International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/IJSRCSEIT

OpenCL Performance Evaluation on Multiple Operating Systems

Rushikesh Vidye[1], Praveen Kulkarni[2], Rutik Wagh[3], Prof. Pravin Patil[4]

*1,2,3 Student, Department of Computer SCTR's Pune Institute of Computer Technology, Pune, Maharashtra, India

4 Assistant Professor, Department of Computer Engineering, SCTR's Pune Institute of Computer Technology, Pune, Maharashtra, India

ARTICLEINFO

Article History:

Accepted: 13 March 2023
Published: 18 March 2023

Publication Issue
Volume 10, Issue 2
March-April-2023

Page Number
145-152

ABSTRACT

OpenCL is an open standard for parallel computing that enables performance portability across diverse computing platforms. In this work, we perform a systematic evaluation of OpenCL performance on several Operating System Platforms including Windows, Linux, Android and macOS. Our results provide insights into the impact of the Operating Systems on OpenCL performance and identify any potential performance bottlenecks. We also compare performance of OpenCL with other parallel computing frameworks like Nvidia's CUDA (Compute Unified Device Architecture), Apple's Metal framework, DirectX Compute etc. on different operating systems to better understand the trade-offs between different OSs. Our findings can help researchers and practitioners make informed decision about choosing the appropriate Operating System for their OpenCL applications and guide future development of OpenCL standard.

Keywords: OpenCL, CUDA, Metal, GPGPU, MacOS, Linux, Windows, Android.

I. INTRODUCTION

The rapid development of computing ability of consumer grade hardware, especially in area of using Graphics Processing Units (GPU's) for general purpose computing using OpenCL, CUDA, ARM Compute, Metal has rendered today's enthusiast PC at or near the level of the super computers of late 90s. Nowadays, GPUs are used not only for graphics applications, but also non-graphic applications, so-called GPU computing or GPGPU. Thanks to their high floating-point operation rates and memory bandwidths, GPU can accelerate various science and engineering computations. [1-6].

Owning to this tremendous performance perspective, GPU programming models have transformed from high-level languages such as HLSL, CG and GLSL to recent programming languages, which has successively increased programmers load and thus enhanced GPUs acceptance. The launch of CUDA by NVIDIA in 2007 has diminished the use of graphical APIs for computational activities, resulting in wide-spread use of GPU computing. Similarly, a programming framework known as APP (Advanced Parallel Processing) that is known to enable ATIs GPUs in concurrence with CPUs speed up a number of requests. These agendas in terms have enabled the programmers to cultivate GPU computing applications without much knowledge on graphics.[7] OpenCL is a new programming standard for various compute devices. OpenCL is used to develop code not only for GPUs, but also for multi-core CPUs, Cell Broadband Engines, and other compute devices; Thus, it enabled programmers to avoid writing a vendor-specific code, resulting in improved code portability. However, although OpenCL allows a programmer to use various compute devices, efficient coding and optimization methodologies for individual compute device are not established yet. While both CUDA and ATI Stream have intensively been optimized to exploit the computing power of their own GPUs, there is a semantic gap between OpenCL and compute devices because OpenCL is vendor-independent and hence not specialized for any compute device. The OpenCL C compiler plays important roles to exploit the potential of compute devices and therefore its capabilities should be clarified. Operating System is also an important aspect that affects performance as different Oss prefer different way to access hardware drivers, a lot of times these drivers abstracted from programmer with some bridging api's like DirectX Compute for Windows, Metal For macOS, and Mesa Interface for Linux.

In this paper, we systematically compare the sustained performances of OpenCL programs with those of CUDA programs. To make a fair comparison, CUDA and DirectX Compute codes are ported to OpenCL codes as faithfully as possible. Based on the performance comparison, this paper analyses and discusses the main factors of causing the performance between different OSs. Since OpenCL is supported by all OSs a fair comparison of their performance can be performed.

II. Similarities Between OpenCL CUDA and DirectX Compute

This section described the different frameworks that are a subject of comparison in this comparative study. Each sections contains sample code for vector addition kernel for respective framework. The popularity of frameworks is based upon the chart in figure 2.1 from google trends.

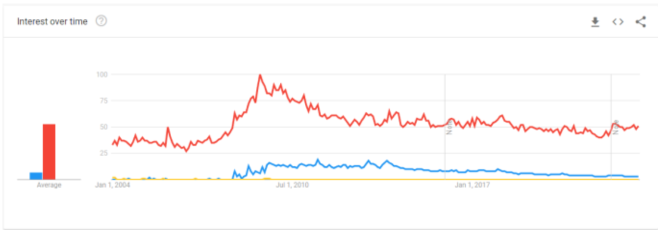


Fig 2.1 Popularity over time of CUDA, OpenCL, Direct Compute. The numbers represent search interest relative to the highest point on the chart for given region and time.

A. CUDA

Released in 2—7, CUDA developed by Nvidia was the first major GPGPU framework to be released. It aims to make the parallelization of a problem more manageable by providing an easy to work with API.[8]

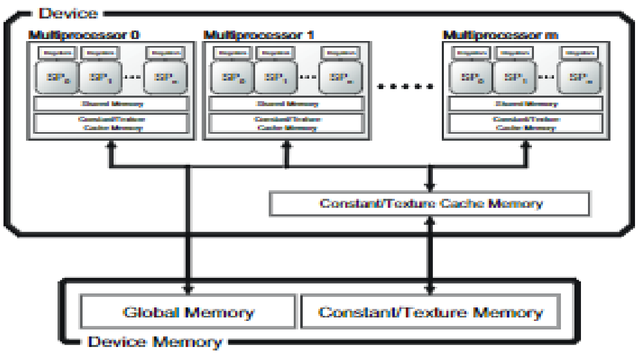


Fig 2.1 GPU architecture in CUDA.

B. OpenCL

In 2009, OpenCL was announced as an open programming standard to access GPU and other compute device in a unified manner. The specifications and programming languages of OpenCL and CUDA have similarities in many aspects. Therefore, CUDA programmers can harness their experiences and skills to write an efficient OpenCL Program.[9-10]

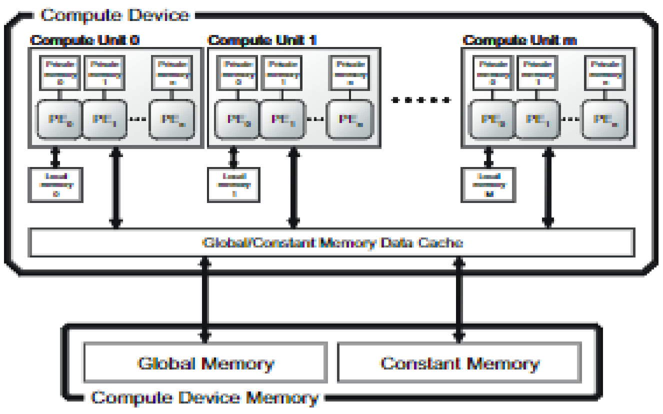


Fig 2.2 GPU architecture in OpenCL.

C. Direct Compute

Since Direct Compute is not a framework, but a API of the DirectX suite and uses the concept of CS to perform GPGPU calculations, it is quite different from CUDA or OpenCL. All the computing in Direct Compute is done in CS (Compute Shader) which is equivalent to a kernel in CUDA or OpenCL. The setup process is quite similar to programable pipeline in OpenGL or Vulkun but uses traditional graphics way of copying data from host and device using buffers, which are copied to CS before CS is run. The CS is written in HLSL (High Level Shading Language) also developed by Microsoft.[11]

Memory Accessibility	**CUDA**	**OepnCL**	**Direct Compute**	**Readable/Writeable**
Thread/ work-item	Local memory	Private memory	Private memory	R / W
CTA / Work-Group	Shared Memory	Local Memory	Local Memory	R / W
Grid / NDRange	Constant Memory	Constant Memory	Constant Buffer	Readable
	Texture Memory	-	Texture Memory	Readable

Fig 2.3 Memory Hierarchy.

III. Implementation

A common problem where GPGPU is used is when computing calculations heavy like vector algebra or Matrix Multiplications. Other common visualizations where GPGPU can be applied is to visualize fractals such a Julia or Mandelbrot set, named after the French mathematician Gaston Julia and Benoit Mandelbrot. GPGPU has also used in medicine for accelerated medical image reconstruction.

This section described the implementation of the vector addition and matrix multiplication algorithms in all discussed frameworks, as well as how the measurements were performed and what OS specific features were used. All implementation was implementation was implemented in C.

A. Vector Addition Algorithm

- Allocate memory to host buffers.
- Copy memory from host to gpu.
- Compile & Build Kernel Program.
- Set kernel arguments.
- Execute Kernel.
- Verify the results.

A.A.1 Sample Kernel for Vector Addition

```
__kernel void vec_add(__global const float* a,
__global const float* b, __global float* result)
{
        int gid = get_global_id(0);
        result[gid] = a[gid] + b[gid];
}
```

B. Matrix Multiplication Algorithm

- Init Matrix A and B.
- Copy matrices to GPU.
- Compile & Build Kernel Program.
- Set kernel arguments.
- Execute Kernel.
- Verify the results.

B.A.1 Sample Kernel for Matrix Multiplication

```
__kernel void matMulGPU(__global int *A,
__global int* B, __global int* C,int numARows, int
numAColumns, int numBColumns, int
numCColumns)
{
        int row = get_global_id(0);
        int column = get_global_id(1);
        if((row < numARows) && (column <
numBColumns))
        {
int value = 0;
for(int k = 0; k < numAColumns; k++)
{
int a = A[row * numAColumns + k];
int b = B[k * numBColumns + column];
        value += a * b;
        }
        C[row * numCColumns + column] = value;
        }
}
```

C. Steps For Building OpenCL Program Windows

1) Install latest version of MSVC and OpenCL libraries.

2) Use following command:

```
cl.exe *.cpp /c /EHSc
link.exe *.obj user32.lib gdi32.lib kernel32.lib
OpenCL32.lib
```

You can execute the executable now.

D. Steps For Building OpenCL Program Linux

1) Install latest version of g++ and OpenCL libraries.

2) Use following command:

```
g++ *.cpp -lOpenCL
```

You can execute the executable now.

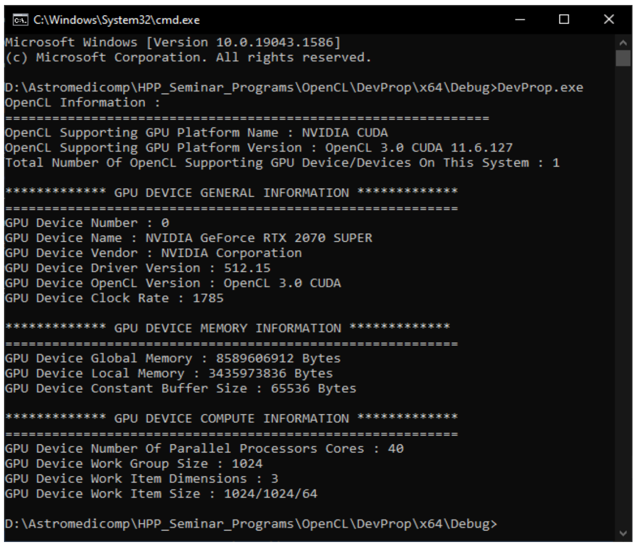
E. Steps For Building OpenCL Program MacOS

1) Install latest version of XCode.

2) Use following command:

```
Clang++ *.mm -framework OpenCL
```

You can execute the executable now.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. All rights reserved.

D:\Astromedicomp\HPP_Seminar_Programs\OpenCL\DevProp\x64\Debug>DevProp.exe
OpenCL Information :
==============================================================
OpenCL Supporting GPU Platform Name : NVIDIA CUDA
OpenCL Supporting GPU Platform Version : OpenCL 3.0 CUDA 11.6.127
Total Number Of OpenCL Supporting GPU Device/Devices On This System : 1

************* GPU DEVICE GENERAL INFORMATION *************
==========================================================
GPU Device Number : 0
GPU Device Name : NVIDIA GeForce RTX 2070 SUPER
GPU Device Vendor : NVIDIA Corporation
GPU Device Driver Version : 512.15
GPU Device OpenCL Version : OpenCL 3.0 CUDA
GPU Device Clock Rate : 1785

************* GPU DEVICE MEMORY INFORMATION *************
==========================================================
GPU Device Global Memory : 8589606912 Bytes
GPU Device Local Memory : 3435973836 Bytes
GPU Device Constant Buffer Size : 65536 Bytes

************* GPU DEVICE COMPUTE INFORMATION *************
==========================================================
GPU Device Number Of Parallel Processors Cores : 40
GPU Device Work Group Size : 1024
GPU Device Work Item Dimensions : 3
GPU Device Work Item Size : 1024/1024/64

D:\Astromedicomp\HPP_Seminar_Programs\OpenCL\DevProp\x64\Debug>
```

F. Steps For Building OpenCL Program Android

1) Install latest version of Android Studio.

2) Create Native App with C++ class.

3) Create Java JNI code to interact with C++ Lib.

4) Add all required libraries to gradle.

5) Build and deploy app on android device.

IV. Results

This section shows the evaluation results to clarify the difference in sustained performance between OpenCL programs on different OSs.

In this evaluation two programs were selected Vector Addition and Matrix Multiplication. PC / Mobile specification is given in below table 5.1 with Oss build versions.

1) Hardware Configuration

CPU	AMD Ryzen 7 3700X
GPU	NVIDIA RTX 2070 Super
Chipset	x64
RAM	32 GB
VRAM	8 GB
OS	Windows 10, Manjaro Linux, MacOS 11
Platform	Nvidia CUDA Toolkit 11.2
Driver	Nvidia Driver Version: 418.25 Nvidia Mesa Driver: For Linux Nvidia Apple INC Driver

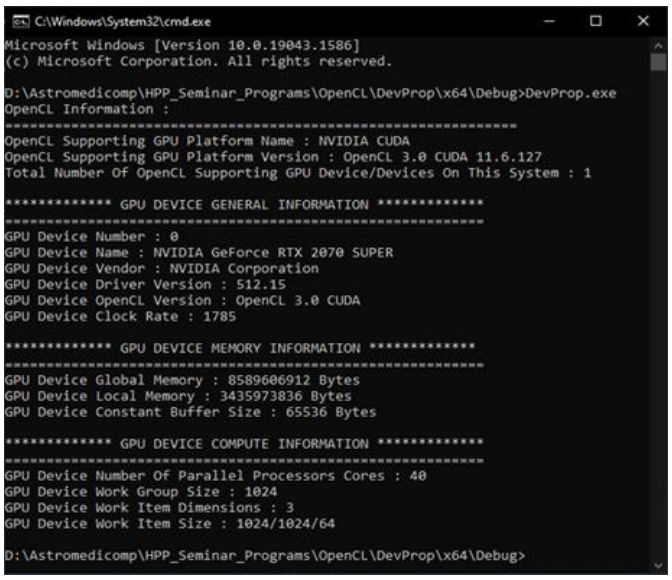
Table 4.1: PC Specification

Model	Xiaomi M2101K7BI
CPU	ARM64-v8a
GPU	Mali G-76
Chipset	ARM
RAM	8 GB
OpenCL Version	OpenCL FULL_PROFILE 1.2
OS	Android 12
Platform	ARM

Table 4.2: Mobile Specification

2) Output of DevProp on All OSs.

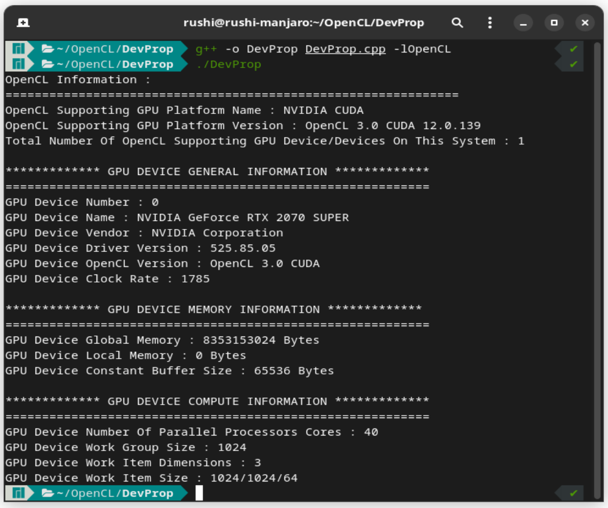
2.1) Windows



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. All rights reserved.

D:\Astromedicomp\HPP_Seminar_Programs\OpenCL\DevProp\x64\Debug>DevProp.exe
OpenCL Information :
=================================================================
OpenCL Supporting GPU Platform Name : NVIDIA CUDA
OpenCL Supporting GPU Platform Version : OpenCL 3.0 CUDA 11.6.127
Total Number Of OpenCL Supporting GPU Device/Devices On This System : 1

************* GPU DEVICE GENERAL INFORMATION *************
==========================================================
GPU Device Number : 0
GPU Device Name : NVIDIA GeForce RTX 2070 SUPER
GPU Device Vendor : NVIDIA Corporation
GPU Device Driver Version : 512.15
GPU Device OpenCL Version : OpenCL 3.0 CUDA
GPU Device Clock Rate : 1785

************* GPU DEVICE MEMORY INFORMATION *************
==========================================================
GPU Device Global Memory : 8589606912 Bytes
GPU Device Local Memory : 3435973836 Bytes
GPU Device Constant Buffer Size : 65536 Bytes

************* GPU DEVICE COMPUTE INFORMATION *************
==========================================================
GPU Device Number Of Parallel Processors Cores : 40
GPU Device Work Group Size : 1024
GPU Device Work Item Dimensions : 3
GPU Device Work Item Size : 1024/1024/64

D:\Astromedicomp\HPP_Seminar_Programs\OpenCL\DevProp\x64\Debug>
```

Fig 4.1: DevProp On Windows

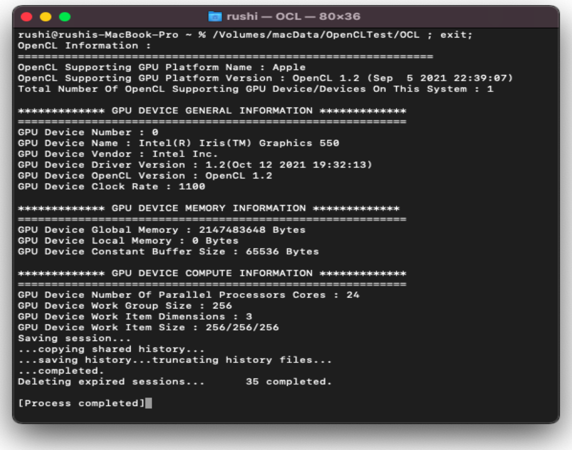
2.2) Linux



```
rushi@rushi-manjaro:~/OpenCL/DevProp
~/OpenCL/DevProp  g++ -o DevProp DevProp.cpp -lOpenCL
~/OpenCL/DevProp  ./DevProp
OpenCL Information :
==============================================================
OpenCL Supporting GPU Platform Name : NVIDIA CUDA
OpenCL Supporting GPU Platform Version : OpenCL 3.0 CUDA 12.0.139
Total Number Of OpenCL Supporting GPU Device/Devices On This System : 1

************* GPU DEVICE GENERAL INFORMATION *************
==========================================================
GPU Device Number : 0
GPU Device Name : NVIDIA GeForce RTX 2070 SUPER
GPU Device Vendor : NVIDIA Corporation
GPU Device Driver Version : 525.85.05
GPU Device OpenCL Version : OpenCL 3.0 CUDA
GPU Device Clock Rate : 1785

************* GPU DEVICE MEMORY INFORMATION *************
==========================================================
GPU Device Global Memory : 8353153024 Bytes
GPU Device Local Memory : 0 Bytes
GPU Device Constant Buffer Size : 65536 Bytes

************* GPU DEVICE COMPUTE INFORMATION *************
==========================================================
GPU Device Number Of Parallel Processors Cores : 40
GPU Device Work Group Size : 1024
GPU Device Work Item Dimensions : 3
GPU Device Work Item Size : 1024/1024/64
~/OpenCL/DevProp
```

Fig 4.2: DevProp on Linux

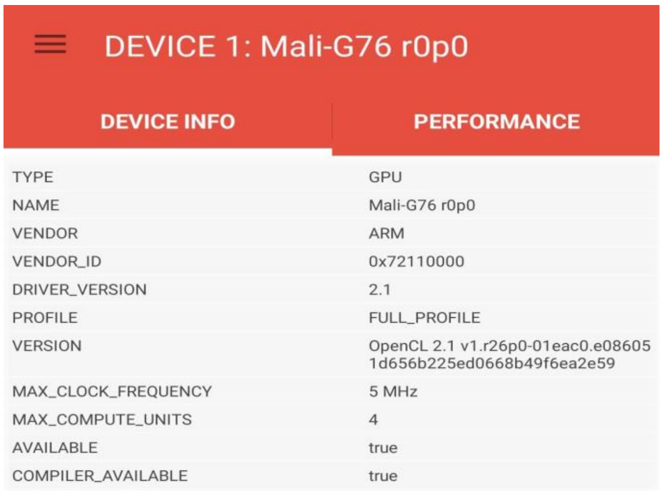
2.3) MacOs



```
rushi@rushis-MacBook-Pro ~ % /Volumes/macData/OpenCLTest/OCL ; exit;
OpenCL Information :
==============================================================
OpenCL Supporting GPU Platform Name : Apple
OpenCL Supporting GPU Platform Version : OpenCL 1.2 (Sep  5 2021 22:39:07)
Total Number Of OpenCL Supporting GPU Device/Devices On This System : 1

************* GPU DEVICE GENERAL INFORMATION *************
==========================================================
GPU Device Number : 0
GPU Device Name : Intel(R) Iris(TM) Graphics 550
GPU Device Vendor : Intel Inc.
GPU Device Driver Version : 1.2(Oct 12 2021 19:32:13)
GPU Device OpenCL Version : OpenCL 1.2
GPU Device Clock Rate : 1100

************* GPU DEVICE MEMORY INFORMATION *************
==========================================================
GPU Device Global Memory : 2147483648 Bytes
GPU Device Local Memory : 0 Bytes
GPU Device Constant Buffer Size : 65536 Bytes

************* GPU DEVICE COMPUTE INFORMATION *************
==========================================================
GPU Device Number Of Parallel Processors Cores : 24
GPU Device Work Group Size : 256
GPU Device Work Item Dimensions : 3
GPU Device Work Item Size : 256/256/256
Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.
Deleting expired sessions...      35 completed.

[Process completed]
```

Fig 4.3 DevProp On MacOS

2.4) Android



DEVICE 1: Mali-G76 r0p0

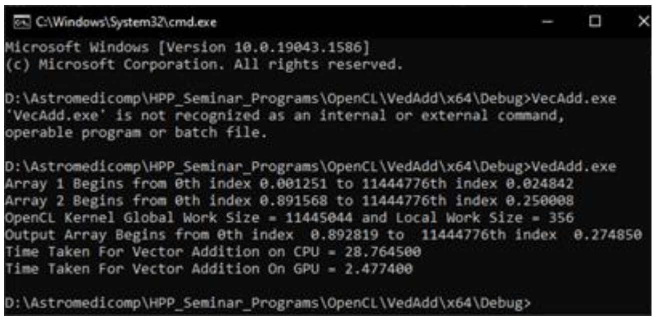
DEVICE INFO | PERFORMANCE

TYPE	GPU
NAME	Mali-G76 r0p0
VENDOR	ARM
VENDOR_ID	0x72110000
DRIVER_VERSION	2.1
PROFILE	FULL_PROFILE
VERSION	OpenCL 2.1 v1.r26p0-01eac0.e08605 1d656b225ed0668b49f6ea2e59
MAX_CLOCK_FREQUENCY	5 MHz
MAX_COMPUTE_UNITS	4
AVAILABLE	true
COMPILER_AVAILABLE	true

Fig 4.4 : DevProp On Android using OpenCL-Z

3) Output of VecAdd on All OSs.

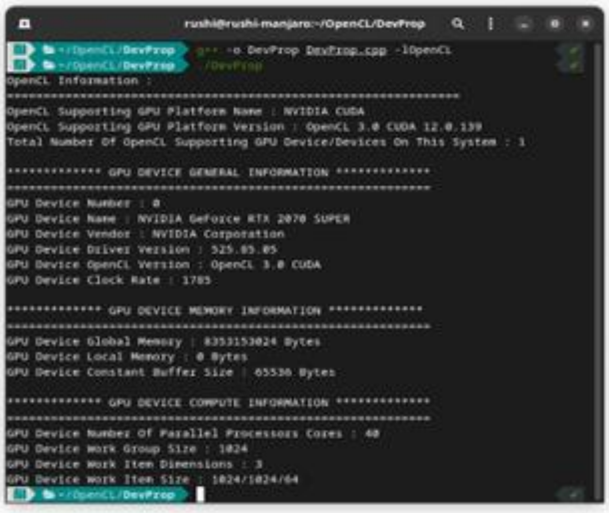
3.1)Windows



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19043.1586]
(c) Microsoft Corporation. All rights reserved.

D:\Astromedicomp\HPP_Seminar_Programs\OpenCL\VedAdd\x64\Debug>VecAdd.exe
'VecAdd.exe' is not recognized as an internal or external command,
operable program or batch file.

D:\Astromedicomp\HPP_Seminar_Programs\OpenCL\VedAdd\x64\Debug>VedAdd.exe
Array 1 Begins from 0th index 0.001251 to 11444776th index 0.024842
Array 2 Begins from 0th index 0.891568 to 11444776th index 0.250008
OpenCL Kernel Global Work Size = 11445044 and Local Work Size = 356
Output Array Begins from 0th index  0.892819 to  11444776th index  0.274850
Time Taken For Vector Addition on CPU = 28.764500
Time Taken For Vector Addition On GPU = 2.477400

D:\Astromedicomp\HPP_Seminar_Programs\OpenCL\VedAdd\x64\Debug>
```

Fig 4.5: VecAdd On Windows

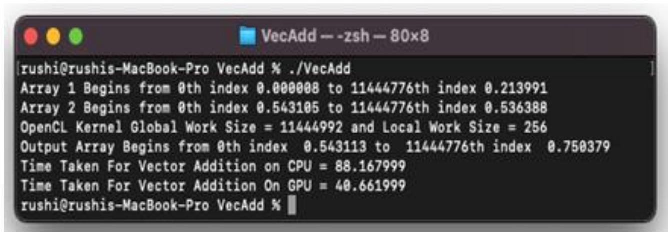
3.2) Linux



```
rushi@rushi-manjaro:~/OpenCL/DevProp
~/OpenCL/DevProp  g++ -o DevProp DevProp.cpp -lOpenCL
~/OpenCL/DevProp  ./DevProp
OpenCL Information :
==============================================================
OpenCL Supporting GPU Platform Name : NVIDIA CUDA
OpenCL Supporting GPU Platform Version : OpenCL 3.0 CUDA 12.0.139
Total Number Of OpenCL Supporting GPU Device/Devices On This System : 1

************* GPU DEVICE GENERAL INFORMATION *************
==========================================================
GPU Device Number : 0
GPU Device Name : NVIDIA GeForce RTX 2070 SUPER
GPU Device Vendor : NVIDIA Corporation
GPU Device Driver Version : 525.85.05
GPU Device OpenCL Version : OpenCL 3.0 CUDA
GPU Device Clock Rate : 1785

************* GPU DEVICE MEMORY INFORMATION *************
==========================================================
GPU Device Global Memory : 8353153024 Bytes
GPU Device Local Memory : 0 Bytes
GPU Device Constant Buffer Size : 65536 Bytes

************* GPU DEVICE COMPUTE INFORMATION *************
==========================================================
GPU Device Number Of Parallel Processors Cores : 40
GPU Device Work Group Size : 1024
GPU Device Work Item Dimensions : 3
GPU Device Work Item Size : 1024/1024/64
~/OpenCL/DevProp
```

Fig 4.6: VecAdd on Linux

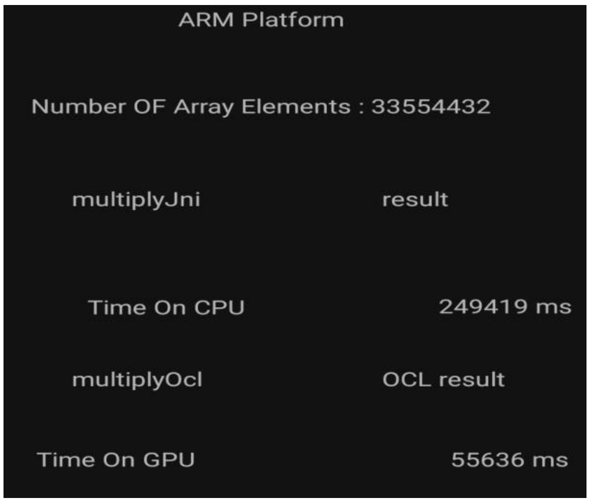
3.3) MacOs



```
VecAdd — -zsh — 80x8
rushi@rushis-MacBook-Pro VecAdd % ./VecAdd
Array 1 Begins from 0th index 0.000008 to 11444776th index 0.213991
Array 2 Begins from 0th index 0.543105 to 11444776th index 0.536388
OpenCL Kernel Global Work Size = 11444992 and Local Work Size = 256
Output Array Begins from 0th index  0.543113 to  11444776th index  0.750379
Time Taken For Vector Addition on CPU = 88.167999
Time Taken For Vector Addition On GPU = 40.661999
rushi@rushis-MacBook-Pro VecAdd %
```

Fig 4.7: VecAdd on MacOS 11

3.4) Android



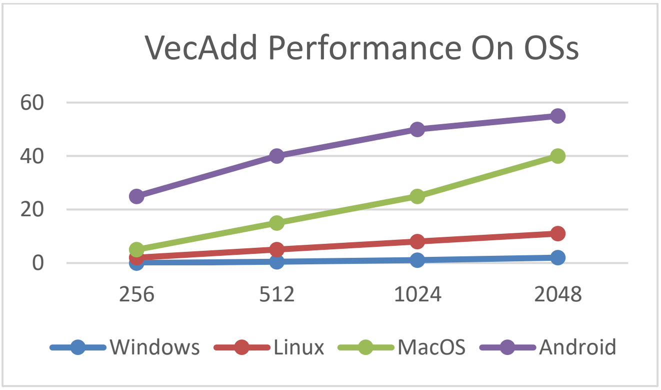
```
ARM Platform

Number OF Array Elements : 33554432

multiplyJni                result

Time On CPU                249419 ms

multiplyOcl                OCL result

Time On GPU                55636 ms
```

Fig 4.8 : VecAdd On Android

Above outputs demonstrate the performance difference of same OpenCL kernel on different OSs. With Same hardware.

V. Summary

Following chart shows the performance difference for same OpenCL kernel. Difference between Windows and MacOS is quite noticeable when we consider that Windows uses proprietary Nvidia Drivers and MacOS uses Apples own driver for same graphic card. This proves even though vendor-independent OpenCL still relies on optimization at driver level.
For android performance manly relies on the ARM OpenCL driver supported by vendor, for this device OpenCL version was locked on 1.2 therefore we can see it's performance is quite slow compared to other versions with OpenCL version 3.0.[12]
Since neither OpenCL or Direct Compute supports classes or C++ like data structures inside kernel, for applications using larger more complex kernels the ability to be able to write objects would be an important feature. [13]



VI. CONCLUSION

This paper has discussed the sustained performance of OpenCL programs in same hardware environment on different OSs. The quantitative evaluation results indicate that the sustained performance of every OpenCL program is different for each of the targeted OSs. To clarify the reasons the performance differences between all OSs, this paper also analysis, and pointed out that the performance difference comes from the difference in the compiler optimization capabilities of each OS. GPU driver also plays very important role in OpenCL performance as we have seen the performance gap between Proprietary Drivers, Open-Source Drivers, and Software Drivers. A challenge for future research is to develop automatic performance tuning methodology based on profiling to enhance the performance and portability of OpenCL applications.

VII. REFERENCES

[1]. Kazuhiko Komatsu, Katsuto Sato, Yusuke Arai, Kentaro Koyama, Hiroyuki Takizawa, and Hiroaki Kobayashi, "Evaluating Performance and Portability of OpenCL Programs"2010 Publication ID : 228868467

[2]. John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips. GPU computing. Proceedings of the IEEE, 96(5):879–899, May2008

[3]. N.K. Govindaraju, S. Larsen, J. Gray, and D. Manocha. A memory model for scientific algorithms on graphics processors. In the 2006 ACM/IEEE conference on Supercomputing (SC06), November 2006.

[4]. Ian Buck et al. Gpu bench: Evaluating gpu performance for numerical and scientic applications. In2004 ACM Workshop on General Purpose Computing on Graphics Processors, pages C–20, 2004.

[5]. S. Che, J. Meng, J. Sheaffer, and K. Skadron. A performance study of general purpose applications on graphics processors. In The First Workshop on General Purpose Processing on Graphics Processing Units, 2007.

[6]. Wen-Mei W. Hwu, Christopher Rodrigues, Shane Ryoo, and John Stratton. Com-pute unified device architecture application

suitability. Computing in Science and Engineering, 11(3):16–26, 2009.

[7]. Hiroyuki Takizawa and Hiroaki Kobayashi. Hierarchical parallel processing of largescale data clustering on a pc cluster with gpu co-processing. The Journal of Super-computing, 38(3):219–234, 2006.

[8]. NVIDIA Corporation.NVIDIA CUDA Compute Unified Device Architecture pro-gramming guide 3.0, 2010.http://developer.nvidia.com/object/cuda.html.

[9]. The Khronos OpenCL Working Group.The OepnCL Specification version 1.0,2008.http://www.khronos.org/oepncl/.

[10]. Dave Shreiner and The Khronos OpenGL ARB Working Group. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 3.0 and 3.1. Addison-Wesley Professional, 7th edition, 2009.

[11]. Microsoft Corporation. DirectX. http://www.microsoft.com/windows/directx/.

[12]. NVIDIA Corporation. PTX : Parallel Tread Execution ISA Version 1.4, 2009.

[13]. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. Compilers: Principles, Techniques, & Tools. Addison Wesley, 2nd edition, 2007.View publication stats

Cite this article as :

Rushikesh Vidye, Praveen Kulkarni, Rutik Wagh, Prof. Pravin Patil, "OpenCL Performance Evaluation on Multiple Operating Systems", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 9, Issue 2, pp.145-152, March-April-2023.