Hijacking the Embedded GPU: Accelerating computation on Raspberry Pi

1st Eric Rippey
Virginia Tech Computer Science Dept.
Blacksburg, U.S.A
erippey3@vt.edu

***Abstract*—In the evolving landscape of Industry 4.0, real-time processing and intelligent decision-making are paramount to optimizing industrial automation and smart manufacturing. However, the reliance on cloud-based computation introduces undesirable latency in time-sensitive applications. This proposal explores an unconventional approach: repurposing the embedded GPU found in the Raspberry Pi 5 to extract further performance from edge devices. Traditionally dedicated to rendering graphics, this GPU exhibits untapped computational potential that can be redirected to accelerate computationally heavy tasks such as running neural network inference. We outline a methodology to adapt and define GPU functionalities beyond their conventional scope, evaluate performance trade-offs, and present preliminary experimental insights. We intend to realize this functionality by porting OpenCL kernels to Vulkan, which the Pi 5's GPU supports.**

***Index Terms*—Embedded Systems, Neural Network, GPU Acceleration, Edge Computing, Raspberry Pi, OpenCL, Vulkan**

I. Introduction

The current era of industry is often referred to as Industry 4.0, or the fourth industrial revolution. It is marked by the shift in industry operations through the use of automation and smart technologies. This transformation has seen the emergence of smart factories and intelligent machines that significantly improve productivity and operational efficiency. Technology is at the center of this revolution, including the emergence of Internet of Things (IoT), Cloud Computing, Artificial Intelligence, Machine Learning, and Edge Computing. These technologies provide a combination of real time data processing and adaptive decision making capabilities.

In industry today a popular edge device commonly used is the Raspberry Pi. This is a low cost, low power, edge device with high configurability which makes it desirable for deployment and embedded applications. A common use case for the Raspberry Pi is for monitoring fields such as, temperature, humidity, light intensity, water level, power consumption, current, and voltage [2], [3], [7].

Data collection tasks such as the ones outlined are only a single step of a larger picture. Smart sensors and IoT devices are used in conjunction more computationally expensive algorithms that recognize patterns, detect anomalies, do classification, and make predictions to improve industry output and increase efficiency. Artificial Neural Networks are particularly strong at these tasks. These more computationally expensive steps are not commonly done on the the edge devices themselves. This paper covers our attempt to utilize the popular edge device, the Raspberry Pi 5, to compute for more computationally expensive tasks with the end goal of being able to run neural network inference. This paper follows our attempt to leverage the embedded GPU of the Raspberry pi which is typically only used for graphical processing. This approach was aided by the clvk library, an open sourced implementation of OpenCL which translates its kernels to vulkan compute shaders. The process unfolded over multiple steps: initial capability testing of clvk on the Pi, benchmarking using Berkley Dwarfs, and analysis of the effectiveness of traditional optimizations.

The ultimate goal proposal was to determine the

efficacy of using the Pi to perform more processing at the edge before sending data to the cloud.

II. Motivation

The current standard is to do data collection at the edge, and send the data to the cloud where all of the analysis is run. A literature review of Industry 4.0 strategies found that edge devices where most commonly used for data collection and that data was sent upstream for processing, machine learning, decision making, etc [4].

The largest issue with this approach is the high latency cost associated with moving data from the edge to the cloud for analysis. For most tasks, this is a good approach as the cloud is far more computationally capable and analysis is expensive. In time sensitive applications, high latency can be unideal. Examples of time sensitive tasks where decision making needs to be near instantaneous are in real time quality control, broad area traffic control, predictive maintenance, autonomous vehicle object detection, and intelligent patient monitoring [18]..

This research was initially inspired by the desire to recreate the work done in the paper titled "ComputeCOVID19+: Accelerating COVID-19 Diagnosis and Monitoring via High-Performance Deep Learning on CT Images" [10]. This paper outlines their work enhancing the resolution of low dosage CT images through their deep learning approach called DenseNet Deconvolution Network, or DDnet. The end goal was to port the OpenCL inference implementation to leverage the GPU of the Raspberry Pi in aid of a mobile application for CT scanning. This has applications such as giving accurate COVID-19 diagnosis within minutes of scanning [10] as well as classifying and flagging mobile CT, X-ray, MRI scans for TB, pneumonia, microcalcifications, and stroke [19].

Several options exist for this task. We chose to use the Raspberry Pi Compute Module 5 for this study. The Raspberry Pi 5 and Compute Module Variant are a line of versatile single board computer that run between 2.5 and 12 watts. Raspberry Pi is the choice for this research for a few reasons. For one, it was chosen for its wide availability and acceptance. The Raspberry Pi 5 did over a million sales in its first 11 months of production [6]. It is a widely available between $70-120 based on the memory capacity. Other options are available such as the NVidia Orin Jetson Nano and its Super variant as well as specialized devices such as the Google Coral TPU. Both of these devices boast more computational power. The Jetson Orin Nano Super, for example, specifically targets AI tasks. It has a 6 core Arm CPU, 1024 CUDA cores, and 32 tensor cores, capable of 67 TOPS at 7-2k watts [20]. However, the NVidia line of products are far more costly and consume more power. The Google TPU, while capable of 2 TOPS, is a specialized device whereas the Pi can do a myriad of tasks. Further, the Raspberry Pi is of interest because of its GPU, which for the most part has seen few use cases outside of display. We are interested in leveraging the compute power of the Pis GPU which has the potential to gain hold of untapped potential.

III. Background

The power of edge devices has improved drastically over the last decade. This has enabled more computation to happen closer to the edge. Comparing the first Raspberry Pi, released in 2012 compared to the Raspberry Pi 5, released in 2023, shows over a 150x improvement in CPU performance and 60x improvement in GPU performance in the last decade. These results where compiled from running Sysbench and GLMark2 on the CPU and GPU respectively.

The Raspberry Pi 5 is powered by the Broadcom BCM2712. This die consists of a 4 core, 4 thread CPU using the ARM architecture. Despite its impressive processing capabilities, the Pi 5 does not include a discrete GPU. Instead, it relies on the integrated Broadcom VideoCore VII, a graphics processor that supports OpenGL ES 3.1 and Vulkan 1.2. This chip is engineered with 12 cores, 128 shader units, and 8 execution units. The VideoCore VII is situated on the CPU die and utilizes dynamically allocated SDRAM, which is not shared with the CPU.

Compared to a discrete GPU, the VideoCore VII is limited by both its operation support and its limited support of common heterogeneous program-

ming languages. Both OpenGL ES and Vulkan are powerful products of the Khronos Group. However, compared to standards like CUDA, OpenACC, and OpenCL, which are standards for parallel programming on heterogeneous systems, OpenGL ES and Vulkan are specifically graphics APIs. Being restricted to the graphics standards can be a difficult challenge when trying to perform general purpose compute on a GPU. As a result, using the GPU for computationally heavy tasks such as neural network inference are not natively supported.

To circumvent this limitation, tools like clvk can be used. clvk is an implementation of OpenCL 3.0 which uses clspv as its compiler. OpenCL 3.0 is a commonly used and portable compute language that can be used on many types of heterogeneous systems. clspv is a open sourced compiler for OpenCL built by Google. It transforms OpenCL C code into SPIR-V modules. SPIR-V modules are the common intermediate binary language that can be consumed by many Khronos Group standards including Vulkan and OpenCL. We plan to implement our code in OpenCL 3.0 standard using clvk and compile it into SPIR-V modules to be interpreted by Vulkan 1.2 which the VideoCore VII supports.

There is little in depth information available about the VideoCore VII architecture. However, the architecture guide for the VideoCore IV, which powers the Raspberry Pi 3 has been released [13]. the VideoCore IV 12 Quad Processors (QPUs). These are floating point shader processors that are 4-way SIMD multiplexed four ways to compute 16-way 32-bit SIMD processing. In other words, the processor executes the same instruction for four cycles on four different 4-way vectors, called 'quads'. The QPUs are organized into groups of four processors called slices. Slices share Instruction cache, Special Function Units, which compute reciprocals, reciprocal square root, logarithm, and exponential operations, as well as one or two Texture and Memory lookup Units. All four QPUs in a slice execute the same set of instructions and resynchronize afterward. The QPUs support packing and unpacking 8 bit unsigned integers, 16 bit signed integers, and 16 bit floats into a single 32 bit value for vector arithmetic.

IV. Approach

Before we are able to commit to rebuilding and optimizing each layer of the DenseNet Deconvolution Network, we need to fully test the efficacy of using the GPU of the Raspberry Pi. The first step in this process was to better understand the capabilities of clvk and the VideoCore VII. All Raspberry Pi tests where performed on a Raspberry Pi 5, 8GB model. clvk and its dependencies where installed and linked as part of each makefile for every executable made during research. The VideoCore VII is a has 32 bit ALUs made septically for graphics processing so it was unclear what operations and data types where supported. Our first task was to create a set of C and OpenCL kernels that would encompass all data types and operations we would commonly encounter in high performance computing. We further gathered and cross referenced the outputs from vulkaninfo and clinfo to check if the results from our tests look consistent with what the underlying system claims to support.

Once we had a baseline of what the the constraints of using clvk on the Raspberry Pi, we needed to benchmark the performance of the GPU. Berkeley Dwarfs are a well known set of 13 benchmarks that are made to evaluate the efficacy of a variety of different processors based on trends on common execution patters [21]. We built benchmarks based on two such Dwarfs: Sparse Matrix Vector Algebra and Dense Matrix Vector Algebra. These two benchmarks where chosen because, out of the 13, Sparse and Dense Matrix Vector Algebra are the most common execution patters as seen in artificial neural network inference. Further, sparse and dense matrix vector algebra represent two different types of memory access patters.

Sparse matrices are ubiquitous across many domains. They are seen in everything from circuit design, to economics. Sparse Matrix Vector Multiplication (SpMV) is a computation defined by random access patterns, making it a bandwidth bound computation [22]. Because sparse matrices are mostly filled with zeros, they can be restructured to save space, reduce computation, and regularize access patters. For general Sparse Matrix Vector

Multiplication, two data structures are commonly seen, compressed sparse row matrices and coordinate matrices. Compressed sparse row matrices hold three arrays. There is an array that holds the value and an array which holds the column index of each non zero element. These arrays are ordered by row in ascending order. The last array holds a pointer to the beginning of each rows non zero elements. This data structure gives fast access to consecutive elements in a row but is particularly inefficient for column major access. Coordinate matrices contain a single array of triplets. Each non zero element explicitly holds the value to its row index, column index, and value. Each non zero element is sorted in the array based on row major access patterns.

To evaluate the performance of the Raspberry Pi, Sparse Matrix Vector Multiplication (SpMV) was implemented using serial code, OpenMP, and OpenCL. The serial and OpenMP versions were developed for both coordinate (COO) and compressed sparse row (CSR) formats. For OpenCL, only the CSR format was implemented. This is because GPUs, particularly on devices like the Raspberry Pi, lack efficient mechanisms for synchronizing writes across threads. The COO format requires reduction of multiple values into the same output vector entry, which is nontrivial to do efficiently without atomics or complex thread coordination. In contrast, CSR enables each thread to process an entire row independently, avoiding race conditions and making it more suitable for GPU acceleration.

In order to test our implementation on a variety of sparse matrices, we collected matrices from the SuiteSparse Matrix Collection, by the University of Florida [23]. We tried to collect a wide variety of sparse graphs that cover the scope of densities, sizes, shapes, and applications. The matrices used are defined in table 1.

The structs and kernels defined in the Virginia Tech OpenDwarfs research where used as a starting point for our work on a sparse matrix vector multiplication implementation [21]. The serial implementation for both coordinate matrices and compressed sparse row where straight forward. The OpenMP implementation of compressed spare row also required little modification. Work was divided by the row. Where each thread would be assigned a set of rows based on chunk size. Because there is no guarantee there will be a consistent number of non zero elements from one row to the next, dynamic scheduling was used. Extra synchronization was required to parallelize the coordinate matrix vector multiplication. Because not all non zero elements within a row are guaranteed to be assigned to the same thread, there is a possible race condition when writing to the resulting vector. To aid in performance and to take advantage of the fact that coordinate vectors are sorted by row major access each thread will keep track of the most recent row it has calculated a sum for and keep a update a local sum variable as long as all subsequent non zero elements are a part of the same row. At a boundary between rows, each thread will atomically update the global resultant vector and reset its local sum. This ensures that each thread only has to do a single atomic write per row it dues calculations for. The OpenCL implementation of SpMV distributed work based on row as well. Each worker was given a row it was responsible. Each worker would locally sum the product of all non zero elements in its row with their associated vector value. A second optimization that was attempted was tiling. Instead of assigning one row per worker, a tile size was defined and each worker would be responsible for that number of consecutive rows. The though process was twofold. This would balance the execution of threads as some rows may be vacant compared to others. Further, for some kinds of graphs in particular diagonal graphs, rows close together access similar vector indices promoting locality.

To implement dense linear algebra benchmarks, we chose to implement level 1, 2, and 3 Basic Linear Algebra Subroutines (BLAS). BLAS are a

TABLE I
SPARSE MATRICES USED IN EVALUATION

Name	Application	Rows	Cols	Nonzeros
barrier2-2	Circuit Simulation	113,076	113,076	3,805,068
webbase-1M	Web Graph	1,000,005	1,000,005	3,105,536
ct20stif	Structural	52,329	52,329	1,375,396
heart2	2D/3D Problem	2,339	2,339	682,797
mac_econ_fwd500	Economics	206,500	206,500	12,733,898

set of vector matrix operations that complete in $O(n), O(n^2),$ and$O(n^3)$ time respectively. Amongst these subroutines we implemented dense dot product, axpy, scale, general matrix vector multiplication, and general matrix matrix multiplication. Due to the regular access patterns of all of these subroutines, they are generally compute bound operations. To test level 1 BLAS operations, we measured throughput. All of these subroutines run in O(n) time and complete very fast. Because of this, we found it was not worth measuring the speedup of a single operation but rather the amount of operations that could be completed in a given time. Dot product, axpy, and scale where implemented serially and in OpenMP. Dot product was not implemented in OpenCL as it requires a reduction and GPUs, especially those as limited as the VideoCore VII, have limited writing synchronization methods. Dot product is such a fast method that the room for synchronization overhead is incredibly small if speedups are to be achieved. For axpy and scale, each thread was assigned a single element of the vector to calculate and the only optimizations attempted where to change the local group size.

Next, general matrix matrix multiplication (GEMM) was attempted. Implementations for GEMM where built upon OpenCL examples from the Khronos Initiative for Training and Education and University of Bristol. A number of kernels with different access structures and work distributions where attempted. The first OpenCL kernel assigned a single element of the resultant vector per thread. The next kernel assigned an entire resultant row to a thread. The next two kernels attempted to use local and private memory of the GPU to store rows and columns of matrix. The last kernel took a blocked approach to amortize column major access of the second matrix. Due to the success of the blocked approach, which will be covered in the results, a blocked approach was used for the general matrix vector multiplication as well. This was compared against an OpenMP and serial implementation of general matrix vector multiplication.

In order to test if GPU optimizations where consistent between the Raspberry Pi and a traditional GPGPU, test results from the Raspberry Pi where cross referenced against results from a system with a GTX 1050.

V. Results

From testing 8GB model of the Raspberry Pi 5, the GPU has access to half of the systems memory. The 4GB model has a global memory size of 2 GB and the 8 GB model has a global memory size of 4 GB. Other limitations have been found using OpenCL to Vulkan. OpenCL is unable to compile kernels with shorts, ushorts, longs, ulongs, doubles, halfs, and trigonometric functions such as sin, cos, tan, etc. These errors are attributed to unsupported data types of the VideoCore VII. Other less explainable errors have occurred during preliminary testing. These include kernels that contain booleans and chars causing the graphics pipeline to hang, preventing all future jobs sent to the GPU to fail to complete. This is something that needs further investigation as the VideoCore VII supports 8 bit operations. Particularly, we would like to look into if it is possible to use clvk and by extension OpenCL to support packing 8 bit values into 32 bit variables.

Within the context of OpenCL, the Raspberry Pi has a few other limitations, these include a small local memory size of 16 KB, having a maximum work item size of $256 \times 256 \times 256$ and having a maximum work group size of 256. Additional constraints of clvk include that each OpenCL context may only have 1 device and that clvk does not support out of order execution.

As for the benchmarks, the results where largely disappointing. Sparse Matrix Vector multiplication performed poorly on the GPU. The average speedup over the serial implementation was 1.29x across the 6 graphs. Further, tiling did not improve performance. It either stayed the same or performed worse. This is likely because each task maps well enough to the hardware, memory accesses between threads in a group are no longer coalesced, and tiling introduces a new branch in the CL kernel. This was consistent between both the VideoCore VII and the GTX 1050. In neither case did tiling improve performance. In any case, the current OpenCL SpMV implementation is less performant on the

Data Type	addition/subtraction	mult/div	mod	exp/exp2/exp10	log/log2/log10	pow	sqrt	sin/cos/tan/asin/acos/atan	floor/ceil/round/trunc/rint	fabs/fmin/fmax	mix/step/smoothstep	native_sin/native_exp/native_log
bool												
char												
uchar												
short												
ushort												
half												
int												
uint												
long												
ulong												
size_t												
float												
double												

TABLE II
PRELIMINARY TESTS OF SUPPORTED OPERATIONS.

Legend:

- **Green** – Operation is supported
- **Red** – Causes program to hang
- **Orange** – Unsupported or does not compile
- **Gray** – Untested

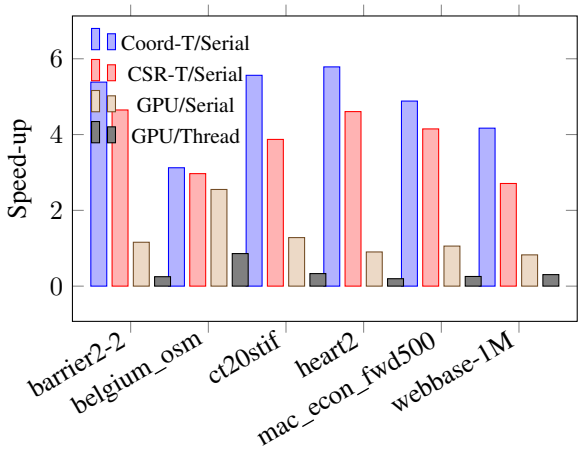


Fig. 1. Relative performance of SpMV variants per matrix (higher is better).

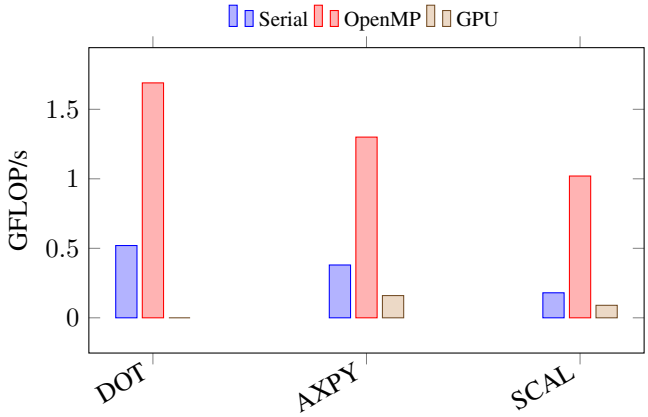


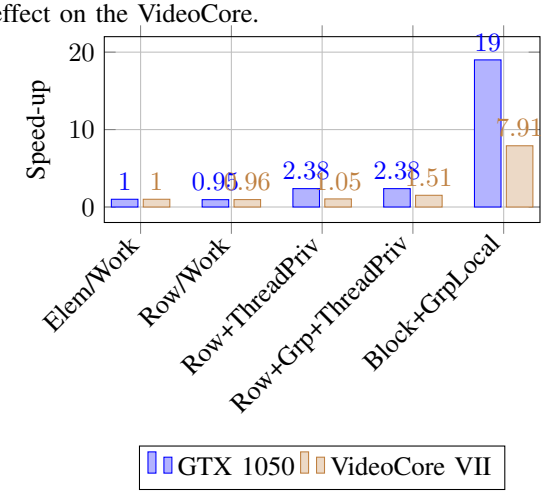
Fig. 2. Throughput of level-1 BLAS kernels on Raspberry Pi 5 (higher is better). GPU *dot* is unimplemented, hence 0.

VideoCore VII than a multithreaded implementation on the Raspberry Pis CPU.

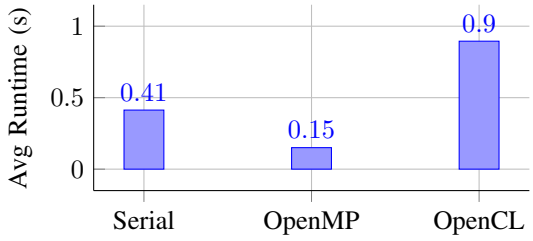
Level 1 BLAS operations performed equally as poor, though the comparison was not in favor of the GPU. The GPU achieved half the throughput as a single core of the CPU. However, additional study needs to be done of the throughput as the current calculation does a two memory allocations per GPU BLAS operation. Whereas there where two CPU memory allocations during the entirety of the serial test. The majority of sequential BLAS operations will be done in place thus having this many OpenCL calls to allocate memory is unrealistic.

The results of GEMM where, however, promising. The most performant result was the block based OpenCL kernel. The block based kernel saw 15.6x speedup, a figure unattainable by the CPU. The GPU achieved a throughput of 3.7 GFLOPS. Unfortunately there are no consistent statistics online, describing the peak floating point performance of the VideoCore VII, otherwise it would be insightful to compare the performance of GEMM to its roofline model as GEMM is a compute bound operation. One interesting thing of note is that the performance gain of various optimizations is not consistent be-

tween the VideoCore VII and other GPGPUs. The majority of optimizations had a substantial effect on the GTX 1050, whereas only blocking had a large effect on the VideoCore.



Based on the success of the blocked approach for general matrix matrix multiplication, we assumed we would see similar improvements for a blocked general matrix vector multiplication approach. This was not the case. The blocking approach for OpenCL GEMV saw a 0.5x speedup by comparison to the serial implementation. The margin for speedup is smaller by comparison to GEMM and a vector is far smaller than a second matrix so it is less able to take advantage of being held in local memory.



A. Related Work

Related to programming for the Raspberry Pis GPU, there is has been little work done. The book "Raspberry Pi GPU Audio Video Programming" by Jan Newmarch looks at writing programs for the GPU through OpenGL, OpenMAX, and OpenVG [11]. It looks at GPU programming through the audio and video points of view rather than programming to the GPU as a general purpose device. There are also a myriad of Papers that look Fast Fourier Transformations (FFTs) utilizing the GPU. These studies utilize GPU_FFT, a program written by Andrew Holme in 2014 [12]. This implementation was written directly in bytecode to utilize the V3D block of the BCM2835, the chip of the Raspberry Pi 1. A spinoff paper also looked at a Vulkan FFT implementation to leverage the GPU [8]. There have been papers which look at Optimizing OpenCL code for the Raspberry Pi. However, these papers look at writing OpenCL code that efficiently maps to the Pi's CPU.

One paper which takes our proposed approach of compiling OpenCL code to Vulkan compute shaders using clvk [14]. Their research was on using heterogeneous systems to reconstruct 3D microwave tomography images from measuring electromagnetic signals. Their paper states their research can benefit from relatively low power portable technology. When running on the Raspberry Pi 4, their algorithm took over twice as long to run on the VideoCore VI as it took to run on the Raspberry Pi's CPU. This aligns with what we saw for the majority of our tests.

This area of research heavily reflects research done in the mid 2000's to bring general purpose compute to graphics hardware. During that time, GPUs where advancing from 8 bit chanel color value pipelines to fully programmable vectorized floating point operation pipelines [15]. GPUs of this time structure their computation into heavily parallelized graphics pipelines consisting of vertex operations, privative assembly, rasterization, fragment operations, and compression into an image. The common languages to interface with the GPU at the time where all based on the idea that the GPU would generate images. Such examples of languages included Cg, HLSL, OpenGL, and Sh. At this time there where early languages and libraries which attempted to map mathematical operations on memory (kernels) to shader streams. Such examples included Accelerator by Microsoft Research, CGiS, and Brook Programming Language. Early papers from this time explored crucial data parallel tasks

and data structures such as map, reduce, scatter and gather, sorting, dense arrays, sparse arrays, and dynamic sparse arrays. Combining these ideas, complex algorithms are explored on GPUs such as differential equations, linear algebra, and data queries.

As seen in the case of the microwave tomography images study, simply having access to heavily parallel hardware does not guarantee an improvement in performance. Since the hardware of the Raspberry Pi is specifically tailored to rendering graphics similar to early GPUs, we hypothesize that modern optimizations may not all work. Looking further into specific optimizations made during the era of early general purpose GPU programming will likely be the best source help when programming for the Pi.

VI. Conclusion

This work presents a novel investigation into harnessing the GPU of the Raspberry Pi 5 for general-purpose computation, particularly in the context of neural network inference at the edge. Through a systematic evaluation of dense and sparse linear algebra workloads, we have demonstrated both the potential and the practical limitations of using the VideoCore VII GPU via OpenCL-to-Vulkan translation. While the GPU performed poorly on bandwidth-bound operations like SpMV and underwhelmed in Level 1 BLAS throughput, it showed promising acceleration in compute-bound workloads such as matrix-matrix multiplication—achieving a 15.6× speedup over the serial baseline. These results highlight that performance gains are highly dependent on memory access patterns, synchronization overhead, and architectural compatibility. Despite the challenges, our findings reaffirm the feasibility of GPU acceleration on low-power edge devices when careful consideration is given to algorithm design and hardware capabilities. This exploration lays the groundwork for more targeted optimizations and future efforts in deploying deep learning workloads directly on embedded GPUs, reducing reliance on the cloud, and enabling more responsive, intelligent edge systems.

References

[1] M. Ghobakhloo, "Industry 4.0, digitization, and opportunities for sustainability," *Journal of Cleaner Production*, vol. 252, p. 119869, Apr. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652619347390

[2] C. M. Endres, C. Pelisser, D. A. Finco, M. S. Silveira, and V. J. Piana, "IoT and Raspberry Pi application in the food industry: a systematic review," *Research, Society and Development*, vol. 11, no. 1, pp. e0 411 124 270–e0 411 124 270, Jan. 2022, number: 1. [Online]. Available: https://rsdjournal.org/index.php/rsd/article/view/24270

[3] M. D. Mudaliar and N. Sivakumar, "IoT based real time energy monitoring system using Raspberry Pi," *Internet of Things*, vol. 12, p. 100292, Dec. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660520301244

[4] I. Henao-Hernández, E. L. Solano-Charris, A. Muñoz-Villamizar, J. Santos, and R. Henríquez-Machado, "Control and monitoring for sustainable manufacturing in the Industry 4.0: A literature review," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 195–200, 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2405896319308778

[5] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, Nov. 2018, publisher: Elsevier. [Online]. Available: https://www.cell.com/heliyon/abstract/S2405-8440(18)33206-7

[6] R. King. Revenue radar: Supply chain disruption halts raspberry pi from further growth. [Online]. Available: https://the-cfo.io/2024/09/26/revenue-radar-supply-chain-disruption-halts-raspberry-pi-from-further-growth/

[7] E. Kadiyala, S. Meda, R. Basani, and S. Muthulakshmi, "Global industrial process monitoring through IoT using Raspberry pi," in *2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2)*, Mar. 2017, pp. 260–262. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8067944

[8] Q. He, V. Weaver, and B. Segee, "Comparing Power and Energy Usage for Scientific Calculation with and without GPU Acceleration on a Raspberry Pi Model B+ and 3B," in *Proceedings on the International Conference on Internet Computing (ICOMP)*. Athens, United States: The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2018, pp. 3–9, num Pages: 3-9. [Online]. Available: https://www.proquest.com/docview/2139478522/abstract/C2EB4E71B3CE455FPQ/1

[9] V. Faerman, V. Avramchuk, K. Voevodin, and M. Shvetsov, "Real-Time Correlation Processing of Vibroacoustic Signals on Single Board Raspberry Pi Computers with HiFiBerry Cards," in *High-Performance Computing Systems and Technologies in Scientific Research, Automation of Control and Production*, V. Jordan, I. Tarasov, and V. Faerman, Eds. Cham: Springer International Publishing, 2022, pp. 55–71.

[10] G. Goel, A. Gondhalekar, J. Qi, Z. Zhang, G. Cao, and W. Feng, "ComputeCOVID19+: Accelerating COVID-19 Diagnosis and Monitoring via High-Performance Deep Learning on CT Images," in *50th International*

Conference on Parallel Processing. Lemont IL USA: ACM, Aug. 2021, pp. 1–11. [Online]. Available: https://dl.acm.org/doi/10.1145/3472456.3473523

[11] J. Newmarch, *Raspberry Pi GPU Audio Video Programming*, 1st ed., ser. Technology in Action. Berkeley, CA: Apress, 2017. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-2472-4

[12] A. Holme. (2014) GPU_FFT: Fast fourier transform library for the raspberry pi. [Online]. Available: http://www.aholme.co.uk/GPU_FFT/Main.htm

[13] Broadcom Corporation, *VideoCore IV 3D Architecture Reference Guide*, June 2013, revision 1.0. [Online]. Available: https://docs.broadcom.com/doc/12358545

[14] T. Vasileiou and P. Kosmas, "Accelerated Medical Microwave Tomography via Heterogeneous Computing," *TechRxiv*, 2020. [Online]. Available: https://www.authorea.com/doi/full/10.36227/techrxiv.171198045.57783972?commit=7dde951108f896bad012b52dac17955e867249c0

[15] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80–113, 2007, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2007.01012.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01012.x

[16] W.-c. Feng, H. Lin, T. Scogland, and J. Zhang, "OpenCL and the 13 dwarfs: a work in progress," in *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*. Boston Massachusetts USA: ACM, Apr. 2012, pp. 291–294. [Online]. Available: https://dl.acm.org/doi/10.1145/2188286.2188341

[17] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, and S. W. Williams, "The Landscape of Parallel Computing Research: A View from Berkeley," Dec. 2006. [Online]. Available: https://escholarship.org/uc/item/1z50m2xt

[18] C. Swetha, "Edge AI for real-time decision making in IOT networks," *International Journal of Innovative Research*, vol. 12, no. 9, pp. 11 293–11 309, Sep. 2024. [Online]. Available: https://www.researchgate.net/publication/387321332_Edge_AI_for_Real-Time_Decision_Making_in_IOT_Networks

[19] A. Kharat, V. A. Duddalwar, K. Saoji, A. Gaikwad, V. Kulkarni, G. Naik, R. Lokwani, S. Kasliwal, S. Kondal, T. Gupte, and A. Pant, "Role of edge device and cloud machine learning in point-of-care solutions using imaging diagnostics for population screening," *CoRR*, vol. abs/2006.13808, 2020. [Online]. Available: https://arxiv.org/abs/2006.13808

[20] NVIDIA Corporation, "Jetson orin nano developer kit datasheet," NVIDIA Corporation, Tech. Rep. 3575392-R2, 2023, accessed: 2025-04-11. [Online]. Available: https://nvdam.widen.net/s/zkfqjmtds2/jetson-orin-datasheet-nano-developer-kit-3575392-r2

[21] K. Krommydas, W.-c. Feng, C. D. Antonopoulos, and N. Bellas, "Opendwarfs: Characterization of dwarf-based benchmarks on fixed and reconfigurable architectures," *Journal of Signal Processing Systems*, vol. 85, pp. 373–392, 2016.

[22] S. Filippone, V. Cardellini, D. Barbieri, and A. Fanfarillo, "Sparse matrix-vector multiplication on gpgpus," *ACM Transactions on Mathematical Software (TOMS)*, vol. 43, no. 4, pp. 1–49, 2017.

[23] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[24] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.