Field-programmable-gate-arrays within physics and their applications

Student number: 2325410

2325410D@student.glasgow.ac.uk

University of Glasgow

Abstract. Algorithms are evolving continuously in both complexity and size. The processing demands of such algorithms must grow as well. Sequential methods, those implemented on central processing units (CPUs), provided a scalable method for a considerable time before power limitations became a major limiting factor. To meet the demand for increased processing capabilities and still be power efficient many have turned their attention to field-programmable-gate arrays (FPGAs). Chips such as these offer true parallel processing capabilities yielding a lesser total power draw compared to modern sequential solutions. This review illustrates some of the applications FPGAs are utilised in and provides a comprehensive outline of standard FPGA architecture modules. This review concluded that FPGA solutions present significant performance increases within big data encryption, machine learning algorithms, and measuring systems compared to modern CPU implementations. Any algorithm that can be parallelised can reap the benefits of FPGAs affinity for parallel processing while also remaining considerably more power-efficient. As such, systems implementing FPGAs were concluded to be more responsive, contend with ever-increasing amounts of data, and offer appropriate scaling compared to conventional sequential methods.

1. Introduction

With the improvement of modern consumer-grade processors, it is evident that a new way of processing data is emerging. Most consumer-grade processors offered on the market now allow for multi-threaded processing. Effectively, CPUs have evolved not just to contain the one processing core but multiple. This shift enables processors to process data in parallel to a degree but with a more significant power draw.

In section 2, a review of the design and origins of FPGAs is conducted. A comprehensive look is taken at the central processing components that implement the logic on the chip.

Moving ahead, section 3 illustrates the main limitation of parallelism and analyses the effect Amdahl's Law has on a system.

Section 4 introduces the first of 3 main areas FPGAs are being utilized in, hardware accelerating machine learning. This section discusses the theory of modern machine learning algorithms, displays the main requirements for optimal processing, and highlights the need for a scalable architecture to accommodate ever-growing complex algorithms. Additionally, a

practical application of machine learning acceleration through the implementation of FPGAs is also illustrated. Following on from section 4, 5 presents the improvements made to measuring systems via FPGA implementation. This section describes the advances made in measuring latency as FPGAs naturally allow a high data throughput. Another final area where FPGAs are being utilized is big data processing, specifically with encryption of large data sets; this is illustrated throughout section 6.

Finally, section 7 presents a discussion on the three areas mentioned earlier, with the article concluding in section 8.

2. Origin and design of the FPGA

The initial production of FPGAs began in the mid-1980s by two prominent rival companies. Altera, acquired by Intel in late 2015, developed the first programmable logic device in 1984, one year after the company was first formed [1]. This chip was not commercially available. Xilinx, founded in 1984 and now owned by AMD, created the first consumer-grade FPGA in 1985 [2].

FPGAs originated from two primary programmable logic devices; a programmable read-only memory (PROM) and programmable logic devices (PLDs). PROM is a type of non-volatile memory where information may be loaded and either programmed by a user or programmed en masse in a factory. The PROM type programmable by the user came in two different forms, erasable (EPROM) and electronic erasable (EEPROM). PLDs came in various types; however, the most abundant used a combination of fixed logical OR gates with a programmable array of AND gates. The FPGA is an adaptation of a PLD, one with programmable interconnects rather than fixed wires heading to the gates [3].

2.1. Modern FPGA Architecture

FPGA architecture has evolved substantially from the original designs. In 1987 the number of equivalent logic gates per chip was 9000; modern designs grew from this, amounting to 50 million gates in 2013. In other words, logic density had increased. In addition to an increase in logic density came the ability to integrate more speciality blocks. A typical architecture consists of; configurable logic blocks (CLBs), input/output blocks (IOBs), and a collection of programmable interconnections as standard. Figure 1 illustrates a simplified view of FPGA architecture.

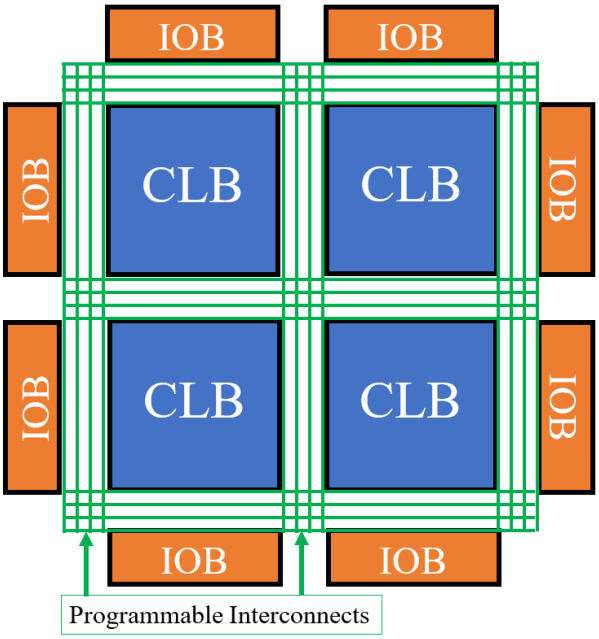

Figure 1: Diagram of modern FPGA architecture heavily simplified. FPGAs have thousands of CLBs to simulate more complex logic functions with substantially more IOBs. Figure adapted from [4]

Modern FPGA architecture can include a wide variety of different additional blocks, each with its own functionality. These blocks range from digital signal processing (DSPs) to internal memory blocks to peripheral component interconnect express interfaces (PCIe). The single most crucial element of these is the configurable logic block.

2.2 The Configurable Logic Block

These blocks are composed of three main subcomponents, a flip-flop, a look-up table (LUT), and a multiplexer.

2.2.1. *The Look-up table* A look-up table is used to manage inputs from the programmable interconnects. Programmable LUTs are the backbone of the logic functionality of the CLB. Typically, within FPGAs, LUTs manage from between 4 to 6 input functions at a time, commonly denoted 4-LUTs or 6-LUTs; modern FPGAs typically include the latter type [5]. A LUT is effectively an extensive, predetermined list of outputs for a combination of any given inputs. Given an input quantity (n), a LUT consists of LUT mask cells (M) of up to 2^n for storing the truth table values of the given inputs.

$$M = 2^n \tag{1}$$

For modern FPGAs, this yields between 16 to 64 possible addresses for storing and looking up truth values hence the term look-up table. A LUT is extremely fast for determining the value of an input function regardless of the complexity because all the outputs are readily

accessible; only one memory look-up is needed. For larger input functions, the table can either be expanded or routed to another LUT. This ability to process complex functions extremely quickly is essential to how FPGAs have unmatched processing speed compared to traditional sequential processing methods. Figure 2 shows an expanded view of a 4-LUT, a look-up table with four possible inputs, including 16 memory cells in the LUT-mask. The LUT then outputs the state to the flip-flop.

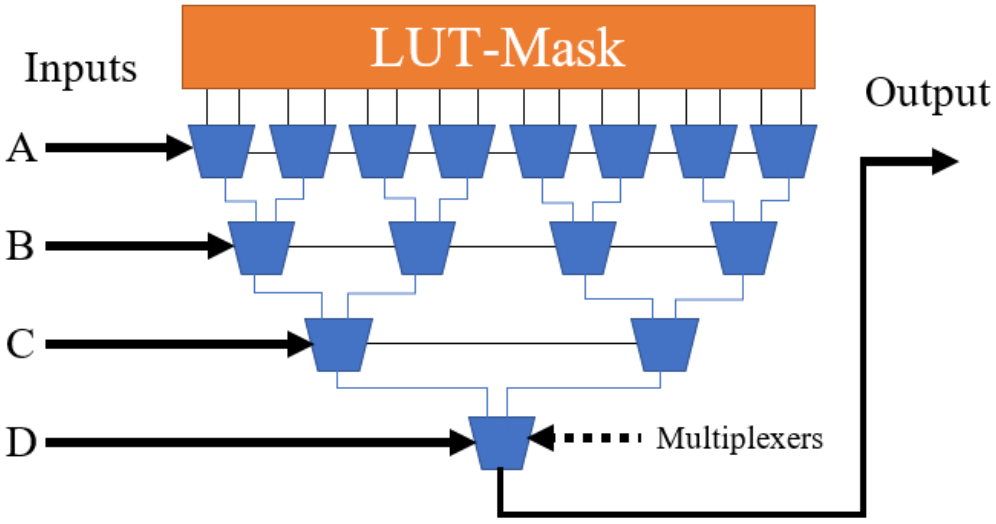

Figure 2: An expanded version of a 4-LUT. Here multiplexers are used to navigate to the LUT-Mask and drive output. Model reconstructed from [6,7].

2.2.2. *The flip-flop.* is a device that is fundamental to all digital electronics. A flip-flop is effectively the simplest form of storage and allows state information storage, each state representing a bit of information. It can store a single bit of information, either a 1 or 0, which is indicative of the state of the circuit. In CLBs and throughout digital electronic circuits, flip-flops are used as a register to store logic states between clock cycles.

Finally, a multiplexer is used to select an input from a selection of inputs and outputs. This selection is then output to the rest of the chip; the multiplexer is used as selection logic. Figure 3 provides a simplified view of the internals of CLBs within FPGAs

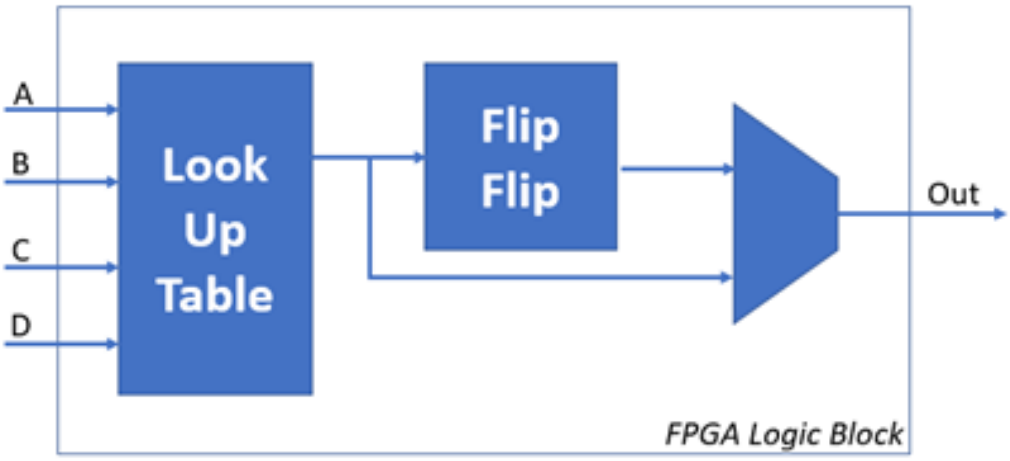

Figure 3: An abstracted view of the layout of a CLB within an FPGA. Typical FPGA designs include multiple LUTs and flip-flops within their CLBs. Simplistic designs, designs containing one or two LUTs, are commonly referred to as slices. Figure sourced from [8]

Thousands of these configurable logic blocks allow the FPGA to execute complex logical functions and effectively behave as any other digital circuit. FPGAs may be used to simulate a simple logic gate or an entire multi-threaded processor. One additional benefit of the FPGA architecture is that it allows for parallel processing of inputs. An application-specific integrated circuit (ASIC) is effectively a non-reconfigurable FPGA chip. Often FPGA chip designs are used to formulate the design for an ASIC.

2.3 Parallel Processing and Frequency Scaling

Modern computers rely on a sequential processing scheme. Computer programs get translated from code to assembly language by a compiler, which then gets translated into microcode on the system processor. These steps happen in a strict order and follow a sequence hence the name sequential processing. Parallel processing, more commonly referred to as parallel computing, has a more detailed workflow. A program is instead distributed among processor cores, and the result is computed simultaneously. This is extremely useful for processing large problems that would take sequential methods much longer to process. More extensive problems are effectively abstracted to smaller independent tasks that each processor core or individual computer can tackle. The parallel computational approach has seen considerable interest within the area of high-performance computing, where the advent of frequency scaling limited computing performance gains.

During the first stages of FPGA production, frequency scaling was the main limiting factor affecting computing performance. Decreasing the runtime of a program was the quickest and most effective way of increasing compute performance. To reduce the runtime, assuming the number of instructions processed by the CPU stays constant, manufacturers increased the clock frequency of CPUs. This increased clock frequency reduced runtimes for CPU-limited programs by decreasing the average time an instruction took to process. Increasing clock frequency however, yielded an increase in power draw by the CPU according to:

$$P = CV^2 f$$

(2)

The power draw (*P*) of the processor is a product of the capacitance (*C*), switched every clock cycle and tied to the number of transistors on the chip that have changed inputs. Maintaining a constant voltage (*V*) across the chip while increasing the clock frequency *(f)* yields an ever-increasing power draw [9]. Increasing power draws on CPUs lead to them overheating regularly, requiring a different approach. This eventually led to the end of frequency scaling as the primary method of performance gains for chip manufacturers [10]. Manufactures, instead of scaling frequency, opted for more efficient processing methods utilizing multiple cores. Multi-core processors introduced a form of parallel computing to the modern computer allowing parallelization of sequential programs. Parallelizing serial programs have now become the norm to leverage the use of multi-core architectures, and as such, FPGAs have gained a lot of attention due to their reconfigurable architecture. The adaption to parallel computing does provide significant performance gains, but this does still have some limitations.

3. Limitations on parallelism

Parallel computing effectively speeds up the time it takes to process and return the output of a serial program. The potential speed gains in terms of an algorithm being process in parallel are illustrated by Amdahl's Law [11,12]:

$$S = \frac{1}{1 - p + \frac{p}{s}}$$

(3)

Not all tasks can be parallelisable; most tasks instead consist of parallel and sequential sections. The potential increase in processing speed in terms of latency (*S*) is given as a function of processing speed increase, attributed to the parallelisable part of the task (*s*) and the percentage of the program runtime dedicated to the parallelised part of the task before any abstraction takes place (*p*). It can be inferred from this that even a small amount of a task that is non-parallelisable will impede the potential speed gains utilising the parallelisation method. Assuming that a considerable portion of a program is parallelisable, this leaves a section that will have to be sequentially processed. The part processed in parallel will undoubtedly decrease runtime performance to a certain degree. The amount which remains sequentially processed will act as an effective bottleneck to the system, invariant to how many processor cores are present. The runtime of the program will forever be dependent on the part, which is sequentially processed. Figure 4 illustrates Amdahl's Law (3) and the redundancy of increasing processor count to decrease runtime.

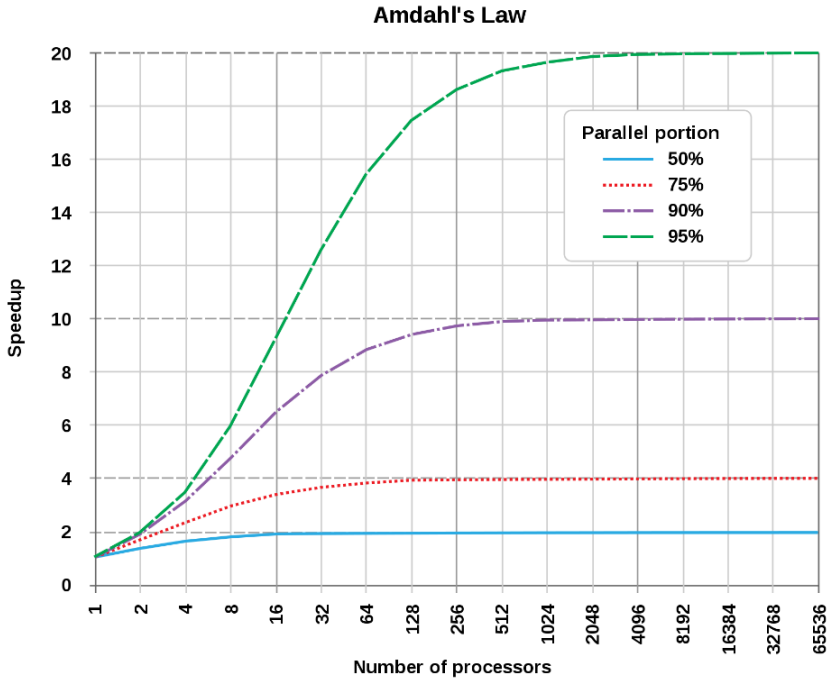

Figure 4: Plot of Amdahl's Law with varying degrees of parallelisation of a task. It can be seen that the greater percentage of the task that is parallelised, the more significant the gain in speed. Increasing processor count does not affect increasing the rate at which the program is eventually processed; the plateau that each plot reaches illustrates this. Figure sourced from [13].

This law, however, does not account for a problem evolving in complexity. A typical example of a computational problem growing would be mapping a tropical storm. For problems that do develop, access to more processing chips still attribute to speeding up computational time. Since their inception, FPGAs have existed as a cost-effective form of hardware acceleration for proper parallel computation.

4. Hardware accelerating machine learning Origin and design of the FPGA

Machine learning has become abundant in many research fields. Deep learning, a specific branch of machine learning methods, has assisted to rapidly develop newer forms of artificial intelligence. Most deep learning models follow a similar structure and consist of multiple arrays of neurons stacked in processing layers. The most predominant of these networks are convolutional neural networks (CNN) [14]. CNNs consist of 3 types of layers, a convolutional, pooling, and fully connected layer. Essentially these models are self-adapting and continuously improve with time provided they are trained with sufficient data. A more precise CNN will likely have a multitude of layers to minimise errors in recognition. A higher layer count for a given CNN will result in higher accuracy.

Consequently, an increase in the number of layers within a model increases the required computing performance. The 2D convolution part of the algorithm takes the most resources requiring typical processing cores to execute large instruction sets for every

convolution involved. 2D Convolutions may be processed more effectively via the implementation of FPGA chips.

4.1 Optimal CNN performance

There exist two main challenges that need to be overcome to achieve a more optimal implementation for CNNs on processors. The first of which is a needed improvement to data flow throughout the processor. The other challenge is finding a method to process convolution functions themselves more effectively [15]. The latter may be solved via the implementation of digital signal processing blocks (DSPs). Previously, DSPs were implemented as a useful tool for carrying out multiplication but could not perform any floating-point addition. Modern DSPs have evolved to contend with the increasing demand for floating-point arithmetic, which is the method of choice for processing convolutions on processing chips. This lessens the logic requirements for the chip in question. Data flow through the processor may be optimised if each CNN layer is mapped to a CLB with access to an embedded memory block. The data is then passed through each of these layers along the programmable interconnects. This lessens the need for external memory look-ups, which are required for GPU-based processing systems. This implementation allows FPGAs to process images quickly and more efficiently than compared to CPUs. Figure 5 shows a projected CNN performance on both a CPU and FPGA-based solution.

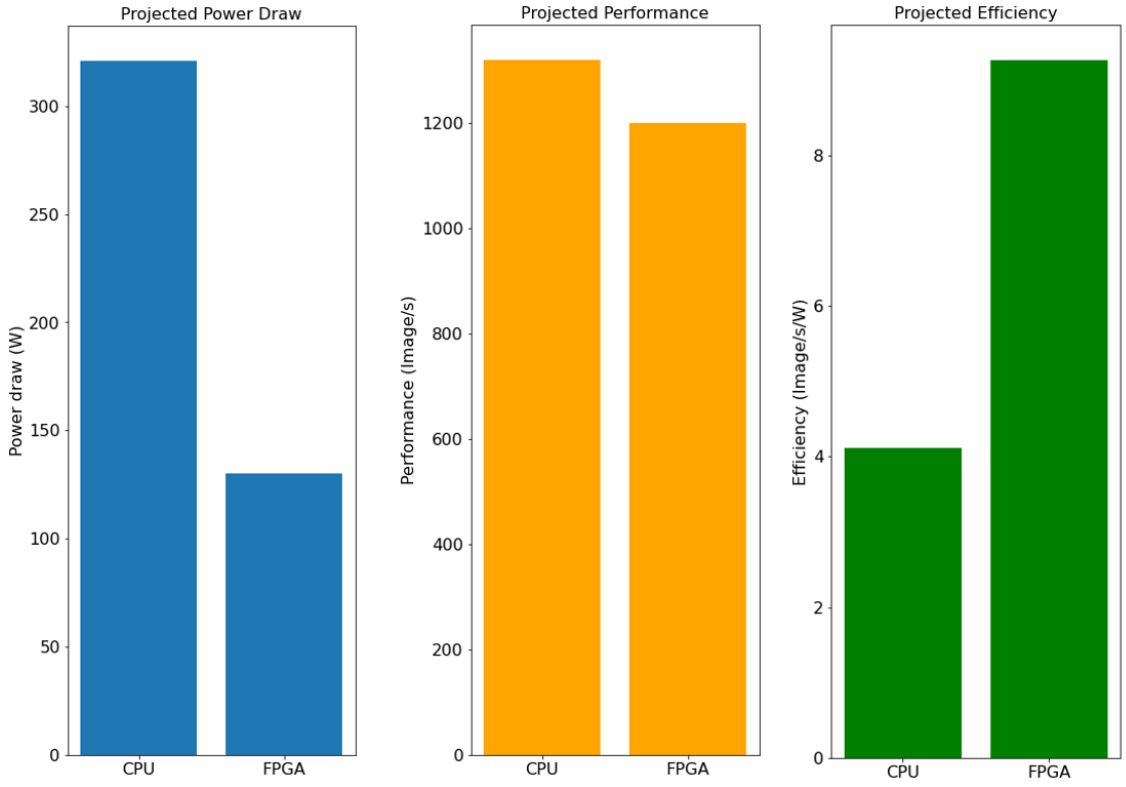

Figure 5: Projected CNN performance on a CPU and FPGA-based implementation. It can be seen that the overall performance, images processed per second, is approximately equivalent across the two solutions. More importantly, it can be seen that there would be considerable efficiency gains, effectively double with regards to this particular CNN. Figure produced from [15].

4.2 Scalable architectures

CNNs are a subset of what is called deep learning techniques. Deep learning techniques illustrate remarkable capabilities to solve even more complex learning problems. Typically, high-performance computing networks were implemented as a solution; however, as algorithms increase in complexity, so do the upkeep costs of maintaining such large data volumes. In particular, the energy requirements for maintaining large ever-expanding data centres are only set to increase unless an alternative, more power-efficient solution is implemented. Three options exist for hardware accelerating deep learning performance, FPGAs, ASICs, and GPUs. While they boast similar performance, they typically require more time to fully implement and are costly compared to FPGAs. FPGAs offer slightly lower performance gains over GPUs; however, they have much lower latency. FPGAs have been proven to be viable for hardware accelerating the restricted Boltzmann machine (RBM). To achieve this hardware acceleration D. L. Ly and P. Chow utilised a method of assigning processor cores to process the RBM algorithm more efficiently [16]. Other forms of optimising the RBM algorithm for FPGAs have also been achieved. Setting RBM processing modules parallel allows each processing core to manage a smaller number of nodes, accelerating processing capabilities[16,17]. Critically, fixed-sized algorithms may be accelerated, but for evolving architectures, a scalable hardware acceleration solution is required. To meet the demands of the increasingly complex problems presented, a scalable, power-efficient solution was needed. C. Wang *et al* illustrates a deep learning accelerator unit (DLAU), which is simultaneously a low-cost, energy-efficient, and, more importantly, scalable architecture fit for use on modern FPGA designs [18].

The DLAU design offers two key improvements over existing forms of FPGA-based hardware acceleration. The first of which is the adaptability of the design. By sectioning the incoming data into tiles, processing becomes much more streamlined for all manner of deep learning utilisations. This provides the opportunity for scaling the architecture up towards more complicated machine learning algorithms. The second improvement is the universality of the design. The DLAU design consists of three central units that can be leveraged for algorithms such as CNNs. This may be extended to other neural networking algorithms as the base three modules are universal to processing such algorithms. To speed up the processing of a CNN, DLAU uses a tiling technique to effectively abstract the many matrix manipulations to dedicated processing units. These processing units are called tile matrix manipulation units (TMMU) and perform the vast majority of operations. It reads in the total weights, nodes, and outputs their respective part sums to other less abundant units. This is implemented in parallel to allow the TMMU architecture to produce part sums every clock cycle [18]. Compared to CPUs, FPGAs present a considerable increase in performance regarding processing deep learning algorithms, as seen in Figure 6.

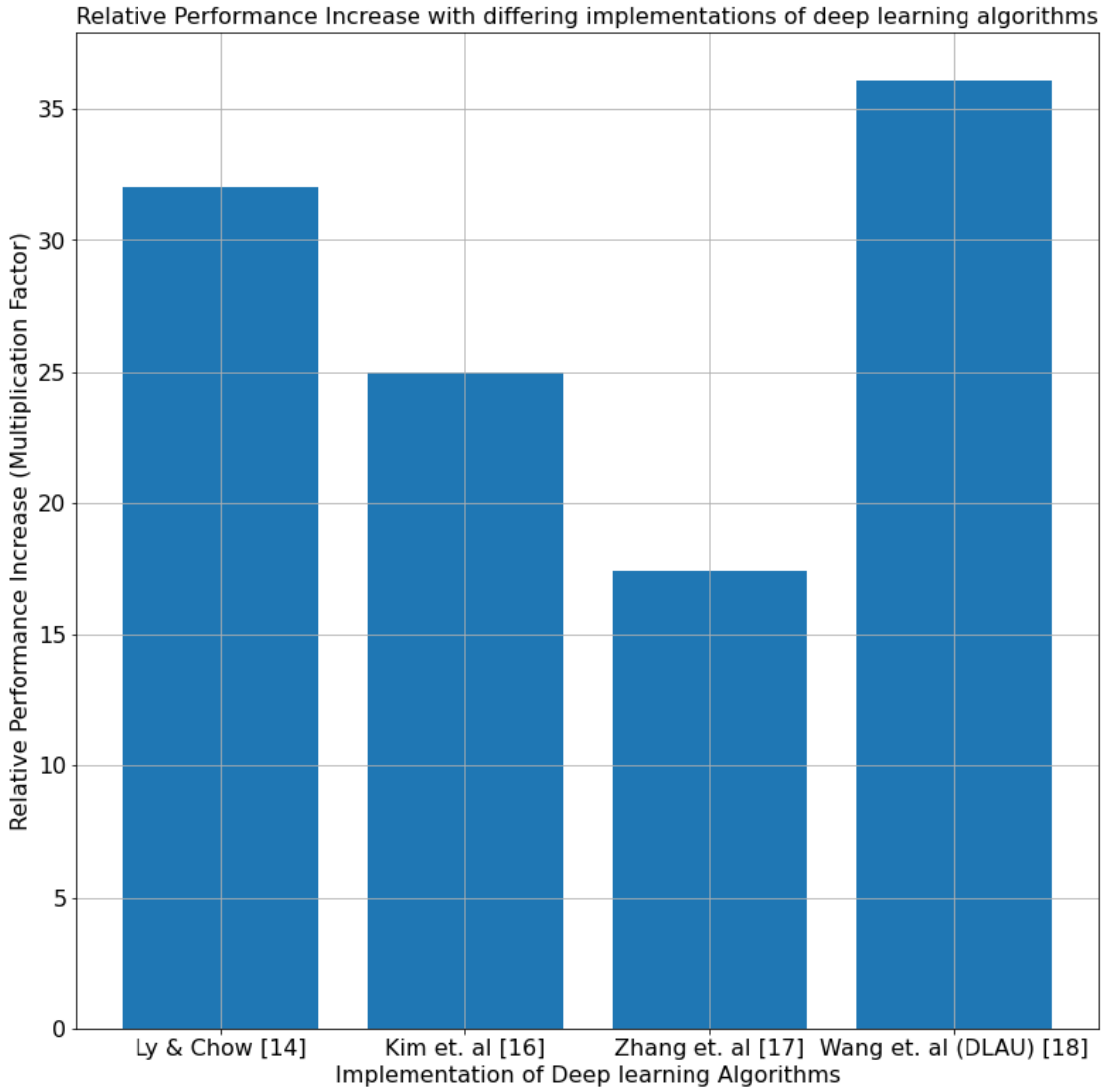

Figure 6: Comparison of performance increases achieved through varying architectures. DLAU shows confidence in offering a considerable performance increase whilst maintaining aspects of scalability. Figure created from [18].

4.3 LHC data processing method

A recent practical example of a proposed FPGA-based form of hardware accelerating machine learning would be the Large Hadron Collider (LHC) upgrade. Particle physics is an ever-evolving area—an area that requires fast, real-time, large-scale data processing. FPGAs have been utilised in this field by providing a much-needed form of hardware acceleration to modern machine learning algorithms. Data sets from recent particle physics experiments are expanding, and so too are their processing algorithms. An extreme overhaul for the Large Hadron Collider (LHC) to transform it into a High Luminosity LHC (HL-LHC) is scheduled in the early 2020s. The upgrade promises an increase in the rate at which collisions occur and the volume of collisions. Respectively, these quantities are known as the luminosity and integrated luminosity [19, 20]. As a result of the LHC upgrade, the Compact Muon Solenoid (CMS) detector will also change to withstand the increased rate of collisions. Without this upgrade, the sensor would become damaged due to increased radiation levels attributed to the increased collision rate.

Consequently, upgrading the CMS detector yields an increase in the processing power required to handle the incoming data [21]. Figure 7 illustrates the estimated CPU resources needed within the next ten years for the CMS detector.

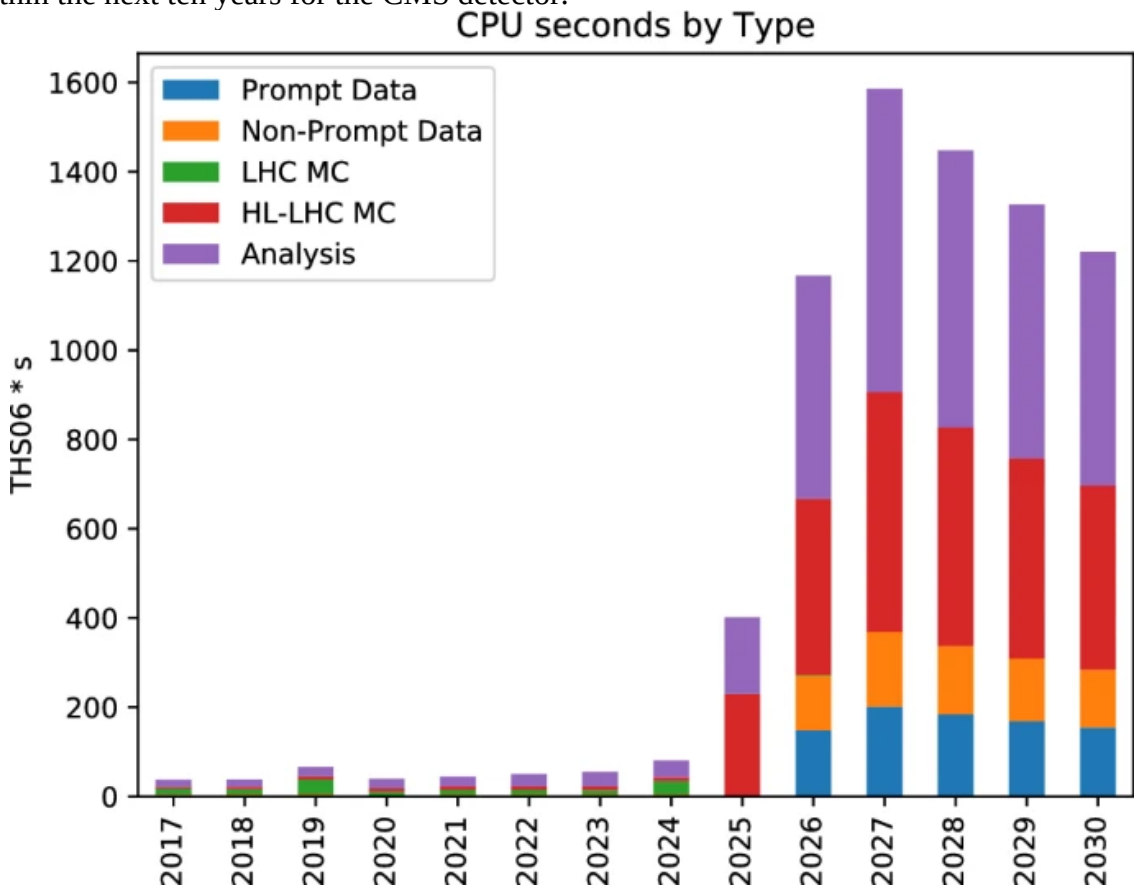

Figure 7: Bar chart illustrating the impact of the intended LHC upgrade. Notice there is a significant portion of CPU resources allocated to the analysis of data sets acquired. THS06 stands for tera (10^{12}) HEP-SPEC06, a standard measure of the performance of a CPU code used in high-energy physics. The planned upgrade requires significantly more CPU resources than practical hence an alternative solution is needed. Figure sourced from [21].

To alleviate this limitation, J. Duarte *et al* describes a heterogeneous architecture, a computing paradigm that uses both CPUs and additional coprocessors such as GPUs, FPGAs, and ASICs [19]. The basis of this design is that CPUs are used as a communication device for the FPGAs. All three coprocessors are much more capable of performing parallelisable tasks compared to CPUs. Due to their affordability and minimal power draw, FPGAs are a clear choice for accelerating machine learning algorithms, specifically big data processing applications.

4.4 Sonar image recognition accelerated

Another example of hardware-accelerated machine learning via FPGAs is the improvements made to sonar image recognition systems. Automatous submarines are used to assess the state of underwater pipelines, mainly for oil and gas. These devices are equipped with side-scanning sonars to evaluate the condition of the subsea pipelines. Structural failing in pipeline networks

can have disastrous consequences if missed and not acted upon quickly; hence autonomous detection is necessary. An FPGA can perform the hardware acceleration needed without requiring as much power than a CPU or GPU implementation [22].

This implementation of FPGAs for hardware acceleration is much like the previously described implementations. The CNN used to identify the potential targets in sonar images was trained to isolate aspects of a sonar image that illustrate an intensity change. A change in tone of the sound waves emitted from the sonar is indicative of a multitude of changes taking place within the range of the sonar. This can be attributed to the movement and features of the seabed, the composition of seabed materials, background noise, and, more importantly, pipeline weaknesses. The volume of received waves is then mapped to a black to white colour scheme. Lighter areas indicate little to no echoed sonar waves were received, implying the surface is smooth and contains very few identifiable features.

In contrast, darker areas indicate extensive feedback, meaning objects are more abundant. This effectively creates a grayscale representation of what is near and around the sonar. To discard unnecessary parts of the resulting image, a series of convolutions were performed to identify the key features.

Much like the previous examples, this integration of CNNs relied heavily on abstracting the CNN into subprocesses that significantly speed up processing. These subprocesses can be classified into four main processing areas, input management, caching, calculation, and learning. Regarding the management of inputs, the FPGA in question contained a high throughput interface allowing it to read and cache multiple input data sets. The caching layer of processing consisted of various modules. The larger of these modules being the image caching section. It is responsible for preparing the inputs for the convolution process and for distributing the data in parallel, leveraging the parallel processing capabilities of the FPGA. This processing area also includes a section for kernel caching used in the convolution process. To correctly order the kernels and load them optimally, another caching system is used. The third and main processing section of the CNN algorithm is contained within a calculation layer. This section performs as one might expect a large volume of calculations. The process starts with a convolution, an initial convolution. The output is then sent through the activation and max-pooling layer of calculation. Once max-pooling has occurred, a counter, indicating the number of times the resulting image has to be convolved, decreases by one. If the output of the counter is greater than zero, the image is sent back to be convolved with another differing kernel. This convolving process, activating, max pooling, and assessing is repeated until the counter reaches an output of zero. Once the output of the counter is zero, the image has finished processing and is flattened for analysis. The flattened data is then presented to the section that governs the learning of the CNN. The flattened data is incident on a neural networking layer, consisting of multiple arrays of nodes or neurons, hence the name. These neurons effectively act as dials, which allow for image classification. A cost function may be used to then train the network. The cost function serves as an indication of the difference between the network prediction and the actual result. Minimising the cost function thus increases the accuracy of future predictions; this is the learning process. The cost function employed in this case consisted of a weighted sum of category probabilities over a range category. The cost function known as the categorical cross-entropy loss (L) can be computed as

$$L = -\sum_{i=1}^{n}\sum_{j=1}^{m} \hat{y}_{ij} \log y_{ij} \tag{4}$$

Where (n) is the number of target images; (m) is the number of categories; $(\hat{y})$ is the expected probability that an image falls into a category; (y) is the actual probability that an image falls into a category [22]. After minimising the loss, the network is considered trained and is ready to be implemented.

In congruence with [15], the FPGA platform performed more efficiently than a workstation processor. Throughout multiple tests, the two platforms achieved comparable loss and accuracy however, it was processing time that enabled the FPGA to outshine conventional implementations. Figure 8 details the results of the tests carried out on both implementations.

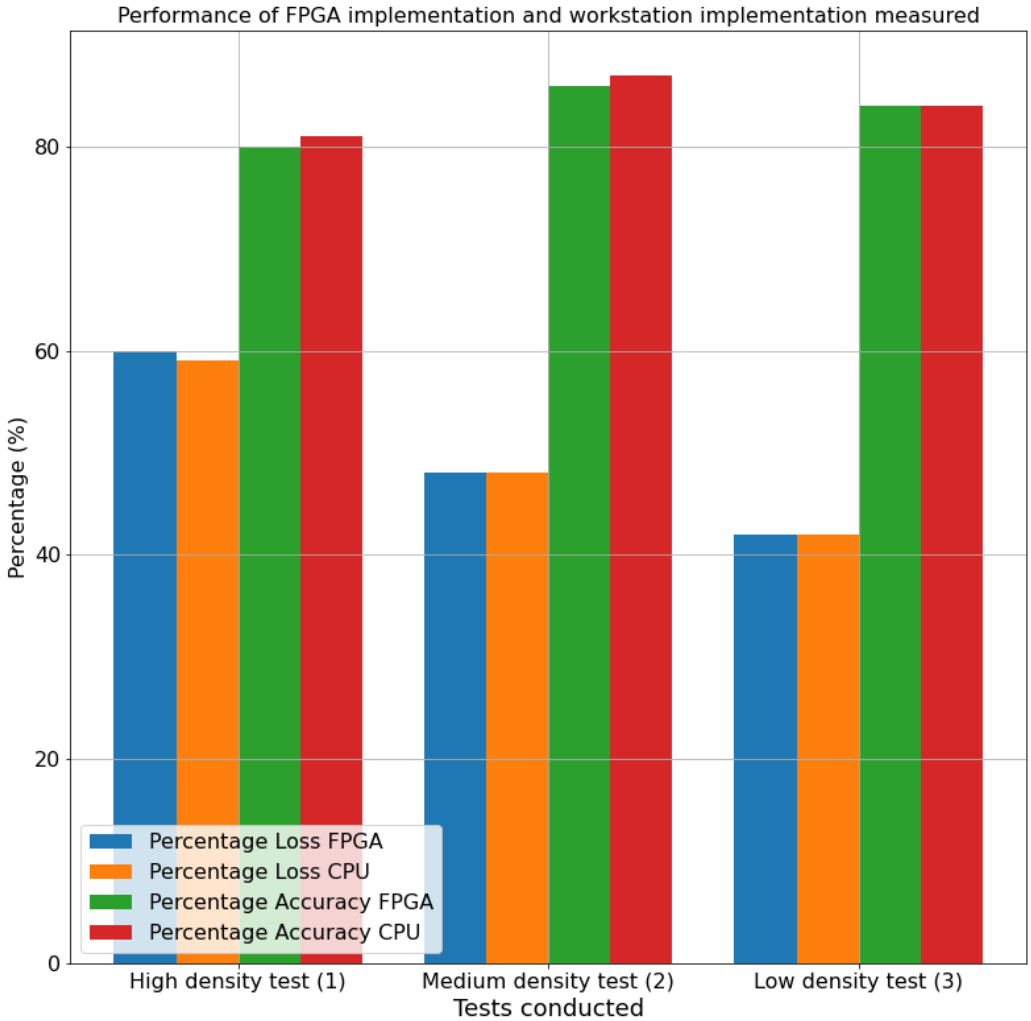

Figure 8: Details the accuracy and loss rates from the subsequent tests. Comparable performance with the workstation implementation is achieved within all three tests. FPGAs implemented provide no impediment to attaining similar results. Figure constructed from [22].

This demonstrates that the parallel processing capabilities of FPGAs translate exceptionally well to accelerating CNN performance in terms of processing time. The time required to process each test on the FPGA system did not fluctuate considerably and offers near real-time analysis with a latency of 2s. A higher loss in the FPGA system with the initial test is explained by the reduced capabilities FPGAs have concerning larger floating-point arithmetic. Since the initial test contained a high-density area, a higher degree of floating-point arithmetic precision would yield a lesser loss, which the workstation CPU is capable of. With comparable results in both accuracy and loss, another factor comes into consideration, the power consumption for each implementation. Figure 9 details the approximate power consumption per workstation.

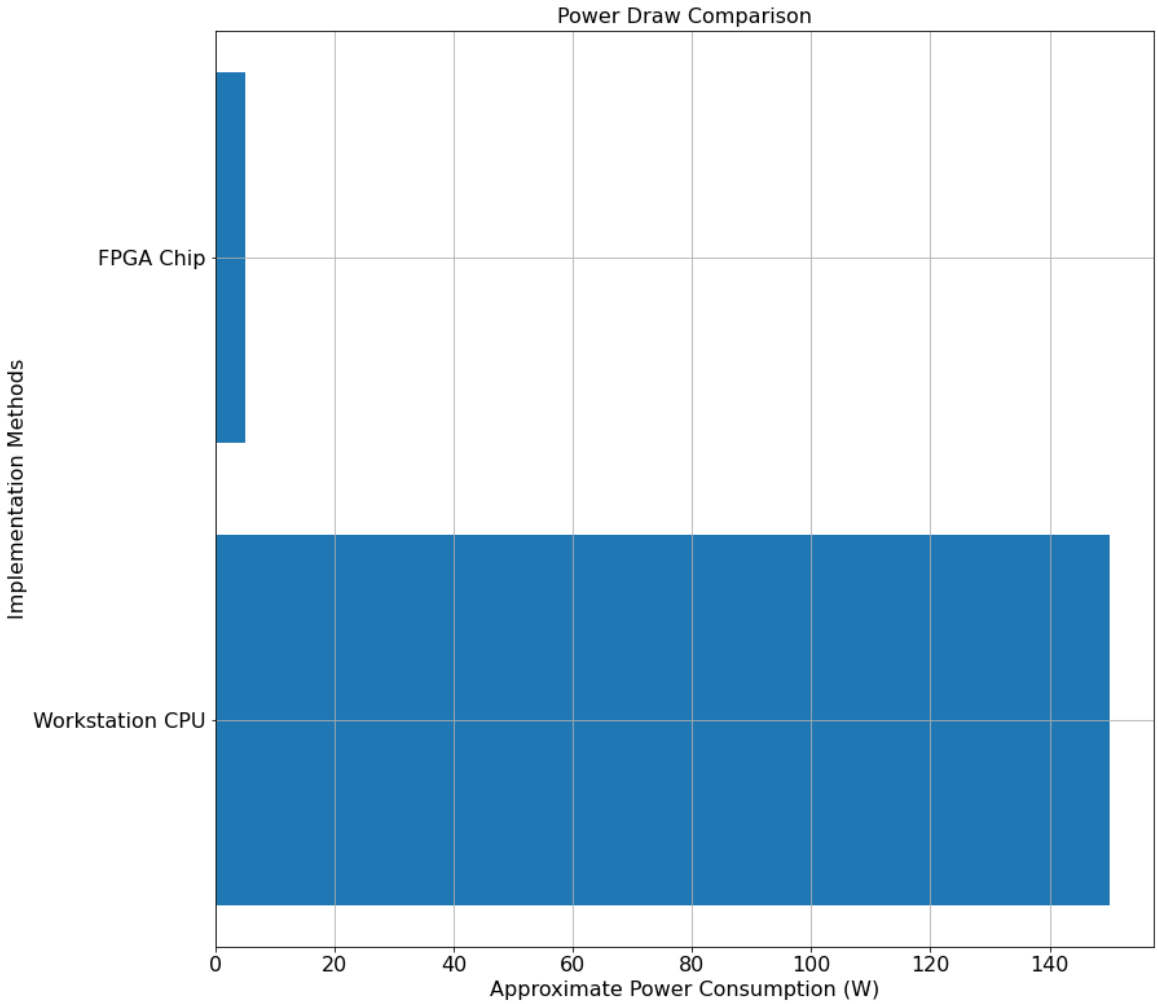

Figure 9: Illustrates the considerable power-saving capabilities of the FPGA implementation. This concludes that the FPGA solution is far superior to the workstation method and can provide comparable results while requiring significantly lesser power—figure produced from [22].

The low power draws from the FPGA do not impose as many restrictions compared to the workstation CPU. Coupled with the near real time processing capabilities and comparable accuracy, the FPGA implementation is far more suited to autonomous underwater vehicles than the workstation implementation. The FPGA, in this context, successfully acts as a form of hardware acceleration to process the CNN. As shown by the 77% increase in recognition times,

FPGAs offer almost real-time data operation while maintaining comparable accuracy to high-performance workstation platforms [22].

5. Improving measuring systems

Measuring systems always have some processing delay between the actual measured event to processing and registering the event. Measuring latency is the term used to describe this delay. To achieve real-time measurements, a system must be optimised to minimise the latency, thus increasing the responsiveness of the system. In modern computing, real-time response is classified as a system that can process input data exceptionally quickly, measurements processed within a timescale of mere milliseconds. The input data is processed and is available immediately. An example of such a system would be the ones found in missile guidance systems.

FPGAs have been employed as a solution to a multitude of latency-sensitive operations. Again, FPGAs act as a form of hardware acceleration for these types of systems. One such example is the recent implementation of real-time sensitivity encoding (SENSE) on magnetic resonance imaging devices (MRI) [23]. Traditionally, SENSE was carried out on a separate dedicated server utilising far more memory resources and could introduce transmission noise. An FPGA implementation could allow near real-time processing of input data without the need for external dedicated servers. This would minimise power usage and computing resources while maximising signal integrity allowing comparably accurate MR images to be produced.

5.1 Exploiting parallelism of algorithms

The inherent parallelism of the SENSE algorithm may be exploited for the FPGA implementation. The algorithm effectively relies on a computationally intensive sequence of matrix manipulations. DSPs blocks are used heavily to provide the resources needed to carry out the required 16-bit floating-point arithmetic to compute this effectively. Matrix inversions and matrix conjugates involve complex numbers which are stored in two parts, 16-bit real numbers and 16-bit imaginary numbers; hence 16-bit floating-point arithmetic is used. The design was implemented alongside a CPU and GPU-based workstation and provided comparable results. Figure 10 shows a comparison between the three implementations illustrating the effectiveness of FPGA low latency processing.

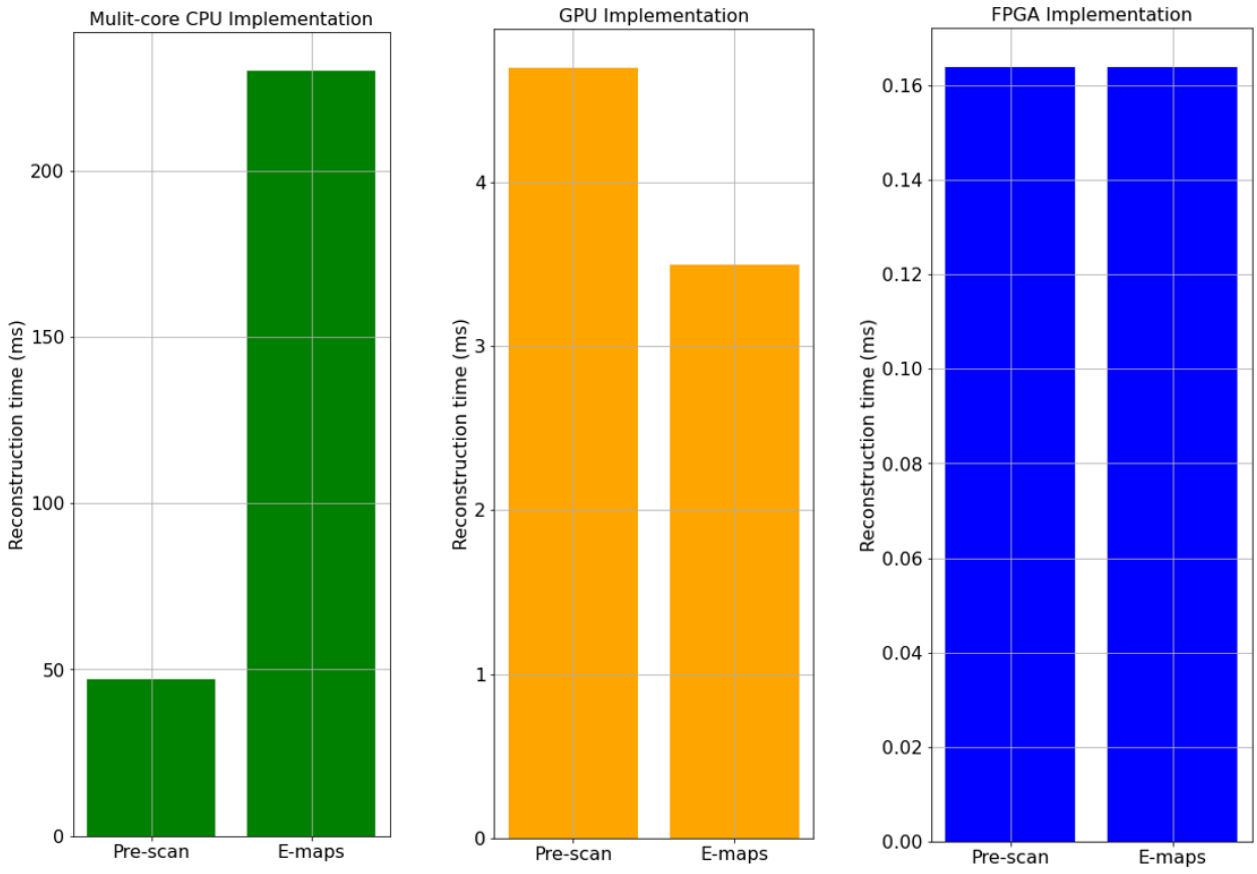

Figure 10: Provides a comparison between different implementations with regards to processing times for different stages. It can be seen that the FPGA implementation is by far the quickest, offering orders of magnitude decreases in latency—figure created from [23].

By acting more efficiently compared to CPU and GPU-based workstations, FPGAs provide the option for developing portable methods of MRI systems. These systems could function on lesser power requirements and still provide the necessary equivalent accuracy regarding reconstructing MR images. The parallelisability of the SENSE algorithm allows for considerable gains in processing speed compared to standard sequential methods. In this context, FPGAs offer a significant improvement to both the portability and efficiency of SENSE-based reconstruction of MR images and do so in a cost-effective manner. It can be stated that measuring systems, specifically systems involving parallelisable algorithms, can be improved upon by utilising FPGAs.

Data processing systems are being streamlined further with the improvement of data processing pipeline schemes. Search engines use such architecture to process and provide page rankings for websites from a searched topic. To improve performance, FPGAs may be used to allocate resources in parallel to each stage of the sorting algorithm [23]. Since the algorithm relies heavily on matrix manipulations, FPGAs may leverage their parallel processing capabilities. The resulting system would be vastly more compact than conventional solutions and again increase system responsiveness due to reduced query latency. With the added ability

to hardware accelerates machine learning algorithms, FPGAs can only improve the capabilities of modern search engines.

6. Big data encryption accelerating Origin and design of the FPGA

Big data, a term used to describe a large volume of structured or unstructured data, is increasing in size with modern machine learning algorithms and the need for larger datasets within physics. Privacy and security of large sets of data are some of the primary issues facing companies today [24,25]. Such big sensitive datasets include search trends by users, users' contacts, and purchasing habits. All of this data may be subject to intrusion if proper encryption of the data in question is not applied. Encrypting such a large volume of data comes with challenges. The main obstacle opposing the encryption of big data sets is the required time to do so via traditional sequential means. Another factor to consider in encryption time is the delay between receiving said data and encrypting it; a latency is present.

6.1. The Advanced Encryption Standard algorithm (AES)

The current algorithm used to encrypt such large volumes of data is the AES algorithm, which comes in various variants, for example, the 256bit of 128bit method. The AES algorithm can be abstracted to a series of matrix manipulations. The data, sectioned into 128bit blocks, is translated into a 4x4 matrix, which is then subject to state manipulations; this is referred to as the AES-128 algorithm. The resulting matrix is effectively the encoded data, which may be decrypted using the same state manipulations in reverse [26]. Since the architecture of the state operations remains the same throughout the development of the cyphertext, parallelism may be employed to speed up the resulting encryption time. Undoubtedly this will consume more resources on the FPGA chip as processes are not being reused. Still, with the capabilities and logic density of modern FPGA chips increasing, this should not pose much of an issue.

S. Chen *et al* improves upon the FPGA implementation [27,28] for big data encryption by streamlining the techniques used within the algorithm. Figure 11 illustrates the performance gains the improvements on the techniques achieved.

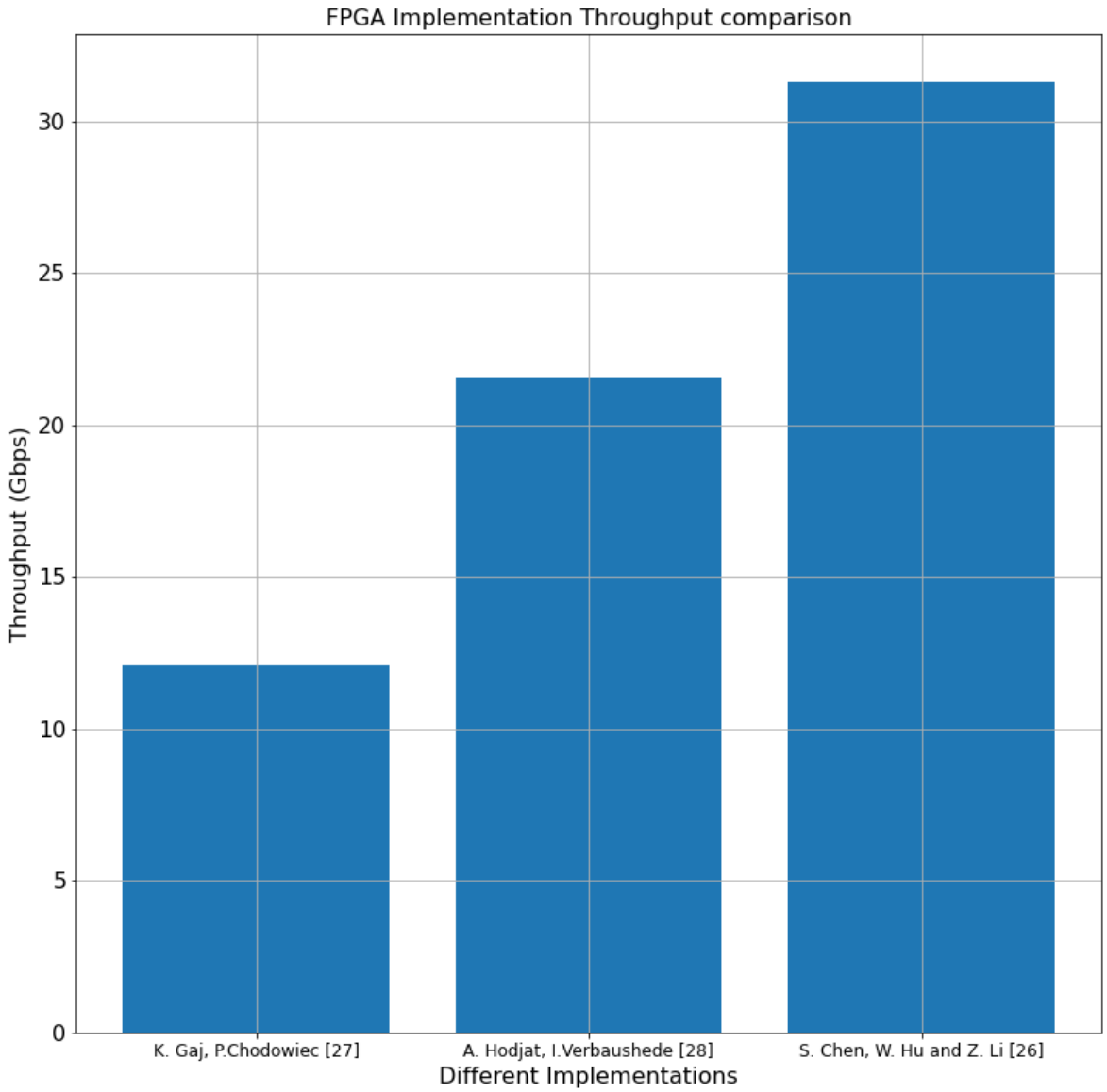

Figure 11: Demonstrates the performance gains through using a more optimised method of data processing. Significant performance gains can be achieved by optimising how data is processed by exploiting the parallelism of algorithms. Figure adapted from [26].

High throughput devices are fundamental to encrypting big data sets. FPGAs offer both the required throughput, and the latency decrease big data encryption needs to be sustainable. Due to their adaptability and power efficiency, FPGAs looks to be the solution to managing high volumes of data and encrypting them faster. While ASIC chips may offer increased performance, their lack of reconfigurability in this area will likely mean that use cases will be limited.

7. Discussion

FPGA chips provide a needed improvement on how data is processed effectively. Further developments within the field of machine learning increase sequential computing requirements significantly. Time-sensitive measuring systems require ever-lower latency than ever before regarding MRI and high energy physics applications such as the LHC. The scale of data is also increasing with an ever-evolving digital world requiring systems to respond ever quicker. FPGAs present a solution to these escalating problems.

Machine learning may be accelerated through scalable hardware implementations on FPGAs. CNNs, the most predominant machine learning algorithm, may be adapted to FPGA architecture using the abundance of DSP blocks to provide the necessary computational ability. The basis of this algorithm requires a large volume of convolutions, which are essentially large matrix operations. By leveraging the architecture of FPGAs and their affinity for parallelism, CNNs may be accelerated significantly. CNNs processed via a sequential method require much more CPU and GPU resources. This method of processing requires far more power and thus proves to be a less power-efficient method.

Practical applications have already benefited from the increase in performance provided by FPGAs. A sonar image recognition system, a trained CNN has outperformed workstation implementations. The reduced power consumption of the FPGA implementation indicates such an implementation is possible for fully autonomous underwater vehicles. With the scalability of the DLAU architecture, projects such as the planned LHC upgrade could benefit greatly from this design. The scalable architecture would provide the required processing performance increase the LHC requires if the upgrade is to go ahead. This would allow the processing of data to become more power-efficient. If the sequential tasks were properly parallelised the latency between results and measurements could be reduced significantly.

Real-time measurement interpretation may also be achieved through the implementation of FPGAs. Measurement system responsiveness may be increased without sacrificing processing capabilities. Systems may be made more compact due to not requiring dedicated external processing servers or computing units. Systems utilising FPGAs may be scaled down in size, resulting in creating portable, real-time measurement systems. The SENSE algorithm responsible for image reconstruction of MR images may be parallelised and accelerated through FPGA hardware. FPGAs improve the responsiveness and reduce the latency of this algorithm to near real-time interpretation.

Big data encryption algorithms such as AES may also be accelerated. The world is becoming more integrated with technology every day, and as such, it generates a vast amount of data. This data needs to be encrypted if the data is private information such as browser history or shopping trends. To accelerate big data encryption, the AES algorithm may leverage the inherent parallel processing capabilities of FPGA chips. The AES algorithm consists of multiple matrix manipulations that may be quickly processed via the multitude of DSP blocks present on modern-day FPGA architectures. This implementation of FPGA logic would effectively increase the throughput of big data and allow a much more responsive encryption system. FPGAs may be scaled without any loss in performance and provide a more power-efficient solution, FPGAs are far more sensible to implement than sequential methods of CPUs and GPUs.

8. Conclusion

It can be concluded that FPGAs offer a significant increase in computational performance. This can be attributed to their ability to provide proper parallel processing. They are also more power-efficient compared to that CPU or GPU implementations. In machine learning, they act as a form of hardware acceleration to increase the performance of an algorithm while also being scalable. Regarding measuring systems, they allow for the creation of near real-time responsive systems thanks to their high throughput capabilities. Finally, they may be used to accelerate the performance of encryption algorithms used to handle big data. This increase in speed while remaining power efficient culminates in a significant performance boost for any application utilising FPGAs. This performance increase can scale as programs become increasingly parallelisable. The comprehensive use cases for FPGAs will continue to grow as programs become increasingly parallelisable. The affinity for parallel processing, exceptional power efficiency, low cost, and high throughput indicates that FPGA applications in physics and beyond will continue to grow considerably.

References

[1] "Intel Completes Acquisition of Altera" https://newsroom.intel.com/news-releases/intel-completes-acquisition-of-altera/#gs.o693v9. Accessed: 10-12-2020.

[2] "The quarterly journal for programmable logic users: XCell" https://www.xilinx.com/publications/archives/xcell/Xcell32.pdf. Accessed: 14-12-2020.

[3] "The History of FPGAs" https://web.archive.org/web/20070412183416/http:/filebox.vt.edu/users/tmagin/history.htm. Accessed: 14-12-2020.

[4] J. Geist, K. Y. Rozier and J. Schumann "Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems," *Runtime Verification. RV 2014. Lecture Notes in Computer Science*, vol 8734, 2014, DOI: https://doi.org/10.1007/978-3-319-11164-3_18

[5] P. H. W. Leong, "Recent Trends in FPGA Architectures and Applications," *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*, pp. 137-141, 2008, DOI: https://doi.org/10.1109/DELTA.2008.14

[6] R. Kastner, J. Matai and S. Neuendorffer, "Parallel Programming for FPGAs," 2018, http://arxiv.org/abs/1805.03648 Accessed: 14-12-2020

[7] "Altera FPGA Architecture," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf Accessed: 10-12-2020

[8] "Overview of Look-up tables in FPGA Design," https://hardwarebee.com/overview-of-lookup-tables-in-fpga-design/. Accessed: 14-12-2020

[9] "Enhanced Intel® SpeedStep® Technology for the Intel® Pentium® M Processor," https://download.intel.com/design/network/papers/30117401.pdf Accessed: 14-12-2020.

[10] "Intel Halts Development of 2 New Microprocessors," https://www.nytimes.com/2004/05/08/business/intel-halts-development-of-2-new-microprocessors.html Accessed: 15-12-2020

[11] "Amdahl's Law," https://demonstrations.wolfram.com/AmdahlsLaw/ Accessed: 14-12-2020.

[12] G. M. Amdahl, "The validity of the single processor approach to achieving large-scale computing capabilities," *In Proceedings of the April 18-20, 1967, spring joint computer conference (AFIPS '67 (Spring))*, 1967, DOI: https://doi.org/10.1145/1465482.1465560

[13] "Amdahl's Law," https://en.wikipedia.org/wiki/Amdahl%27s_law Accessed: 14-12-2020.

[14] C. Zhang et al., "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*

(FPGA '15), 2015, DOI: https://doi.org/10.1145/2684746.2689060

[15] "Efficient Implementation of Neural Network Systems Built on FPGAs, and Programmed with OpenCL™," www.intel.co.uk/content/dam/www/programmable/us/en/pdfs/literature/solution-sheets/efficient_neural_networks.pdf. Accessed 14-12-2020.

[16] D. L. Ly and P. Chow, "A high-performance FPGA architecture for restricted Boltzmann machines," *In Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays (FPGA '09)*, 2009, DOI: https://doi.org/10.1145/1508128.1508140

[17] S. K. Kim, L. C. McAfee, P. L. McMahon and K. Olukotun, "A highly scalable Restricted Boltzmann Machine FPGA implementation," *2009 International Conference on Field Programmable Logic and Applications*, pp. 367-372, 2009 DOI: https://doi.org/10.1109/FPL.2009.5272262

[18] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 3, pp. 513-517, 2017, DOI: https://doi.org/10.1109/TCAD.2016.2587683

[19] J. Duarte, P. Harris, S. Hauck, *et al.* "FPGA-Accelerated Machine Learning Inference as a Service for Particle Physics Computing," *Comput Softw Big Sci 3: 13 (2019)*, 2019, DOI: https://doi.org/10.1007/s41781-019-0027-2

[20] G. Apollinari, I. B. Alonso, O. Brüning, M. Lamont and L. Rossi "High-luminosity large hadron collider (HL-LHC): preliminary design report," 2015, DOI: http://dx.doi.org/10.5170/CERN-2015-005

[21] The HEP Software Foundation., Albrecht, J., Alves, A.A. *et al.* "A Roadmap for HEP Software and Computing R&D for the 2020s," *Comput Softw Big Sci 3, 7 (2019)*, 2019, DOI: https://doi.org/10.1007/s41781-018-0018-8

[22] C. Wang, Y. Jiang, K. Wang and F. Wei "A field-programmable gate array system for sonar image recognition based on convolutional neural network," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 2020, DOI: https://doi.org/10.1177/0959651820939345

[23] M. F. Siddiqui, A. W. Reza, A. Shafique, H. Omer, J. Kanesan, "FPGA implementation of real-time SENSE reconstruction using pre-scan and Emaps sensitivities" *Magnetic Resonance Imaging*, vol. 44, pp. 82-91, 2017, DOI: https://doi.org/10.1016/j.mri.2017.08.005.

[24] V. Mayer-Schonberger and K. Cukier, "Big Data: A Revolution that Will Transform How We Live, Work and Think," Boston: Houghton Mifflin Harcourt, 2013.

[25] X. Meng, X. Ci, "Big data management: Concepts, Techniques and Challenges," *Journal of Computer Research and Development*, 2013, pp. 146-169.

[26] S. Chen, W. Hu and Z. Li, "High-Performance Data Encryption with AES Implementation on FPGA," *2019 IEEE 5th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing, (HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*, 2019, pp. 149-153, DOI: https://doi.org/10.1109/BigDataSecurity-HPSC-IDS.2019.00036

[27] K. Gaj, P. Chodowiec, "Fast Implementation and Fair Comparison of the Final Candidates for Advanced Encryption Standard Using Field Programmable Gate Arrays," *Proceedings of the 2001 Conference on Topics in Cryptology: The Cryptographer's Track at RSA*, 2001, DOI: https://doi.org/10.1007/3-540-45353-9_8

[28] A. Hodjat and I. Verbauwhede, "A 21.54 Gbits/s fully pipelined AES processor on FPGA," *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2004, pp. 308-309, DOI: https://doi.org/10.1109/FCCM.2004.1