Abstract

Graphical Processing Units (GPUs), originally designed to accelerate 3D graphics, have been increasingly adopted for Machine Learning (ML) tasks due to their high parallelism and memory bandwidth. This paper explores the use of GPUs for ML computations, particularly focusing on how frameworks like OpenGL and WebGL can leverage GPU capabilities to enhance training performance. The growing support for ML in JavaScript-based frameworks positions the web as a promising platform for high-performance ML applications, with potential for real-time tasks such as computer vision.

INTRODUCTION

Graphical Processing Units (GPUs) have now been popularised in Machine Learning (ML) computations. A GPU is a single chip processor designed to accelerate 3D graphics to a user and has been 'adopted' into the personal computer replacing the Central Processing Unit (CPU) for calculations. Steinkraus et al. [1] discusses how dedicated hardware for ML tasks is out-performed by GPUs in terms of memory bandwidth, and parallelism. They implement a 2-layer Neural Network, fully connected network on a GPU which has bi-directional communication with the CPU to fetch training data and output weights.

Neural Networks (NN) are supervised models reliant on input data and weights to arrive at a target output. Every NN consists of layers of nodes with a visible source and end nodes while the intermediate nodes are hidden. These nodes mimic brain neurons to make decisions via weights to arrive at conclusions.[2]

OpenGL is an API (Application Programming Interface) that provides the developer access to the GPU by various functions. Using these functions the developer is able to use 'Shaders' which are programs to avail of the GPU. These shaders run on stages of the Graphics Pipeline. Fernando. [3] discusses the Graphics Pipeline which can be simplified into two sections. 'Vertex shaders' and 'Fragment shaders'. Inputs such as 3D coordinates are transformed into 2D locations on screen and map textures or colour to this data. Steinkraus and others. [1] exploited these programs for the General Purpose Parallel Processor by passing training sets onto the GPU and computing large amounts of information in parallel to the CPU where the CPU can adjust learning parameters. This is repeated until a final set of trained weights is sent from the GPU to the CPU.

Javascript Environment with GPUs

WebGL (Kronos,2011) is a framework for OpenGL built on JS (Javascript). It retains much of the functionality of OpenGL functions but is accessible via web browsers. This allows for web browser communication with the user's GPU allowing for client or cloud rendering. Smilkov et al. [4] implement 'TensorFlow.js' a ML Library built on JS which takes advantage of WebGL technologies. Primarily languages such as python are used in ML for its extensive support for ML (Sklearn, PyTorch). as it latches onto C++ libraries making it 'compilable' rather than 'interpreted'. [4] JS is an interpreted language so it is outperformed by C++ or Java for numerical computations. New frameworks such as WebAssembly (Haas et al., 2017) and WebGL have emerged to compensate for this. .

Optimisation of OpenGL applications

The Graphical Pipeline has rapidly grown in complexity with new emerging technologies such as ‘Real-time Ray Tracing’. [6] Fernando describes how to identify these bottlenecks to minimise performance loss when rendering 3D Scenes. The author discusses identifying these overheads via stages in the pipeline. By varying the workloads such as ‘increasing data points in vertices’ or ‘increasing shader calls’ the frames per second (fps) can vary, differentiating the stages and identifying the bottleneck. Repeating this process until the preferred performance is achieved. Many optimisation techniques are covered in relation to the Rendering process but these techniques can also be applied in ML optimisation on the GPU.
Techniques such as:
Fragment Shading and Vertex Processing: By modifying the length Vertex or Fragment programs such as ‘number of cycles to render high detailed pixels’ or ‘cycles to draw complex models’, GPU performance can be increased reducing the size of data passed into the Graphics pipeline. In ML terms, training data can be normalised and processed to reduce memory bandwidth between the GPU and CPU. Fragment shaders require many calculations at pixel level, while NN requires large-scale parallel processing at similar complexity. The GPUs parallelism allows for acceleration of Fragment processing or in an ML environment, matrix calculations.
Texture Bandwidth:
When rendering objects the CPU and GPU are communicating bi-directionally (both ways). This communication is limited as the CPU-to-GPU bandwidth is ‘1GB/s’. [1] while GPU-to-CPU is limited. Modern GPUs have optimised this issue using a FIFO (First In First Out) cache to store the results of the most recently transformed vertices.[3] A hit on this cache allows the result data to be passed instead of using shaders to calculate. In relation to ML, input data such as Images use high memory to store and process. By caching it can reduce the computational costs of transferring image data to the GPU and performing calculations.
Batch Size Maximization:
A ‘Batch’ is a group of geometric data types rendered in a single draw call (gl.drawArrays()). The size of the batch is the number of data it contains. [3] By maximising the number of data submitted with every draw call, CPU work can be minimised. In the ML environment, maximising the training, validation or test data to the model keeps the GPU’s cores occupied and reduces bandwidth waste.

Observations:

Leveraging parallelism and computational power of GPUs is important to complex ML models. Though modelled for Graphical use, Steinkraus et al takes advantage of the GPUs functionality compared to dedicated hardware with a ‘X3’ speedup in training and test time. Though not discussed, the results of the experiment could be further optimised by implementing OpenGL performance techniques by processing the size of training data, maximising the data fetched into training and utilising the GPU cache previously discussed.

When developing the JS library for ML applications, Smilkov and others observed ‘two orders of magnitude’ faster rate compared to plain JS backend. Though WebGL out performed plain JS computation it had been outperformed by CUDA by a gap of 3-10x.[4]

‘CUDA’ (Nvidia,2007) is another API but specifically targeted to general purpose processing. With the emergence of WebGPU this gap could be shortened.

By leveraging JS’s compatibility with browsers and backend services, Web ML computing can be achieved with Graphical APIs which gives users access to the GPU. With the increasing amount of JS developers ‘2.3 million JS GIT pull requests in 2017). [4] compared to ‘1 million Python (GitHub.com, 2017). The JS environment has the potential to support low-latency applications and complex tasks such as ‘Real time computer vision’ (Jeeliz virtual try on). [7]

References

[1]D. Steinkraus, I. Buck, and P. Y. Simard, “Using GPUs for machine learning algorithms,” *Eighth International Conference on Document Analysis and Recognition (ICDAR’05)*, 2005, doi: https://doi.org/10.1109/icdar.2005.251.

[2]IBM, “What Are Neural Networks?,” *www.ibm.com*, 2023. https://www.ibm.com/topics/neural-netwo r ks

[3]R. Fernando, *GPU gems*. 2006.

[4]D. Smilkov *et al.*, “TensorFlow.js: Machine Learning for the Web and Beyond,” *arXiv:1901.05350 [cs]*, Feb. 2019, Available: https://arxiv.org/abs/1901.05350

[5]A. Tam, “Use PyTorch Deep Learning Models with scikit-learn - MachineLearningMastery.com,” *MachineLearningMastery.com*, Feb. 07, 2023. https://machinelearningmastery.com/use-pytorch-deep-learning-models-with-scikit-learn/ (accessed Nov. 13, 2024).

[6]“What is real-time ray tracing?,” *Unreal Engine*. https://www.unrealengine.com/en-US/explainers/ray-tracing/what-is-real-time-ray-tracing

[7]“JEELIZ Sunglasses virtual try-on,” *JEELIZ.COM*, 2024. https://jeeliz.com/sunglasses/?sku=rayban_boyfriend_noir_marron_degrade (accessed Nov. 13, 2024).