: San Francisco, CA, USA, 2016.
26. Hennessy, J.L.; Patterson, D.A. *Cumputer Architecture, Sixth Edition: A Quantitative Approach*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2017; ISBN 0128119055.
27. Asanovic, K.; Bodik, R.; Catanzaro, B.C.; Gebis, J.J.; Husbands, P.; Keutzer, K.; Patterson, D.A.; Plishker, W.L.; Shalf, J.; Williams, S.W.; et al. *The Landscape of Parallel Computing Research: A View from Berkeley*; Technical Report UCB/EECS-2006-183; EECS Department, University of California: Berkeley, CA, USA, 2006.
28. Barlas, G. Multicore and GPU Programming. Elsevier Morgan-Kaufmann Publishers Inc.: Burlington, MA, USA, 2014.
29. Amdahl, G.M. Validity of the single processor approach to achieving large scale computing capabilities. In Proceedings of the Spring Joint Computer Conference, Atlantic City, NJ, USA, 18–20 April 1967; pp. 483–485. [CrossRef]

30. Hill, M.D.; Marty, M.R. Amdahl's Law in the Multicore Era. *Comput. Long. Beach. Calif.* **2008**, *41*, 33–38. [CrossRef]
31. Flynn, M.J. Very High-speed Computing Systems. *Proc. IEEE* **1966**, *54*, 1901–1909. [CrossRef]
32. Roy, A.; Xu, J.; Chowdhury, M.H. Multi-core processors: A new way forward and challenges. In Proceedings of the 2008 International Conference on Microelectronics, Sharjah, United Arab Emirates, 14–17 December 2008; pp. 454–457. [CrossRef]
33. IEEE. *IEEE P1003.1c/D10*; Draft Standard for Information Technology—Portable Operating Systems Interface (POSIX); The Institute of Electrical & Electronics Engineers: New York, NY, USA, 1994.
34. OpenMP Application Programming Interface, OpenMP Specification. Available online: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf (accessed on 15 October 2021).
35. Reinders, J. *Intel Threading Building BLOCKS-outfitting C++ for Multi-Core Processor Parallelism*; O'Reilly Media, Inc.: Sebastopol, CA, USA; ISBN 978-0-596-51480-8.
36. Marr, D.; Binns, F. Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technol. J.* **2002**, 1–12.
37. Membarth, R.; Hannig, F.; Teich, J.; Körner, M.; Eckert, W. Frameworks for Multi-core Architectures: A Comprehensive Evaluation Using 2D/3D Image Registration. In *Architecture of Computing Systems—ARCS 2011*; Berekovic, M., Fornaciari, W., Brinkschulte, U., Silvano, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 62–73.
38. Ekström, S.; Pilia, M.; Kullberg, J.; Ahlström, H.; Strand, R.; Malmberg, F. Faster dense deformable image registration by utilizing both CPU and GPU. *J. Med. Imaging* **2021**, *8*, 1–15. [CrossRef]
39. Kegel, P.; Schellmann, M.; Gorlatch, S. Comparing programming models for medical imaging on multi-core systems. *Concurr. Comput. Pract. Exp.* **2011**, *23*, 1051–1065. [CrossRef]
40. Kalamkar, D.D.; Trzaskoz, J.D.; Sridharan, S.; Smelyanskiy, M.; Kim, D.; Manduca, A.; Shu, Y.; Bernstein, M.A.; Kaul, B.; Dubey, P. High performance non-uniform FFT on modern X86-based multi-core systems. In Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium 2012, Shanghai, China, 21–25 May 2012; pp. 449–460. [CrossRef]
41. Saxena, S.; Sharma, N.; Sharma, S. An intelligent system for segmenting an abdominal image in multi core architecture. In Proceedings of the 2013 10th International Conference and Export on Emerging Technologies for a Smarter World (CEWIT), Melville, NY, USA, 21–22 October 2013. [CrossRef]
42. Alsmirat, M.A.; Jararweh, Y.; Al-Ayyoub, M.; Shehab, M.A.; Gupta, B.B. Accelerating compute intensive medical imaging segmentation algorithms using hybrid CPU-GPU implementations. *Multimed. Tools Appl.* **2017**, *76*, 3537–3555. [CrossRef]
43. Valsalan, P.; Sriramakrishnan, P.; Sridhar, S.; Charlyn, G.; Latha, P.; Priya, A.; Ramkumar, S.; Robert, S.A.; Rajendran, T. Knowledge based fuzzy c-means method for rapid brain tissues segmentation of magnetic resonance imaging scans with CUDA enabled GPU machine. *J. Ambient Intell. Humaniz. Comput.* **2020**, *1*, 3. [CrossRef]
44. Vaze, S.; Xie, W.; Namburete, A.I.L. Low-Memory CNNs Enabling Real-Time Ultrasound Segmentation Towards Mobile Deployment. *IEEE J. Biomed. Health Inform.* **2020**, *24*, 1059–1069. [CrossRef] [PubMed]
45. Kirk, D.B.; Wen-Mei, W.H. *Programming Massively Parallel Processors: A Hands-on Approach*; Morgan kaufmann: San Francisco, CA, USA, 2016.
46. Wilt, N. *The Cuda Handbook: A Comprehensive Guide to Gpu Programming*; CreateSpace Independent Publishing Platform: Scotts Valley, CA, USA, 2017; ISBN 9781548845162.
47. Woo, M.; Neider, J.; Davis, T.; Shreiner, D. *Opengl Programming Guide: The Official Guide To Learning Opengl*; Version 1.2; Addison-Wesley: Boston, MA, USA, 1999; ISBN 978-0-201-60458-0.
48. Microsoft. DirectX Developer Site. Available online: https://docs.microsoft.com/en-us/windows/win32/directx (accessed on 15 October 2021).
49. Owens, J.D.; Luebke, D.; Govindaraju, N.; Harris, M.; Krüger, J.; Lefohn, A.E.; Purcell, T.J. A survey of general-purpose computation on graphics hardware. *Comput. Graph. Forum* **2007**, *26*, 80–113. [CrossRef]
50. Cook, S. CUDA Programming: A Developer's Guide to Parallel Computing with GPUs. Elsevier Morgan-Kaufmann Publishers Inc.: Burlington, MA, USA, 2013; 012415334.
51. Otterness, N.; Anderson, J.H. AMD GPUs as an alternative to NVIDIA for supporting real-time workloads. In *Leibniz International Proceedings in Informatics (LIPIcs), Proceedings of the 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020), Virtual, 7– 10 July 2020*; Völp, M., Ed.; Schloss Dagstuhl–Leibniz-Zentrum für Informatik: Dagstuhl, Germany, 2020; Volume 165, pp. 10:1–10:23. [CrossRef]
52. Nvidia Corporation, Nvidia NVIDIA A100 Tensor Core GPU Architecture Unprecedented Acceleration at Every Scale (White Paper). Available online: https://images.nvidia.cn/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf (accessed on 15 October 2021).
53. Nvidia Corporation. NVIDIA®NVLink TM High-Speed Interconnect: Application Performance (White Paper). Available online: http://info.nvidianews.com/rs/nvidia/images/NVIDIA%20NVLink%20High-Speed%20Interconnect%20Application%20Performance%20Brief.pdf (accessed on 15 October 2021).
54. Jia, Z.; Maggioni, M.; Staiger, B.; Scarpazza, D.P. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. *arXiv* **2018**, arXiv:cs.DC/1804.06826.
55. Bell, N.; Garland, M. Efficient Sparse Matrix-Vector Multiplication On Cuda. Available online: https://www.nvidia.com/docs/IO/66889/nvr-2008-004.pdf (accessed on 15 October 2021).
56. Volkov, V. Understanding Latency Hiding On Gpus. Ph.D. Thesis, University of California Berkeley, Berkeley, CA, USA, 2016.

57. Bell, N.; Hoberock, J. Thrust: A Productivity-Oriented Library for CUDA. In *GPU Computing Gems JADE Edition*; Elsevier: Amsterdam, The Netherlands, 2012; pp. 359–371.
58. Merrill III, D.G. *Allocation-Oriented Algorithm Design with Application to Gpu Computing*; University of Virginia: Charlottesville, VA, USA, 2011.
59. Armyr, D.; Doherty, D. MathWorks. Prototyping Algorithms and Testing CUDA Kernels in MATLAB. Available online: https://es.mathworks.com/company/newsletters/articles/prototyping-algorithms-and-testing-cuda-kernels-in-matlab.html (accessed on 15 October 2021).
60. Munshi, A.; Gaster, B.; Mattson, T.G.; Ginsburg, D. *OpenCL Programming Guide*; Pearson Education: London, UK, 2011.
61. Welcome to AMD ROCm™ Platform. AMD Corporation. OCm Documentation. Available online: https://rocmdocs.amd.com/en/latest/ (accessed on 15 October 2021).
62. The OpenACC© Application Programming Interface. Version 3.0. Nov. 2019. Available online: https://www.openacc.org/sites/default/files/inline-images/Specification/OpenACC.3.0.pdf (accessed on 15 October 2021).
63. Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding. In Proceedings of the 22nd ACM International Conference on Multimedia, Orlando, FL, USA, 3–7 November 2014; pp. 675–678. [CrossRef]
64. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogenous Distributed Systems. *arXiv* **2016**, arXiv:cs.DC/1603.04467].
65. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 8026–8037.
66. Johnson, D.H.; Narayan, S.; Flask, C.A.; Wilson, D.L. Improved fat-water reconstruction algorithm with graphics hardware acceleration. *J. Magn. Reson. Imaging* **2010**, *31*, 457–465. [CrossRef]
67. Herraiz, J.L.; España, S.; Cabido, R.; Montemayor, A.S.; Desco, M.; Vaquero, J.J.; Udias, J.M. GPU-based fast iterative reconstruction of fully 3-D PET Sinograms. *IEEE Trans. Nucl. Sci.* **2011**, *58*, 2257–2263. [CrossRef]
68. Alcaín, E.; Torrado-Carvajal, A.; Montemayor, A.S.; Malpica, N. Real-time patch-based medical image modality propagation by GPU computing. *J. Real-Time Image Process.* **2017**, *13*, 193–204. [CrossRef]
69. Torrado-Carvajal, A.; Herraiz, J.L.; Alcain, E.; Montemayor, A.S.; Garcia-Canamaque, L.; Hernandez-Tamames, J.A.; Rozenholc, Y.; Malpica, N. Fast patch-based pseudo-CT synthesis from T1-weighted MR images for PET/MR attenuation correction in brain studies. *J. Nucl. Med.* **2016**, *57*, 136–143. [CrossRef]
70. Punithakumar, K.; Boulanger, P.; Noga, M. A GPU-Accelerated Deformable Image Registration Algorithm With Applications to Right Ventricular Segmentation. *IEEE Access* **2017**, *5*, 20374–20382. [CrossRef]
71. Florimbi, G.; Fabelo, H.; Torti, E.; Ortega, S.; Marrero-Martin, M.; Callico, G.M.; Danese, G.; Leporati, F. Towards Real-Time Computing of Intraoperative Hyperspectral Imaging for Brain Cancer Detection Using Multi-GPU Platforms. *IEEE Access* **2020**, *8*, 8485–8501. [CrossRef]
72. Torti, E.; Leon, R.; Salvia, M.L.; Florimbi, G.; Martinez-Vega, B.; Fabelo, H.; Ortega, S.; Callicó, G.M.; Leporati, F. Parallel classification pipelines for skin cancer detection exploiting hyperspectral imaging on hybrid systems. *Electronics* **2020**, *9*, 1503. [CrossRef]
73. Zachariadis, O.; Teatini, A.; Satpute, N.; Gómez-Luna, J.; Mutlu, O.; Elle, O.J.; Olivares, J. Accelerating B-spline interpolation on GPUs: Application to medical image registration. *Comput. Methods Programs Biomed.* **2020**, *193*, 105431. [CrossRef]
74. Milshteyn, E.; Guryev, G.; Torrado-Carvajal, A.; Adalsteinsson, E.; White, J.K.; Wald, L.L.; Guerin, B. Individualized SAR calculations using computer vision-based MR segmentation and a fast electromagnetic solver. *Magn. Reson. Med.* **2021**, *85*, 429–443. [CrossRef]
75. Brunn, M.; Himthani, N.; Biros, G.; Mehl, M.; Mang, A. Fast GPU 3D diffeomorphic image registration. *J. Parallel Distrib. Comput.* **2021**, *149*, 149–162. [CrossRef]
76. Martinez-Girones, P.M.; Vera-Olmos, J.; Gil-Correa, M.; Ramos, A.; Garcia-Cañamaque, L.; Izquierdo-Garcia, D.; Malpica, N.; Torrado-Carvajal, A. Franken-ct: Head and neck mr-based pseudo-ct synthesis using diverse anatomical overlapping mr-ct scans. *Appl. Sci.* **2021**, *11*, 3508. [CrossRef]
77. Frantz, G.; Brantingham, L. Signal Core: A Short History of the Digital Signal Processor. *IEEE Solid-State Circuits Mag.* **2012**, *4*, 16–20. [CrossRef]
78. Karam, L.; Alkamal, I.; Gatherer, A.; Frantz, G.; Anderson, D.; Evans, B. Trends in multicore DSP platforms. *IEEE Signal. Process. Mag.* **2009**, *26*, 38–49. [CrossRef]
79. Eyre, J.; Bier, J. The evolution of DSP processors. *IEEE Signal Processing Magazine* **2000**, *17*, 43–51. [CrossRef]
80. Frantz, G. Digital signal processor trends. *IEEE Micro* **2000**, *20*, 52–59. [CrossRef]
81. Illgner, K. DSPs for image and video processing. *Signal Process.* **2000**, *80*, 2323–2336. [CrossRef]
82. Narendra, C.P.; Kumar, K.M.R. Low power MAC architecture for DSP applications. In Proceedings of the International Conference on Circuits, Communication, Control and Computing, Bangalore, India, 21–22 November 2014; pp. 404–407. [CrossRef]
83. Simar, R. The TMS320C40: A DSP for parallel processing. In Proceedings of the ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing, Toronto, ON, Canada, 14–17 April 1991; pp. 1089–1092.
84. Fridman, J.; Greenfield, Z. The TigerSHARC DSP architecture. *IEEE Micro* **2000**, *20*, 66–76. [CrossRef]

85. Takala, J. General-Purpose DSP Processors. In *Handbook of Signal Processing Systems*; Bhattacharyya, S., Deprettere, E., Leupers, R., Takala, J., Eds.; Springer: Boston, MA, USA, 2010. [CrossRef]
86. Patterson, J.; Dixon, J. Optimizing Power Consumption in DSP Designs (White Paper). Available online: https://www.ti.com/lit/wp/spry089/spry089.pdf?ts=1639187652517&ref_url=https%253A%252F%252Fwww.google.com.hk%252F (accessed on 15 October 2021).
87. Rao, A.; Nagori, S. Optimizing Video Encoders with TI DSPs (White Paper). Available online: https://www.ti.com/lit/wp/spry106/spry106.pdf?ts=1639187749440&ref_url=https%253A%252F%252Fwww.google.com.hk%252F (accessed on 15 October 2021).
88. Bergsagel, J.; Leconte, L. Super high resolution displays empowered by the OMAP4470 mobile processor (White paper). Available online: https://www.ti.com/lit/wp/swpy028/swpy028.pdf (accessed on 15 October 2021).
89. Chaoui, J.; Cyr, K.; De Gregorio, S.; Giacalone, J.P.; Webb, J.; Masse, Y. Open multimedia application platform: Enabling multimedia applications in third generation wireless terminals through a combined Risc/Dsp architecture. In Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, Salt Lake City, UT, USA, 7–11 May 2001; Volume 2, pp. 1009–1012. [CrossRef]
90. Bahri, N.; Belhadj, N.; Ben Ayed, M.A.; Masmoudi, N.; Grandpierre, T.; Akil, M. Real-time H264/AVC high definition video encoder on a multicore DSP TMS320C6678. In Proceedings of the International Conference on Computer Vision and Image Analysis Applications, Sousse, Tunisia, 18–20 January 2015; pp. 1–6.
91. Igual, F.D.; Ali, M.; Friedmann, A.; Stotzer, E.; Wentz, T.; van de Geijn, R.A. Unleashing the high-performance and low-power of multi-core DSPs for general-purpose HPC. In Proceedings of the SC'12 International Conference on High Performance Computing, Networking, Storage and Analysis, Salt Lake City, UT, USA, 25 February 2012; pp. 1–11.
92. Rao, M.V.G.; Kumar, P.R.; Prasad, A.M. Implementation of real time image processing system with FPGA and DSP. In Proceedings of the 2016 International Conference on Microelectronics, Computing and Communications (MicroCom), Durgapur, India, 23–25 January 2016; pp. 4–7. [CrossRef]
93. NXP Semiconductors. Crossover Embedded Processors-Bridging the Gap between Performance and Usability (White Paper). Available online: https://www.nxp.com/docs/en/white-paper/I.MXRT1050WP.pdf (accessed on 15 October 2021).
94. Zhong, M.; Yang, Y.; Zhou, Y.; Octavian, P.M.; Chandrasekar, M.; Venkat, B.G.; Manikandan, C.; Balaji, V.S.; Saravanan, S.; Elamaran, V. Advanced Digital Signal Processing Techniques on the Classification of the Heart Sound Signals. *J. Med. Imaging Heal. Inform.* **2020**, *10*, 2010–2015. [CrossRef]
95. Berg, R.; König, L.; Rühaak, J.; Lausen, R.; Fischer, B. Highly efficient image registration for embedded systems using a distributed multicore DSP architecture. *J. Real-Time Image Process.* **2018**, *14*, 341–361. [CrossRef]
96. Chandrashekar, L.; Sreedevi, A. A Multi-Objective Enhancement Technique for Poor Contrast Magnetic Resonance Images of Brain Glioblastomas. *Procedia Comput. Sci.* **2020**, *171*, 1770–1779. [CrossRef]
97. Ueng, S.K.; Yen, C.L.; Chen, G.Z. Ultrasound image enhancement using structure-based filtering. *Comput. Math. Methods Med.* **2014**, 1–14. [CrossRef]
98. Ali, M.; Magee, D.; Dasgupta, U. Signal Processing Overview of Ultrasound Systems for Medical Imaging. Available online: https://www.k-space.org/ymk/sprab12.pdf (accessed on 15 October 2021).
99. Pailoor, R.; Pradhan, D. Digital signal processor (DSP) for portable ultrasound. *Tex. Instrum. Appl. Rep. SPRAB18A* **2008**, 1–11.
100. Boni, E.; Bassi, L.; Dallai, A.; Giannini, G.; Guidi, F.; Meacci, V.; Matera, R.; Ramalli, A.; Ricci, S.; Scaringella, M.; et al. ULA-OP 256: A portable high-performance research scanner. In Proceedings of the 2015 IEEE International Ultrasonics Symposium (IUS), Taipei, Taiwan, 21–24 October 2015; pp. 1–4.
101. Ricci, S.; Ramalli, A.; Bassi, L.; Boni, E.; Tortoli, P. Real-time blood velocity vector measurement over a 2-D region. *IEEE Trans. Ultrason. Ferroelectr. Freq. Control.* **2017**, *65*, 201–209. [CrossRef] [PubMed]
102. Ramalli, A.; Guidi, F.; Dallai, A.; Boni, E.; Tong, L.; D'hooge, J.; Tortoli, P. High frame rate, wide-angle tissue doppler imaging in real-time. In Proceedings of the 2017 IEEE International Ultrasonics Symposium (IUS), Washington, DC, USA, 6–9 September 2017; pp. 1–4.
103. Tortoli, P.; Lenge, M.; Righi, D.; Ciuti, G.; Liebgott, H.; Ricci, S. Comparison of carotid artery blood velocity measurements by vector and standard Doppler approaches. *Ultrasound Med. Biol.* **2015**, *41*, 1354–1362. [CrossRef]
104. Arulkumar, M.; Chandrasekaran, M. An Improved VLSI Design of the ALU Based FIR Filter for Biomedical Image Filtering Application. *Curr. Med. Imaging* **2021**, *17*, 276–287. [CrossRef]
105. Beddad, B.; Hachemi, K. Application of brain tumor detection on DSP environment using TMS320C6713 DSK. In Proceedings of the 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, Spain, 5–7 July 2017; pp. 599–603.
106. A Brief History of FPGA | Make. Available online: https://makezine.com/2019/10/11/a-brief-history-of-fpga/ (accessed on 15 October 2021).
107. Farooq, U.; Marrakchi, Z.; Mehrez, H. *Tree-Based Heterogeneous FPGA Architectures: Application Specific Exploration and Optimization*; Springer: New York, NY, USA, 2012; ISBN 9781461435945.
108. Anusudha, K.; Naguboina, G.C. Design and implementation of PAL and PLA using reversible logic on FPGA SPARTAN 3E. In Proceedings of the 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, Spain, 5–7 July 2017; pp. 1–6. [CrossRef]
109. Grout, I. *Digital Systems Design with FPGAs and CPLDs*; Newnes: Oxford, UK, 2008; ISBN 9780080558509.

110. Kuon, I.; Rose, J. Measuring the Gap between FPGAs and ASICs. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2007**, *26*, 203–215. [CrossRef]
111. Trimberger, S.M.S. Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology: This Paper Reflects on How Moore's Law Has Driven the Design of FPGAs Through Three Epochs: The Age of Invention, the Age of Expansion, and the Age of Accumulation. *IEEE Solid-State Circuits Mag.* **2018**, *10*, 16–29. [CrossRef]
112. Wolf, W. *FPGA-Based System Design*; Prentice Hall PTR Pearson Education: Upper Saddle River, NJ, USA, 2004; ISBN 978-0137033485.
113. Ahmad, S.; Boppana, V.; Ganusov, I.; Kathail, V.; Rajagopalan, V.; Wittig, R. A 16-nm multiprocessing system-on-chip field-programmable gate array platform. *IEEE Micro* **2016**, *36*, 48–62. [CrossRef]
114. Ahmad, S.; Subramanian, S.; Boppana, V.; Lakka, S.; Ho, F.-H.; Knopp, T.; Noguera, J.; Singh, G.; Wittig, R. Xilinx first 7 nm device: Versal AI core (VC1902). In Proceedings of the 2019 IEEE Hot Chips 31 Symposium (HCS), Cupertino, CA, USA, 18–20 August 2019; pp. 1–28.
115. Voogel, M.; Frans, Y.; Ouellette, M.; Coppens, J.; Ahmad, S.; Dastidar, J.; Mohsen, E.; Dada, F.; Thompson, M.; Wittig, R.; et al. Xilinx Versal™ Premium. In Proceedings of the 2020 IEEE Hot Chips 32 Symposium (HCS), Palo Alto, CA, USA, 16–18 August 2020; pp. 1–46.
116. Xilinx, Inc. Versal, the First Adaptive Compute Acceleration Platform (ACAP) (White Paper). Available online: https://www.xilinx.com/support/documentation/white_papers/wp505-versal-acap.pdf (accessed on 15 October 2021).
117. Gaide, B.; Gaitonde, D.; Ravishankar, C.; Bauer, T. *Xilinx Adaptive Compute Acceleration Platform: Versal™ Architecture. Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*; Association for Computing Machinery: New York, NY, USA, 2019; FPGA '19; pp. 84–93. [CrossRef]
118. Dillinger, P.; Vogelbruch, J.F.; Leinen, J.; Suslov, S.; Patzak, R.; Winkler, H.; Schwan, K. FPGA Based Real-Time Image Segmentation for Medical Systems and Data Processing. *IEEE Trans. Nuclear Sci.* **2006**, *53*. [CrossRef]
119. Jiang, R.M.; Crookes, D. FPGA implementation of 3D discrete wavelet transform for real-time medical imaging. In Proceedings of the 2007 18th European Conference on Circuit Theory and Design, Seville, Spain, 27–30 August 2007; pp. 519–522.
120. Korcyl, G.; Bialas, P.; Curceanu, C.; Czerwinski, E.; Dulski, K.; Flak, B.; Gajos, A.; Glowacz, B.; Gorgol, M.; Hiesmayr, B.C.; et al. Evaluation of Single-Chip, Real-Time Tomographic Data Processing on FPGA SoC Devices. *IEEE Trans. Med. Imaging* **2018**, *37*, 2526–2535. [CrossRef] [PubMed]
121. Gebhardt, P.; Weissler, B.; Zinke, M.; Kiessling, F.; Marsden, P.K.; Schulz, V. FPGA-based singles and coincidences processing pipeline for integrated digital PET/MR detectors. In Proceedings of the 2012 IEEE Nuclear Science Symposium and Medical Imaging Conference Record (NSS/MIC), Anaheim, CA, USA, 27 October–3 November 2012; pp. 2479–2482. [CrossRef]
122. Marjanovic, J.; Reber, J.; Brunner, D.O.; Engel, M.; Kasper, L.; Dietrich, B.E.; Vionnet, L.; Pruessmann, K.P. A Reconfigurable Platform for Magnetic Resonance Data Acquisition and Processing. *IEEE Trans. Med. Imaging* **2020**, *39*, 1138–1148. [CrossRef] [PubMed]
123. Kim, G.D.; Yoon, C.; Kye, S.B.; Lee, Y.; Kang, J.; Yoo, Y.; Song, T.K. A single FPGA-based portable ultrasound imaging system for point-of-care applications. *IEEE Trans. Ultrason. Ferroelectr. Freq. Control* **2012**, *59*, 1386–1394. [CrossRef]
124. Zhang, Z.; Xin, Y.; Liu, B.; Li, W.X.Y.; Lee, K.H.; Ng, C.F.; Stoyanov, D.; Cheung, R.C.C.; Kwok, K.W. FPGA-based high-performance collision detection: An enabling technique for image-guided robotic surgery. *Front. Robot. AI* **2016**, *3*, 1–14. [CrossRef]
125. Mittal, S.; Gupta, S.; Dasgupta, S. FPGA: An efficient and promising platform for real-time image processing applications. In Proceedings of the National Conference On Research and Development In Hardware Systems (CSI-RDHS), Kolkata, India, 20–21 June 2008.
126. Mondal, P.; Biswal, P.K.; Banerjee, S. FPGA based accelerated 3D affine transform for real-time image processing applications. *Comput. Electr. Eng.* **2016**, *49*, 69–83. [CrossRef]
127. Conficconi, D.; D'Arnese, E.; Del Sozzo, E.; Sciuto, D.; Santambrogio, M.D. A framework for customizable Fpga-based image registration accelerators. In Proceedings of the FPGA'21: The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Virtual Event, USA, 28 February–2 March 2021; pp. 251–261. [CrossRef]
128. Mondal, P.; Banerjee, S. FPGA-accelerated adaptive projection-based image registration. *J. Real-Time Image Process.* **2021**, *18*, 113–125. [CrossRef]
129. Gebhardt, P.; Wehner, J.; Weissler, B.; Botnar, R.; Marsden, P.K.; Schulz, V. FPGA-based RF interference reduction techniques for simultaneous PET-MRI. *Phys. Med. Biol.* **2016**, *61*, 3500–3526. [CrossRef]
130. Zhou, H.; Machupalli, R.; Mandal, M. Efficient FPGA implementation of automatic nuclei detection in histopathology images. *J. Imaging* **2019**, *5*, 21. [CrossRef]
131. Othman, M.F.B.; Abdullah, N.; Rusli, N.A.B.A. An overview of MRI brain classification using FPGA implementation. In Proceedings of the 2010 IEEE Symposium on Industrial Electronics and Applications (ISIEA), Penang, Malaysia, 3–5 October 2010; pp. 623–628. [CrossRef]
132. Sanaullah, A.; Yang, C.; Alexeev, Y.; Yoshii, K.; Herbordt, M.C. Real-time data analysis for medical diagnosis using FPGA-accelerated neural networks. *BMC Bioinform.* **2018**, *19*, 19–31. [CrossRef]
133. Li, L.; Wyrwicz, A.M. Design of an MR image processing module on an FPGA chip. *J. Magn. Reson.* **2015**, *255*, 51–58. [CrossRef]
134. Siddiqui, M.F.; Reza, A.W.; Shafique, A.; Omer, H.; Kanesan, J. FPGA implementation of real-time SENSE reconstruction using pre-scan and Emaps sensitivities. *Magn. Reson. Imaging* **2017**, *44*, 82–91. [CrossRef]

135. Inam, O.; Basit, A.; Qureshi, M.; Omer, H. FPGA-based hardware accelerator for SENSE (a parallel MR image reconstruction method). *Comput. Biol. Med.* **2020**, *117*, 103598. [CrossRef] [PubMed]
136. Liang, X.; Binghe, S.; Yueping, M.; Ruyan, Z. A digital magnetic resonance imaging spectrometer using digital signal processor and field programmable gate array. *Rev. Sci. Instrum.* **2013**, *84*, 054702. [CrossRef] [PubMed]
137. Choi, Y.; Cong, J. Acceleration of EM-Based 3D CT Reconstruction Using FPGA. *IEEE Trans. Biomed. Circuits Syst.* **2016**, *10*, 754–767. [CrossRef]
138. Windisch, D.; Knodel, O.; Juckeland, G.; Hampel, U.; Bieberle, A. FPGA-based Real-Time Data Acquisition for Ultrafast X-Ray Computed Tomography. *IEEE Trans. Nucl. Sci.* **2021**. [CrossRef]
139. Goel, G.; Gondhalekar, A.; Qi, J.; Zhang, Z.; Cao, G.; Feng, W. ComputeCOVID19+: Accelerating COVID-19 diagnosis and monitoring via high-performance deep Learning on CT images. In Proceedings of the 50th International Conference on Parallel Processing; Association for Computing Machinery, Lemont, IL, USA, 9–12 August 2021.
140. Gong, K.; Berg, E.; Cherry, S.R.; Qi, J. Machine Learning in PET: From Photon Detection to Quantitative Image Reconstruction. *Proc. IEEE* **2020**, *108*, 51–68. [CrossRef]
141. Won, J.Y.; Lee, J.S. Highly Integrated FPGA-Only Signal Digitization Method Using Single-Ended Memory Interface Input Receivers for Time-of-Flight PET Detectors. *IEEE Trans. Biomed. Circuits Syst.* **2018**, *12*, 1401–1409. [CrossRef]
142. Won, J.Y.; Park, H.; Lee, S.; Son, J.-W.; Chung, Y.; Ko, G.B.; Kim, K.Y.; Song, J.; Seo, S.; Ryu, Y.; et al. Development and Initial Results of a Brain PET Insert for Simultaneous 7-Tesla PET/MRI Using an FPGA-Only Signal Digitization Method. *IEEE Trans. Med. Imaging* **2021**, *40*, 1579–1590. [CrossRef]
143. Assef, A.A.; Ferreira, B.M.; Maia, J.M.; Costa, E.T. Modeling and FPGA-based implementation of an efficient and simple envelope detector using a Hilbert Transform FIR filter for ultrasound imaging applications. *Res. Biomed. Eng.* **2018**, *34*, 87–92. [CrossRef]
144. Wu, X.; Sanders, J.L.; Zhang, X.; Yamaner, F.Y.; Oralkan, Ö. An FPGA-Based Backend System for Intravascular Photoacoustic and Ultrasound Imaging. *IEEE Trans. Ultrason. Ferroelectr. Freq. Control* **2019**, *66*, 45–56. [CrossRef] [PubMed]
145. Nasser, Y.; Lorandel, J.; Prévotet, J.-C.; Hélard, M. RTL to Transistor Level Power Modeling and Estimation Techniques for FPGA and ASIC: A Survey. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2020**, *40*, 479–493. [CrossRef]
146. Weste, N.H.E.; Harris, D. *CMOS VLSI Design: A Circuits and Systems Perspective*; Pearson Education India: Upper Saddle River, NJ, USA, 2015.
147. Rabaey, J.M.; Chandrakasan, A.P.; Nikolić, B. *Digital Integrated Circuits: A Design Perspective*; Pearson education: Upper Saddle River, NJ, USA, 2003; Volume 7.
148. Zahiri, B. Structured ASICs: Opportunities and challenges. In Proceedings of the 21st International Conference on Computer Design, San Jose, CA, USA, 13–15 October 2003; pp. 404–409. [CrossRef]
149. Nekoogar, F. *From ASICs to SOCs: A Practical Approach*; Prentice Hall Professional: Hoboken, NJ, USA, 2003.
150. Kmon, P.; Szczygiel, R.; Maj, P.; Grybos, P.; Kleczek, R. High-speed readout solution for single-photon counting ASICs. *J. Instrum.* **2016**, *11*, C02057. [CrossRef]
151. Sundberg, C.; Persson, M.U.; Sjölin, M.; Wikner, J.J.; Danielsson, M. Silicon photon-counting detector for full-field CT using an ASIC with adjustable shaping time. *J. Med. Imaging* **2020**, *7*, 53503. [CrossRef] [PubMed]
152. Xu, C.; Persson, M.; Chen, H.; Karlsson, S.; Danielsson, M.; Svensson, C.; Bornefalk, H. Evaluation of a second-generation ultra-fast energy-resolved ASIC for photon-counting spectral CT. *IEEE Trans. Nucl. Sci.* **2013**, *60*, 437–445. [CrossRef]
153. Zannoni, E.M.; Wilson, M.D.; Bolz, K.; Goede, M.; Lauba, F.; Schöne, D.; Zhang, J.; Veale, M.C.; Verhoeven, M.; Meng, L.J. Development of a multi-detector readout circuitry for ultrahigh energy resolution single-photon imaging applications. *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrometers Detect. Assoc. Equip.* **2020**, *981*, 164531. [CrossRef]
154. Jones, L.; Seller, P.; Wilson, M.; Hardie, A. HEXITEC ASIC-a pixellated readout chip for CZT detectors. *Nucl. Instrum. Methods Phys. Res. Sect. A Accel. Spectrometers Detect. Assoc. Equip.* **2009**, *604*, 34–37. [CrossRef]
155. Pushkar; Junnarkar, S. Development of high resolution modular four side buttable small field of view detectors for three dimensional gamma imaging. *J. Nucl. Med.* **2021**, *62* (Suppl. 1), 3038.
156. Kang, J.; Yoon, C.; Lee, J.; Kye, S.B.; Lee, Y.; Chang, J.H.; Kim, G.D.; Yoo, Y.; Song, T.K. A System-on-Chip Solution for Point-of-Care Ultrasound Imaging Systems: Architecture and ASIC Implementation. *IEEE Trans. Biomed. Circuits Syst.* **2016**, *10*, 412–423. [CrossRef]
157. Kim, T.; Fool, F.; Dos Santos, D.S.; Chang, Z.-Y.; Noothout, E.; Vos, H.J.; Bosch, J.G.; Verweij, M.D.; de Jong, N.; Pertijs, M.A.P. Design of an ultrasound transceiver asic with a switching-artifact reduction technique for 3D carotid artery imaging. *Sensors* **2021**, *21*, 150. [CrossRef]
158. Rothberg, J.M.; Ralston, T.S.; Rothberg, A.G.; Martin, J.; Zahorian, J.S.; Alie, S.A.; Sanchez, N.J.; Chen, K.; Chen, C.; Thiele, K.; et al. Ultrasound-on-chip platform for medical imaging, analysis, and collective intelligence. *Proc. Natl. Acad. Sci. USA* **2021**, *118*, 1–9. [CrossRef] [PubMed]
159. Sheng, D.; Liu, C.-H.; Chen, S.-Y.; Song, B.-Y.; Chiu, Y.-C.; Cai, M.-H. DLL-based transmit beamforming IC for high-frequency ultrasound medical imaging system. In Proceedings of the 2021 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Penghu, Taiwan, China; 2021; pp. 1–2.
160. Cela, J.M.; Freixas, L.; Lagares, J.I.; Marin, J.; Martinez, G.; Navarrete, J.; Oller, J.C.; Perez, J.M.; Rato-Mendes, P.; Sarasola, I.; et al. A Compact Detector Module Design Based on FlexToT ASICs for Time-of-Flight PET-MR. *IEEE Trans. Radiat. Plasma Med. Sci.* **2018**, *2*, 549–553. [CrossRef]

161. Sánchez, D.; Gómez, S.; Mauricio, J.; Freixas, L.; Sanuy, A.; Guixé, G.; López, A.; Manera, R.; Marin, J.; Pérez, J.M.; et al. HRFlexToT: A High Dynamic Range ASIC for Time-of-Flight Positron Emission Tomography. *IEEE Trans. Radiat. Plasma Med. Sci.* **2021**. [CrossRef]
162. Nemallapudi, M.V.; Rahman, A.; Chen, A.E.-F.; Lee, S.-C.; Lin, C.-H.; Chu, M.-L.; Chou, C.-Y. Positron emitter depth distribution in PMMA irradiated with 130 MeV protons measured using TOF-PET detectors. *IEEE Trans. Radiat. Plasma Med. Sci.* **2021**, *7311*, 1. [CrossRef]
163. Shen, W.; Harion, T.; Schultz-Coulon, H.C. STiC an ASIC CHIP for silicon-photomultiplier fast timing discrimination. In Proceedings of the IEEE Nuclear Science Symposuim & Medical Imaging Conference, Knoxville, TN, USA, 30 October–6 November 2010; pp. 406–408. [CrossRef]
164. Gandhare, S.; Karthikeyan, B. Survey on FPGA architecture and recent applications. In Proceedings of the 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN), Vellore, India, 30–31 March 2019; pp. 1–4.
165. Romoth, J.; Porrmann, M.; Rückert, U. Survey of FPGA applications in the period 2000–2015 (Tech. Rep.). Center of Excellence Cognitive Interaction Technology, Bielefeld University: Bielefeld, Germany, 2017; pp. 1–43.
166. Wong, H.S.P.; Akarvardar, K.; Antoniadis, D.; Bokor, J.; Hu, C.; King-Liu, T.J.; Mitra, S.; Plummer, J.D.; Salahuddin, S. A Density Metric for Semiconductor Technology [Point of View]. *Proc. IEEE* **2020**, *108*, 478–482. [CrossRef]
167. Al-Ali, F.; Gamage, T.D.; Nanayakkara, H.W.T.S.; Mehdipour, F.; Ray, S.K. Novel casestudy and benchmarking of AlexNet for edge AI: From CPU and GPU to FPGA. In Proceedings of the 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), London, ON, Canada, 30 August–2 September 2020; pp. 1–4.
168. Xiong, C.; Xu, N. Performance comparison of BLAS on CPU, GPU and FPGA. In Proceedings of the 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 11–13 December 2020; Volume 9, pp. 193–197.
169. Asano, S.; Maruyama, T.; Yamaguchi, Y. Performance comparison of FPGA, GPU and CPU in image processing. In Proceedings of the 2009 International Conference on Field Programmable Logic and Applications, Prague, Czech Republic, 31 August–2 September 2009; pp. 126–131. [CrossRef]