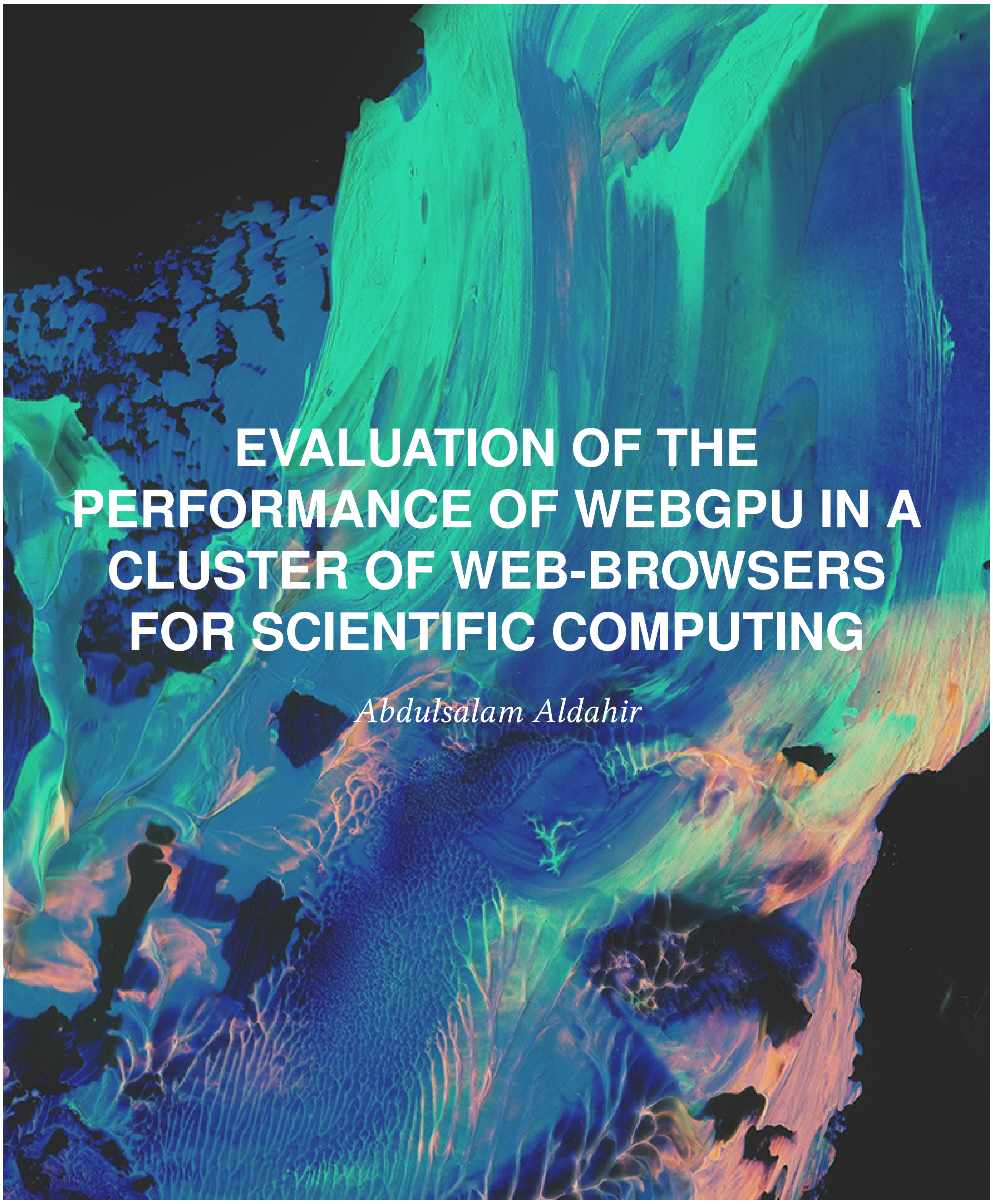
UMEÅ UNIVERSITY



EVALUATION OF THE PERFORMANCE OF WEBGPU IN A CLUSTER OF WEB-BROWSERS FOR SCIENTIFIC COMPUTING

Abdulsalam Aldahir

Bachelor Thesis, 15 hp/credits
Bachelor Of Science Programme in Computing Science

2022

Abstract

The development and widespread of Internet browsers and technologies make them a tool that can be used for many scientific problems. This raises the question of whether Internet browsers, together with WebGPU and WebRTC, can be used to do scalable computing in a distributed cluster. This thesis answers the question by implementing a peer-to-peer cluster and testing it with two problems, Matrix multiplication and Mandelbrot sets generation. The experimental results show that computing embarrassingly parallel problems are scalable with more than 75% efficiency.

Acknowledgements

I am extremely grateful to my parents for all the support.

Many thanks to Mustafa Aldaher and Mohammed Msheleh for their comments on the thesis.

Contents

1 Introduction

Science domains are generating a massive amount of data that sometimes exceeds the computational power available in today's computers, making it difficult to handle and process. For some problems, it is common to use a Graphics Processing Unit (GPU), which is a processor initially designed for graphics rendering. Over the past 15 years, GPUs got a noticeable increase in performance and have become used widely as an accelerator for general-purpose high-performance computing, which is usually called general-purpose GPU (GPGPU).

Another way of doing high-performance computing is to use supercomputers or a cluster of computers and GPUs. Recently, the *World Wide Web Consortium (W3C)* started developing a new Application Programming Interface (API) called *WebGPU* to be the new standard for GPU access on the web. Given that it allows GPGPU raises the question of whether it would be scalable to use it on a cluster of web browsers, using WebRTC for communication.

1.1 Motivation

Today's computers, web browsers, and modern GPUs are available almost everywhere. Building a cluster of web browsers could, for example, enable people to donate their computer power to science through a simple web page. Given that there are more than 3.2 billion users of web-browsers, this could turn the internet into a distributed computer for research purposes, and would significantly impact the development of many scientific fields, such as Machine learning [4] which depends on GPUs and parallel programming.

Access to a reliable GPU API from the browser is also essential for the limited-scale problems. It also would be necessary for computer science classes that teach GPU and parallel programming, which need to have a consistent, scalable, and friendly development infrastructure to rely on [5].

Dedicated GPUs would make building GPU clusters on web browsers easier. Usually, GPU APIs such as Compute Unified Device Architecture (CUDA) and even WebGL make researchers ambivalent of using them or building complex supercomputers/clusters to do time-intensive computing. Because they are often cumbersome and platform dependent [9] which also requires time and some technical skills. WebGPU tries to do such abstraction to solve such problems by providing direct access to the GPU hardware making GPU access more portable.

1.2 Research Questions

This report investigates if WebGPU has less run-time than CUDA on a single machine, and tries to answer if a cluster of web browsers, using WebGPU and WebRTC is scalable or not. More specifically, does the speedup increase as more nodes(web-browsers) join the cluster?

1.3 Thesis Outline

The rest of the thesis is as follows, Chapter 2 talks about some related works on the subject and some similar libraries and APIs. Then, Chapter 3 explains some theories about the algorithms and the tools that were used. In Chapters 4 the method and the experiments for the evaluation are presented. At the end, in Chapters 5 and 6 the results and a discussion is presented.

2 Related Work

Several studies investigated if web browsers can be used in distributed clusters. R. Cushing et al. presented the paper *Distributed Computing on an Ensemble of Browsers* [1] about using web-browsers as cluster to compute thousands of bio-informatics tasks. In their results, they demonstrate that computing on browsers is possible, although it is less efficient than native solutions, but it is a trade-off to get portability across all platforms.

In their paper *JSDoop and TensorFlow.js: Volunteer Distributed Web Browser-Based Neural Network Training* [7] J. Á. Morell et al. presented a distributed neural network on web browsers. They tested the cluster with up to 32 volunteers, and their results show that having such a system feasible with high scalability. Their implementation was based on WebGL, and it uses the STOMP protocol for communications.

T. Koskela and some others presented a *RADE: Resource-aware distributed browser-to-browser 3D graphics delivery in the web* [6] that evaluate to evaluate the load, response time, and cost of service of 3D asset delivery in web-browsers. They used WebRTC in almost the same way used in this thesis for communication.

Since WebGPU is still in the working draft, there is almost no research on it yet. However, there is much research on GPU and GPGPU on web browsers. Before WebGPU, several tries were made to abstract the dependency and create a more friendly API. However, almost all of them are an extension of WebGL or are directly based on it.

Multiple Javascript libraries tried to do the abstraction WebGPU is doing. Sapuan et al. presented an API for Javascript called `gpu.js` [9]. They attempt to solve the problems of complicated GPU APIs by making the library easier to use. For example, it compiles/transpiles Javascript functions into the shader language, making it friendly. The library was inspired by different other libraries that try to solve the same problem, such as `WebCLGL` [9]. More recent libraries that try to do the same the thing are `turbo.js` and `WebMonkeys`. These libraries aim at lower level compared to `gpu.js` by using a simplified GLSL as a shader language. This allows programmers to take more advantage of the GPU, which also grant better performance. Nevertheless, all of the libraries mentioned above use WebGL behind the scenes. But, because it was not designed for doing computing, there are some issues with transporting data between the CPU and the GPU [9].

3 Theoretical Background

This section describes the algorithms and the tools used to evaluate the API and answers the research questions.

3.1 Speedup and Efficiency

The Speedup S is defined as the speed improvement gained by executing the same task on multiple nodes. Both the Efficiency E and speedup S can be calculated by the following formulas

$$S = \frac{T_s}{T_p}, \quad E = \frac{S}{p},$$

Where T_s is the time of the sequential algorithm (one node) and T_p is the time of the parallel algorithm(multiple nodes) and p is the number of parallel nodes.

3.2 WebGPU

WebGL is a well-known low-level 3D graphics API for GPU on web browsers. It is based on OpenGL ES via HTML and is designed to only do graphics rendering, and no direct GPGPU is possible. WebGPU on the other hand, is also an API used to access the GPU through web browsers. It is maintained by the W3C to be the successor of WebGL. It is based on Vulkan, Metal, and Direct3D meaning that it is not a direct port of any existing native API like the case with WebGL. As seen in Figure 1 the API uses other APIs exposed by the OS.

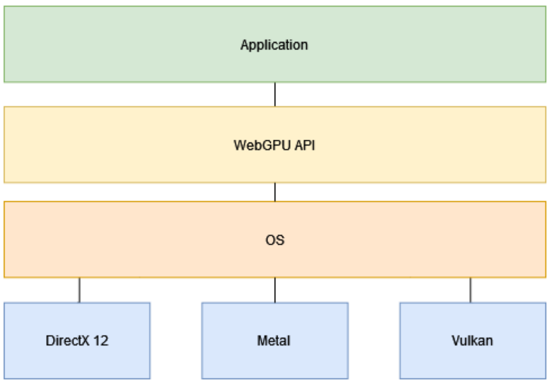


Figure 1: The architecture of WebGPU

Like most other GPU APIs, WebGPU uses Shaders (sometimes called kernels), which are a general kind of program code that runs on the GPU. WebGPU Shading Language (WGSL) is the default shading language, yet other languages are also supported, such as the OpenGL Shading Language (GLSL). WGSL has two kinds of shaders, render shaders, which is used for graphics rendering, and compute shaders to do GPGPU. This paper focuses only on the compute shaders.

3.3 The Mandelbrot Set

A *fractal* is a mathematical set that shows infinitely complex patterns that are self-similar across different scales. The Mandelbrot set, seen in Figure 2 is such a fractal set, which consists of an infinite sequence of numbers defined by the formula below.

$$z_{n+1} = z_n^2 + c$$

Where c is a constant and z is a variable calculated recursively by the formula.

Generating the set is considered an embarrassingly parallel (also called perfectly parallel) problem, meaning that each number can be computed independently which also means little or no communication between nodes/cores.

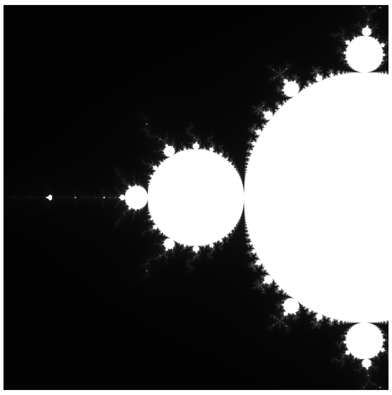


Figure 2: Mandelbrot set example

3.4 The Power Method

The power method (also called Von Mises iteration or The Power iteration) is used to compute the *dominant eigenvector* (the eigenvector corresponding to the largest eigenvalue), which has many usages. For example, Google uses the so-called PageRank as one factor to determine its search ranking [8]. It is also fundamental in AI for problems such as facial recognition [10] and in physics and chemistry for solving differential equations [2].

The method is described by the recurrence relation in the equation below, where A is the input matrix, and x is the approximated vector. Its time complexity is $O(kn^2)$, where n is the size of the matrix, and k is the number of iterations.

$$x_{k+1} \leftarrow \frac{Ax_k}{||Ax_k||}$$

3.5 Compressed Sparse Row

Compressed sparse row (CSR) is a data structure used to represent matrices containing many zeros. As Figure 3 shows, it consists of three lists. The *rowptr* list contains the indices that indicate the start and the end of each row, the *colind* list, which contains the column indices of the non zero values of each row, and the *val* list, which consists of the actual values. Compared to a normal matrix (2D array), CSR matrices have a more efficient space complexity but less efficient time complexity for the insert/update and delete operations. Which makes CSR perfect for problems that involve matrix multiplication, such as the power method, since the matrix is not updated but loaded to memory when the program starts.

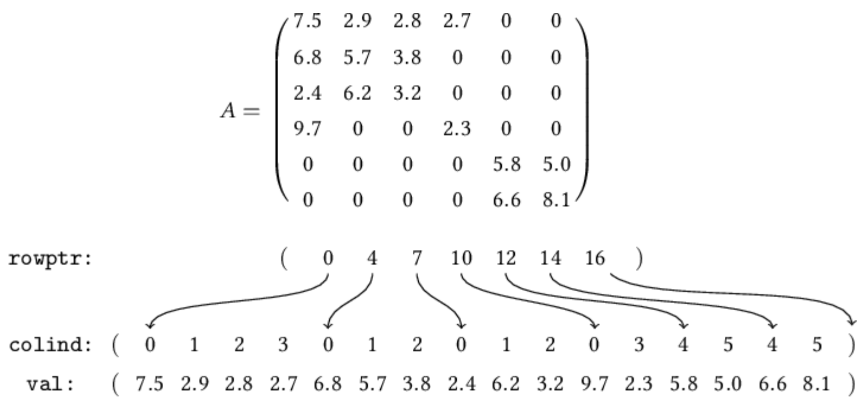

$A = \begin{pmatrix} 7.5 & 2.9 & 2.8 & 2.7 & 0 & 0 \\ 6.8 & 5.7 & 3.8 & 0 & 0 & 0 \\ 2.4 & 6.2 & 3.2 & 0 & 0 & 0 \\ 9.7 & 0 & 0 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5.8 & 5.0 \\ 0 & 0 & 0 & 0 & 6.6 & 8.1 \end{pmatrix}$
rowptr: (0 4 7 10 12 14 16)
colind: (0 1 2 3 0 1 2 0 1 2 0 3 4 5 4 5)
val: (7.5 2.9 2.8 2.7 6.8 5.7 3.8 2.4 6.2 3.2 9.7 2.3 5.8 5.0 6.6 8.1)

Figure 3: CSR matrix storage format [3]

3.6 WebRTC & Communication

Web Real Time Communication (WebRTC) is an open-source real-time communication API over the web. It allows sending video/audio and data between peers. The way webRTC works is that there is no server in the middle. Peers communicate with each other directly. However, peers still need to send their information, such as IP address and port, to other peers to make a connection. WebRTC uses a protocol called ICE that let peers know how to connect. Sending ICEs between peers is called signaling and is usually done using a another way of communication such as WebSockets or chat services.

4 Method

To evaluate the performance of WebGPU and the scalability of it in a cluster of web-browsers two experiments were conducted. A peer-to-peer system was built using WebRTC data channels, and test programs were implemented (for each problem). The WebGPU programs are written *Typescript (CPU)* and *WGSL(GPU)*, and the CUDA program is written in *C++*.

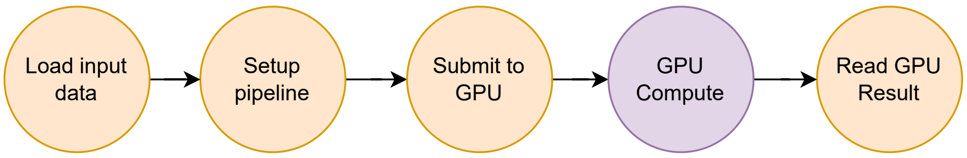


Figure 4: The process of computing on a GPU

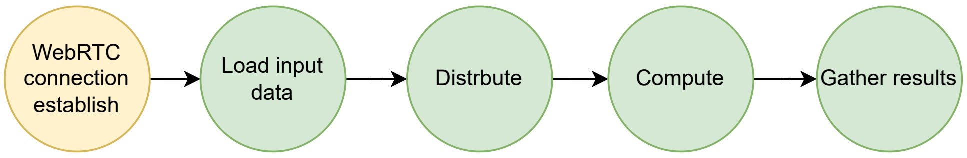


Figure 5: Peer-to-peer process of computing

4.1 Experiments

Three experiments were used to answer the research question. *Matrix Multiplication*, the Power Method and the *Mandelbrot set*. The first two, use CSR to store the matrices and a $2D$ array for the last one. All experiments are conducted on the same computer, and its specifications are seen in Table 1. For each problem size, the test program runs 100 times and the average time is the one that is collected.

Table 1: System Specifications

Software	**OS**	Windows 10
	Browser	Chrome Canary
Hardware	**CPU**	Intel i7-4790 (8) @ 4.000GHz
	GPU	NVIDIA GeForce GT 730
	RAM	32 GB

Table 2: Statistics about the input matrices

Matrix	Dimension	Nonzeros	Diagonal	Average nonzero/row
BCSSTK01	48	224	48	8.3
BCSSTK20	485	1810	485	6.5
BCSSTK16	4884	147631	4884	59
FIDAP011	16614	1091362	16614	66
S3DKT3M2	90449	1921955	90449	21.24
S3DKQ4M2	90449	2455670	90449	27.14

WebGPU Test

The first test was designed to compare the performance of WebGPU and CUDA. The test conducts two experiments, first, computing the largest eigenvector using the power method, and second, computing a Mandelbrot set. The purpose of the experiment is to investigate if the WebGPU would impact the performance on its own. This experiment was run on one node with an increasing problem size. Therefore, no network communication was involved, and multiple matrix sizes and grid sizes were used. Table 2 shows statistics about the matrices[1] used in the power method. The most important thing here is their size. However, for the Mandelbrot, the four grid sizes, 1024^2, 2048^2, 4096^2 and 8192^2 were used.

Figure 4 shows the process of computing, which is split into five phases. First, it loads the input data, which involves parsing the transferring to the GPU device. Then, set up a pipeline that tells the GPU what buffer data are mapped to, how to compute, and what shader to use. Phase three submits the pipeline to GPU and decides how many cores will run in parallel. Phase four is just the GPU running the computation, and finally, in phase five, the results are transferred back to the CPU. Note that all phases run on CPU except the fourth one, which runs on GPU. Also, except for the syntax differences, the process is the same on WebGPU and CUDA.

The run-time of the first two phases are not considered as part of the experiment. Therefore the time is recorded before submitting.

Scalability Test

The second test investigate the scalability of a peer-to-peer system using the Mandelbrot and matrix multiplication. The experiment aims to investigate if speedup increases as increasing the size of the cluster. The 8192^2 size was used for the Mandelbrot, and the matrix `FIDAP011` was for matrix multiplication. The process of the experiment is seen in Figure 5 which also has five phases. First, establishing connections between the nodes using WebRTC. For that, each node signals its ICE through a service called PubNub, which is a simple communication service. One of the nodes acts as a master node and the other are only workers. The master is responsible for phase two, and three, parsing the CSR matrix and distributes it row-wise to workers as seen in Figure 6. Once a worker get a its share of the matrix, it starts computing and report results back to master once done.

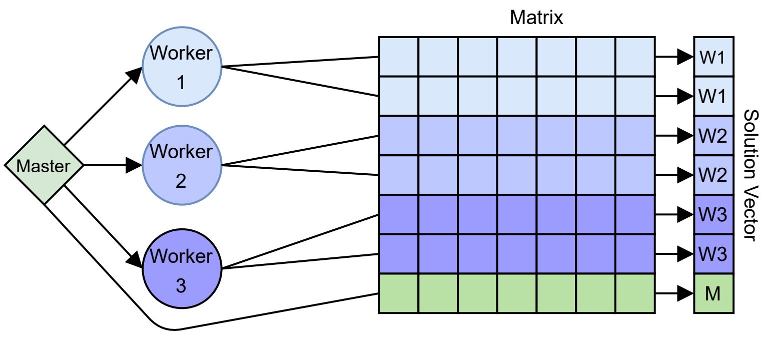


Figure 6: Distributing flow of a matrix between four nodes

[1]Theses matrices were downloaded from The Matrix Market https://math.nist.gov/MatrixMarket/

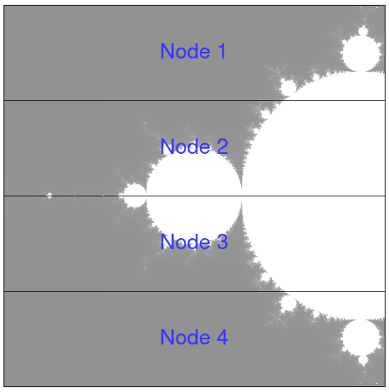


Figure 7: Distbution process of the Mandelbrot set over four nodes

In the case of Mandelbrot, the master sends only the coordinates to each worker. i.e., the x and y coordinates of their loacl computation part, see Figure 7 shows an example of that. However, they still need to report the results back to the master node.

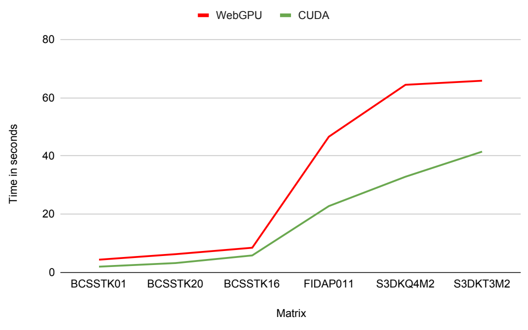
For both problems, the time was recorded after the second phase, where the time of network communication is included.

5 Results and Analysis

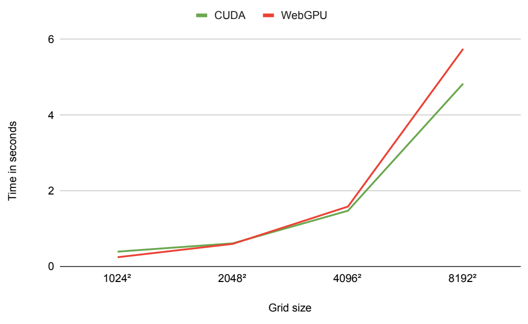
This section presents the results of the problems with some analysis to each one of them.

5.1 Size one cluster - WebGPU vs CUDA

As seen in Figure 8 CUDA has a better performance almost all the time. Although the performance difference gets much worse as the size of problem increase, WebGPU is a little better at the beginning with the Mandelbrot set.



(a) Power method

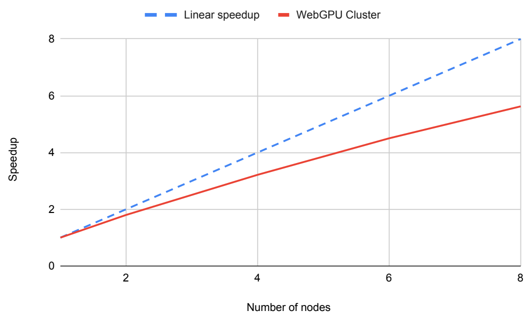


(b) Mandelbrot

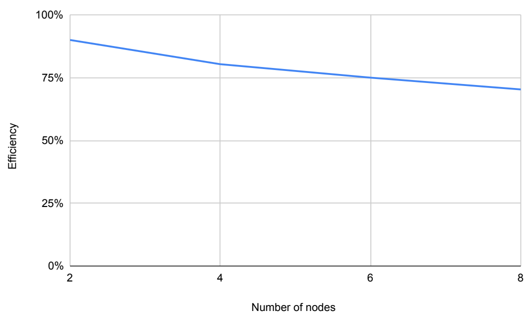
Figure 8: WebGPU vs CUDA on Power Method and Mandelbrot

5.2 Fixed size problem - Scalability

Figure 9 and 10 show the results of speedup and efficiency of generating a Mandelbrot set and multiplying two matrices.

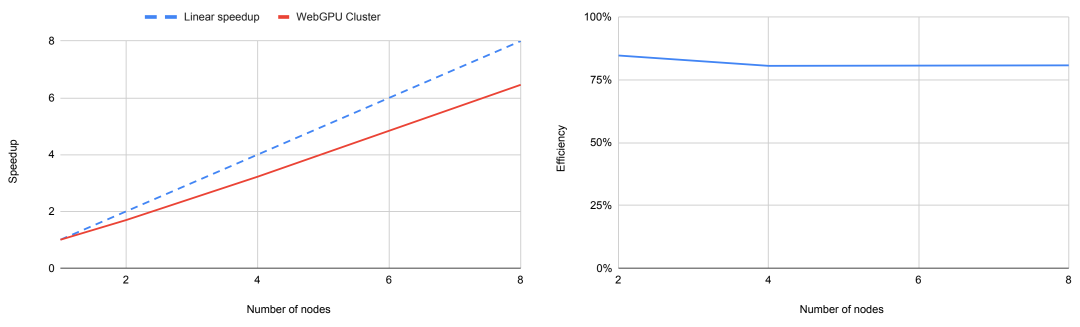


(a) Speedup



(b) Efficiency

Figure 9: Matrix Multiplication - FIDAP011



(a) Speedup **(b)** Efficiency

Figure 10: Mandelbrot - 8192^2

The experiments have very similar speedup at the beginning. The speedup increases as the number of nodes increases. compared to mandelbrot, both speedup and efficiency of matrix multiplication gets a little worse with six and eight nodes. The efficiency of the mandelbrot stays constant after two nodes.

6 Discussion

The results show that WebGPU take longer run-time than CUDA. However this still expected and seems to be logical, given that WebGPU is new and not stable. It is also accessed through Javascript, which usually cost some performance. Also, CUDA is designed specifically for computing with Nvidias GPUs.

The results of the scalability seems to be more positive. The speedup decreases a little as the size of the cluster increases with almost constant efficiency. The reason why speedup decreases is probably because of communication overhead. As more nodes connects to each other, the data to be sent from each peer get bigger and might exceed the bandwidth. Another reason could be that the extra computation that runs on the CPU have a huge impact on the results. Such computation could be data parsing and preparing. One more thing worth noticing, as seen in Figure 9. is that the speedup and efficiency of the Mandelbrot is slightly better than matrix multiplication although both problems are of the same type and they have almost the same process. However, the reason for that is possibly the input data. In the case of matrix multiplication, each node needs to copy the matrices form CPU to GPU, which is very expensive in terms of time. This does not happen in the case of mandelbrot where the input data is only coordinates (3 bytes).

7 Conclusion

According to the results, WebGPU has close performance to CUDA. The results also show that building cluster of web browsers using WebGPU and WebRTC is very scalable with a more than 75% efficiency. This makes building a cluster for GPU scientific purposes very promising. However, this is limited to the cases used in this study. Such a cluster might not be as scalable in other test cases.

8 Future Work

There are multiple areas one could extend this research in. Since this research was limited to three test problems, it would be important to investigate further cases.

Another interesting thing worth investigation is memory usage/utilization and memory transferring between CPU and GPU. This is very important because memory allocation can be very expensive and impact the performance in a negative way.

It would be important to the environment to study if WebGPU and WebRTC have any impact on the energy efficiency. This is especially important for mobiles and laptops since they rely on batteries.

References

[1] Reginald Cushing, Ganeshwara Herawan Hananda Putra, Spiros Koulouzis, Adam Belloum, Marian Bubak, and Cees de Laat. Distributed computing on an ensemble of browsers. *IEEE Internet Computing*, 17(5):54–61, 2013.

[2] Judi J. McDonald David C. Lay, Steven R. Lay. *Linear Algebra and Its Applications, 5th ed.* Pearson, Boston, NY, 2015.

[3] Athena Elafrou, Georgios Goumas, and Nectarios Koziris. A lightweight optimization selection method for sparse matrix-vector multiplication. *ArXiv*, 2015.

[4] Masatoshi Hidaka, Yuichiro Kikura, Yoshitaka Ushiku, and Tatsuya Harada. Webdnn: Fastest dnn execution framework on web browser. *Proceedings of the 25th ACM international conference on Multimedia*, 2017.

[5] Abdul Dakkak; Carl Pearson; Wen-Mei Hwu. Webgpu: A scalable online development platform for gpu programming courses. *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016.

[6] Timo Koskela, Arto Heikkinen, Erkki Harjula, Mikko Levanto, and Mika Ylianttila. Rade: Resource-aware distributed browser-to-browser 3d graphics delivery in the web. In *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 500–508, 2015.

[7] José Á. Morell, Andrés Camero, and Enrique Alba. Jsdoop and tensorflow.js: Volunteer distributed web browser-based neural network training. *IEEE Access*, 7:158671–158684, 2019.

[8] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999.

[9] Fazli Sapuan, Matthew Saw, and Eugene Cheah. General-purpose computation on gpus in the browser using gpu.js. *Computing in Science Engineering*, 20(1):33–42, 2018.

[10] Matthew Turk and Alex Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 01 1991.