Vitiate

Diego Remírez, Gabriel Ruíz

Abstract

This paper revolves around the analysis of neural networks (NN) deployment on the main hardware platforms commercially available: CPUs, GPUs, and FPGAs, the main objective being to evaluate performance and power efficiency.

Introduction

Modern neural network-based artificial intelligence (AI) systems show an ever-increasing parameter usage to address more complex problems.

This makes networks execution exponentially complex, both forward and backward processes, which, in turn, requires brute force computation, usually delivered by GPUs.

GPUs provide excellent parallel execution and great availability to price ratio, as well as widespread and familiar development environments. However, they are not intended for industrial purposes and are energy demanding.

On the other hand, application-specific integrated circuits (ASICs) are the fastest and the most energy-friendly hardware one can find. They are however scarce, lack the flexibility to perform tasks other than what they are designed for, and they are highly costly. ASICs remain therefore used only by development teams with great budgets.

This paper focuses on an in between hardware: field programmable logic arrays or FPGAs, which will be compared against the CPU and GPU.

Precedents

This paper will only work with deep neuronal networks (DNN) and, as such, related concepts are assumed and only implementation clarifications will be made.

FPGA

Programmable logic devices whose internal networking and functionality is specified by a specialized hardware description language. This means that FPGA programming is, in fact, hardware manipulation.

OpenCL

Open-source computation standard for parallel computing. Developed by Apple and maintained by Khronos group, has been implemented by different companies (Intel, NVIDIA, XILINX, etc.) allowing simultaneous management of multiple computing devices (CPU, GPU, FPGA, etc.).

Activation functions

Network activation functions are key in learning convergence and, depending on the task, a well-chosen function may generate better suited architectures.

Even though any activation function might be chosen, more complex functions (like sigmoid or hyperbolic tangents) will negatively impact FPGA resources usage, while others like Rectified Linear Unit (ReLu) are much more resource friendly.

We have chosen the Leaky Relu variant (LRL) as the main activation function to go, with the negative domain factor being a power of 2 (which implementation-wise, translates into just bit-shifting).

Quantization

Information numeric representation is important for both network efficiency and effectiveness.

FPGAs can use float values but at a high cost in resources. Therefore, integer representation is much more valuable.

Float representation is normally used in machine learning due to its precision and handiness, even more, float values around unity modulus can reduces overflow problems and allow for local behaviors.

Seeking to merge both representations, we observe that, for LRL:

Given

$$q \in \mathbb{R} \mid q > 0$$

And

$LRL(x)$:

$$\underline{\mathbb{R} \rightarrow \mathbb{R}}$$

$$x \rightarrow \begin{cases} \frac{x}{a} & x < 0 \\ x & x \geq 0 \end{cases}$$

With

$$a \in \mathbb{R} \mid 0 < a < 1$$

It follows

$$LRL(qx) = q * LRL(x)$$

Finally, for an arbitrary LRL neuron layer

$$C(qI, q^2B, qW) = q^2 * C(I, B, W)$$

Where

I	Layer input vector [n]
B	Layer bias vector [m]
W	Layer parameters matrix [n x m]

By forcing q to be a power of 2 and normalizing each layer output dividing by q, it results in a highly efficient structure which permits float trained networks to be evaluated using integer representation with any desired precision.

The integer size choice relies in a compromise between precision and speed and memory usage. Results here shown use 32 bit-integer with $q = 2^7$.

Hardware

Device	Main features
CPU Intel Core i9-10900KF 3.70 GHz	• RAM 64 GB 48 GB/s • 10 cores • 20 threads • Max. Freq. 5.30 GHz
GPU EVGA GeForce RTX 3070 Ti FTW3 ULTRA GAMING 8GB GDDR6X	• Dedicated RAM 8GB 608GB/s • CUDA cores 6144 • Clock 1860 MHz

FPGA
 	
MUSTANG F100-A10 with Intel Arria 10 1150 GX
	• Dedicated RAM 8GB 19.2 GB/s
	• Logic blocks
	  ○ LE 115000
	  ○ ALM 1708800
	  ○ DSP 1518
	• Clock up to 1000 MHz

CPU is only used as an execution time benchmark (with single thread execution) and to manage GPU and FPGA high-level operations. It's also referred to as the main test platform for the scratch developed neural network software and the ongoing functionalities addition.

The chosen GPU is a modern model (2020) using NVIDIA's state of the art Ampere architecture. In contrast, the FPGA is a relatively old model (2013) with a fairly high count of logic blocks but sporting a much less capable RAM.

Both devices are connected to the main PC via PCIe express bus, with PCIe3 x8 for the FPGA and PCIe4 2x8 for the GPU.

Software

Each device implementation has been built into a library that can be called from a handler, both in C++ and Python.

CPU

CPU code is written using C++17 and constitutes the main benchmark against which to test GPU and FPGA metrics.

Even though the CPU is multi-thread, code is kept single-threaded for sake of simplicity, while allowing the introduction of some parallelism through AVX vector processors.

GPU

GPU code has been developed using the CUDA tool set, which is NVIDIA proprietary and thus reserved for their graphic cards only.

CUB and CUBLAS libraries are extensively used because they are heavily optimized for reduction and vector and matrix operations. On the other hand, careful code abstraction and functions and structures generalization ensures an intuitive and somehow simple portability from CPU to GPU.

FPGA

FPGA implementation is being undertaken using Intel's implementation of OpenCL standard, covering FPGA support.

In opposition to traditional languages, such as Verilog and VHDL, OpenCL eases FPGA and peripherals (PCIe, RAM) management and allows for much shorter development times.

Carried out tests always employ networks with a total footprint of less than 8GB (RAM size) which, for 32-bit integers, means about $2 * 10^9$ parameters.

For larger sizes, required data can be downloaded in real time through the PCIe bus, but at the expense of great time penalization.

Procedure

This paper aims to evaluate execution speed progression and power usage of each device over neural network parameters increment.

Parameter number depends on layer number and number of parameters per layer. The following tests cases will keep the amount of layers fixed while varying the number of neurons per layer. This is done because adding or subtracting layers gives a linear behavior, while increasing neuron count per layer translates into an exponential increment in parameter number hence being much more resource demanding.

Moreover, modern deep connected networks seek to use architectures with limited fully connected layers and many neurons per layer.

To get reliable measurements of power usage and speed, the first 10 execution iterations won't be considered, as to diminish the effect of initializations, subprocesses and whatnot.

Tested NNs will have 16 inputs and 5 layers with the same neuron count for each layer.

Power usage will be measured using a ± 0.1 W error wattmeter. Performance will rely on CPU high-precision timers, with ± 10^{-6} s error, to timestamp execution over several iterations. This metric's inverse result gives as desired iterations per second.

Results

Neurons per layer	N. of parameters	Power usage [W]		
		CPU	GPU	FPGA
160	130560	63	145	43
320	517120	66	147	40
480	1159680	68	151	37
640	2058240	67	164	38
800	3112800	66	166	36
960	4623360	61	175	36
1120	6289920	59	182	36
1280	8212480	58	190	35
1440	10391040	61	202	35
1600	12825600	61	203	34

Table 1 Power usage

Neurons per layer	N. of parameters	Performance [1/s]		
		CPU	GPU	FPGA
160	130560	83000	11000	5800
320	517120	21000	9100	4300
480	1159680	9700	10000	3700
640	2058240	5200	9900	3400
800	3112800	2800	8800	3000
960	4623360	1200	8300	2900
1120	6289920	870	7700	2700
1280	8212480	620	7300	2600
1440	10391040	460	6500	2400
1600	12825600	370	5100	2200

Table 2 Performance

Standard deviation oscillates between ± 4 W and ± 10%, for power and time respectively. As these values are far greater than the instruments errors,

they will be considered as the main source of error.

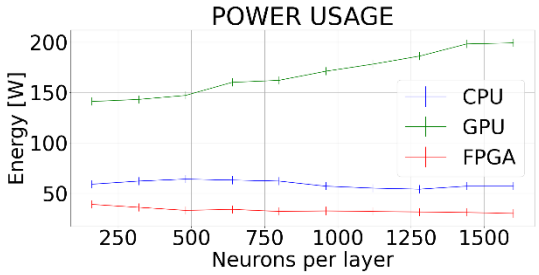

Figure 1 Power usage

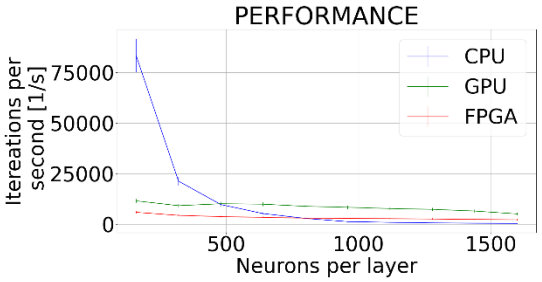

Figure 2 Performance

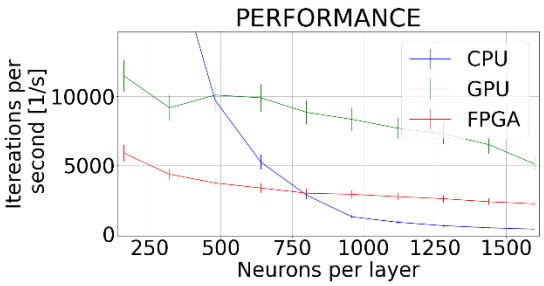

Figure 3 Performance area focus

Conclusions

From the data, we can stablish that:

1. FPGA power usage is stable in general and lesser than that of GPU and CPU.
2. GPU power usage increases in a linear fashion up to 200 W, where it stabilizes.
3. For small NNs, (< 480 neurons per layer) CPU gives the best performance, but it decreases exponentially.
4. GPU performance is 2 to 3 times greater over the FPGA.

We also observe that:

1. An updated FPGA with a faster RAM should be able to match the GPU performance with much better energy efficiency.
2. Being a great tool, OpenCL still requires important time investment and effort for development.