The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.10, pp.574~578

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.10.574

Image Processing Acceleration using WebGPU and WebAssembly

Hyunwoo Nam† · Myungho Lee†† · Neungsoo Park†††

웹GPU와 웹어셈블리를 이용한 이미지 프로세싱 가속

남 현 우† · 이 명 호†† · 박 능 수†††

ABSTRACT

JavaScript is slow for high-performance image processing in web browsers and cannot directly utilize the GPU. Therefore, web plugin technology or server-based processing methods have been used. However, since web plugins are no longer supported by the latest web browsers and server processing methods become increasingly expensive as the number of users grows. In this paper, an image processing acceleration method is proposed using the latest web standards such as WASM and WebGPU in a client environment, instead of plugins or server-based methods. The final experimental results confirmed that the WASM+WebGPU-based code, which utilizes both the CPU and GPU, improved execution performance by up to 10 times compared to traditional javaScript.

Keywords : Web Assembly, WebGPU, Image Processing

요 약

웹 브라우저 기반 고성능 이미지 프로세싱을 위해 JavaScript 언어는 속도가 느리고 GPU를 직접 활용할 수 없어서, 웹 플러그인 기술이나 서버 기반 처리 방식이 사용되었다. 하지만 최신 웹 브라우저에서 더 이상 플러그인 기술들이 지원되지 않고, 서버 처리 방식은 사용자가 늘어날수록 운영 비용이 증가하는 문제가 발생하였다. 따라서 본 논문에서는 플러그인이나 서버 기반이 아닌 최신 웹 표준 기술인 WASM과 WebGPU를 활용하여 클라이언트 환경에서 고성능 이미지 프로세싱 알고리즘을 구현하였다. 최종 실험 결과 기존 Javascript에 비해 CPU와 GPU를 동시 활용하는 WASM+WebGPU 기반의 코드에서 최대 10배 이상 실행 성능이 개선되었다.

키워드 : 웹 어셈블리, 웹GPU, 이미지 프로세싱

※ 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(RS-2023-00321688).
† 비 회 원 : 건국대학교 컴퓨터공학부 박사, namhw@konkuk.ac.kr, https://orcid.org/0000-0002-6828-3946
†† 준 회 원 : 명지대학교 컴퓨터공학부 교수, myunghol@mju.ac.kr, https://orcid.org/0000-0003-4383-6425
††† 종신회원 : 건국대학교 컴퓨터공학부 교수, neungsoo@konkuk.ac.kr, https://orcid.org/0000-0002-1032-5760
Manuscript Received : July 26, 2024
Accepted : August 28, 2024
* Corresponding Author : Neungsoo Park(neungsoo@konkuk.ac.kr)

1. 서 론

웹 3.0 시대의 도래와 함께 AR/VR, 인공지능, 이미지/비디오 프로세싱 등 고성능이 요구되는 서비스들이 웹 기반 애플리케이션으로 개발되고 있다[1]. 하지만 JavaScript로 작성된 웹 애플리케이션은 충분한 성능을 제공하지 못하여 Active-X와 같은 비표준 웹 플러그인 기술이나 서버 기반 프로세싱 방식이 사용되었다[2]. 그러나 웹 플러그인 기술은 최신 웹 브라우저에서는 더 이상 지원되지 않으며, 서버 방식은 사용량 증가에 따른 비용 증가 문제가 발생하였다.

따라서 비표준 플러그인 및 서버 기반의 문제점들을 해결하고자 최신 웹 브라우저에서 제공하는 GPU를 직접 활용 가능한 WebGPU 표준과 CPU에서 고속 실행을 보장하는 WASM(WebAssembly) 표준을 적용하고자 한다. 이를 통해 웹 브라우저 기반 고성능 프로세싱 알고리즘의 성능을 개선하고자 한다. 본 논문에서는 WASM 및 WebGPU를 활용한 구체적인 고성능 이미지 프로세싱 방안을 제안하고 구현하여 성능 검증을 진행하였다.

논문의 구성은 다음과 같다. 2장에서는 관련 웹 기술 스택의 발전사와 주요 기술 스택을 분석하고, 3장에서는 이를 활용한 고성능 프로세싱 방안을 제안한다. 4장에서는 최신 웹 표준 기술을 이용한 고성능 이미지 프로세싱 알고리즘의 구현 결과에 대한 성능 측정 결과를 다루고 있다.

2. 관련 연구

2.1 웹 브라우저 기술 스택의 발전

과거 웹 브라우저와 인터프리터 방식의 JavaScript 언어는 속도가 느려 단순 웹 문서 렌더링만 수행하는 목적으로 사용되었다. 따라서 고성능 이미지 처리를 위한 목적으로는 충분하지 않아 Active-X, Flash, Java Applet 등의 플러그인 기술이 사용되었으나, 보안 문제와 기술 파편화로 인하여 최신 웹 브라우저에서는 지원이 중단되었다. 그리고 현재는 더 이상 비표준 웹 기술들은 사용이 불가능해져 앞으로는 서버 기반의

Table 1. Comparison Analysis by Code Type

Type	Speed	Dev	Language	Environments
Javascript	Slow	CPU	Javascript	any web browser
WASM	Faster than JS	CPU	C, C++, Rust	modern web browser
WebGPU	Fast	GPU	Rust	modern web browser

프로세싱 또는 웹 표준 기술 형태로만 고성능 프로세싱 알고리즘을 구현해야만 한다.

고성능 처리를 위한 웹 표준 기술로는 2014년 WebCL[3] 표준이 발표되어 웹에서도 이종 환경 기반 고성능 알고리즘 구현이 가능해졌으나, 현재는 표준 참여 단체들의 입장 차이로 인하여 모든 웹 브라우저에서 지원이 중단되었다.

이외에도 2013년 파이어폭스에서는 자바스크립트 엔진에 최적화된 asm.js 기술을 발표하여 JavaScript보다 뛰어난 실행 속도를 제공하였으나, 파일의 크기가 크기 때문에 초기 로드 시간 느리다는 문제가 발생하였다. 이를 해결하기 위해 주요 웹 브라우저들은 2017년 asm.js 기반의 바이너리 포맷인 WASM(WebAssembly) 표준 기술을 발표하였으며, 이는 웹뿐 아니라 다양한 플랫폼으로 확장될 전망이다.

다음으로 OpenGL 기반 WebGL 표준으로 웹에서 GPU 기반 3D 그래픽 처리를 가능케 하였으나 이는 범용 목적의 GPGPU 연산 처리에는 적합하지 않았다. 이를 보완하기 위해 2021년 WebGPU 표준 초안이 발표되었으며, 주요 웹 브라우저 벤더들이 함께 개발을 진행해오고 있다. 특히 2023년 4월 크롬 웹브라우저 공식 버전에 WebGPU 기능이 활성화되면서 그 확장 가능성이 커지고 있다[4].

따라서 최신 웹 브라우저를 위한 고성능 이미지 프로세싱 구현을 위해서는 표준 기술인 WASM과 WebGPU를 활용하여 핵심 로직을 개발해야 하며 각 기술별 특징은 Table 1과 같다. WASM 코드는 대부분의 최신 웹 브라우저에서 지원되며 CPU에서 실행된다. 반면, WebGPU는 현재 크롬에서만 공식적으로 지원되고 있으며, 다른 브라우저에서는 실험 모드에서 적용 가능한 상태이다[5,6].

2.2 WASM 및 WebGPU 런타임 엔진 구조

웹 표준 기반의 고성능 애플리케이션을 개발하기 위해서는 WASM 및 WebGPU 코드를 사용한 웹 애플리케이션 개발이 필요하다. 각 코드는 CPU 및 GPU에서 실행되며 기존 Javascript 언어에 비해 월등한 성능 향상을 보인다. 최신 크롬 웹 브라우저에서 WASM과 WebGPU 코드는 Fig. 1과 같은 런타임 구조에서 실행된다.

먼저 JS 코드는 파싱 및 컴파일을 통해 바이트 코드를 생성하여 Turbofan 컴파일러를 통해 CPU에서 실행된다. 하지만 바이너리 파일 형태인 WASM은 Liftoff에서 파싱 및 컴파일 과정이 단축되어 성능이 개선되었다. 이와 달리 WebGPU는

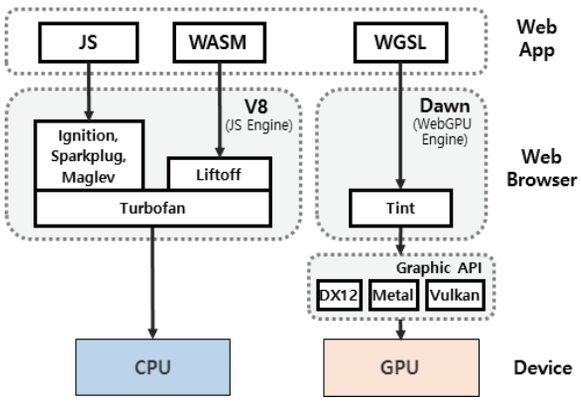

Fig. 1. Chrome Web Browser Execution Architecture for WASM and WebGPU

WGSL(WebGPU Shader Language) 코드를 Tint 컴파일러로 빌드하며 이는 Dawn 엔진에서 실행 환경의 Graphic API인 DirectX12, Metal, Vulkan을 통해 GPU에서 실행된다.

2.3 웹 기반 이미지 프로세싱 라이브러리

- OpenCV.js : 컴퓨터 비전 분야에서 많이 사용되는 OpenCV 오픈소스 프로젝트를 웹에서 사용 가능하도록 Emscripten을 활용하여 WASM 코드로 변환한 라이브러리이다[7]. OpenCV의 주요 기능들이 포팅되어 있어 기존 Native 환경에서 구현되었던 코드들을 웹에서도 활용할 수 있다는 호환성 측면에서의 장점이 있다.
- Photon : 웹 브라우저와 Node.js에서 사용 가능한 웹 이미지 프로세싱 라이브러리로, Rust 언어로 작성된 코어 라이브러리를 WASM으로 컴파일하여 웹에서도 네이티브에 가까운 실행 성능을 보장하는 라이브러리이다.[8]
- wgimage : WebGPU 기반의 이미지 필터 알고리즘을 제공하는 라이브러리로. OpenCV.js나 Photon에 비해 GPU를 직접 활용하고 있어 기존 CPU 기반 라이브러리보다 성능이 월등하다[9]. 하지만 현재 지원하는 알고리즘의 개수가 적다는 단점이 있다.

위와 같은 웹 표준 기반의 이미지 프로세싱 라이브러리들은 WASM 및 WebGPU으로 구현되어 성능적으로 개선되었다. 또한 라이브러리들은 JavaScript 기반의 API를 공통적으로 제공하여 일반적인 웹 개발과 유사한 방식으로 개발할 수 있다는 장점이 있다.

3. 웹 표준 기반 고성능 이미지 프로세싱

최신 웹 브라우저 기반의 고성능 이미지 프로세싱 로직을 구현하기 위하여 본 논문은 웹 표준 기술인 WASM 코드와 WebGPU 코드를 사용하여 알고리즘을 구현하였다. WASM과 WebGPU 코드는 각 실행 환경에 최적화하기 위해 선택적으

로 적용하거나 병합하여 실행해야 한다.

이를 위해 본 연구에서는 고성능 이미지 프로세싱을 위한 필터 알고리즘(Grayscale, Invert, Sepia)들을 WASM 및 WebGPU 코드로 작성하였다. 그리고 이를 하나의 Web App으로 배포될 수 있도록 단일 패키지 형태로 병합하였으며 최적의 성능을 도출하기 위해 CPU, GPU를 모두 활용할 수 있는 형태로 필터 알고리즘을 구성하였다.

3.1 이미지 프로세싱 WASM 및 WebGPU 코드 구현

먼저 현재 웹 브라우저에서 최적의 성능을 보장할 수 있도록 이미지 프로세싱 알고리즘 로직들을 WASM 및 WebGPU 코드로 생성해야 한다. 각 알고리즘들은 Fig. 3과 같이 공통적으로 전처리와 프로세싱 단계 그리고 후처리 단계로 크게 3단계로 나눌 수 있다. 따라서 각 단계의 세부 로직들을 WASM 및 WebGPU 코드로 작성하였으며, 코드의 구성을 달리해가며 최적의 알고리즘 코드 구성을 확인하고자 하였다.

WASM 코드는 C, C++, Rust 언어 등을 사용하여 작성 가능하며 만약 이미지 프로세싱 코드가 이미 해당 언어들로 구현되어 있을 경우 Emscripten 컴파일러를 사용하여 해당 코드들을 WASM 코드로 변환할 수 있다. 하지만 원본 소스코드에서 사용하는 외부 라이브러리가 WASM으로 변환되지 않은 경우, 해당 부분들도 변환 및 링킹 작업이 필요하다. 따라서 참조하는 외부 라이브러리의 소스코드가 공개되어 있지 않거나 복잡한 경우 웹 환경으로 포팅하는 작업은 어렵기 때문에 신규로 개발하는 방안이 효율적이다. 본 논문은 코드 타입별로 성능을 비교 분석하고자 C언어를 이용하여 신규로 이미지 프로세싱 알고리즘 코드를 작성하였으며, 이를 이용하여 WASM 코드를 생성하였다.

다음으로 WebGPU 코드를 생성하기 위하여 WebGPU 표준 그룹에서 제정한 WGSL(WebGPU Shader Language) 셰이더[10]를 Rust 언어[11]를 사용하여 작성하였다. 또한 셰이더 코드는 동일 기능의 WASM 코드와 같은 입/출력 데이터를 처리할 수 있도록 인터페이스를 호환되도록 개발하였다.

다만 웹 페이지의 리소스(이미지)를 로드하거나 전/후처리 해주는 작업등은 Host 영역을 접근해야 하기 때문에 GPU에서는 직접적으로 처리할 수 없다. 따라서, 해당 기능들은 Host 언어로 사용 가능한 Javascript 또는 WASM 코드로 작성하여 적용하였다.

3.2 웹 애플리케이션 패키징

고성능 웹 애플리케이션을 위해 생성된 WASM 및 WebGPU 커널 코드들은 하나의 애플리케이션에 병합되어야 하며, 이는 Fig. 2와 같은 패키징 단계를 거쳐 생성된다.

먼저 WebGPU 코드는 WGSL 셰이더 언어로 작성되며 이는 별도의 컴파일 과정 없이 소스코드 형태로 Web App에 패키징 된다. 현재 WGSL 셰이더는 Rust 언어로 작성되었으며 런타임시에 빌드되어 각 GPU 실행 환경에 따라 DirectX 12

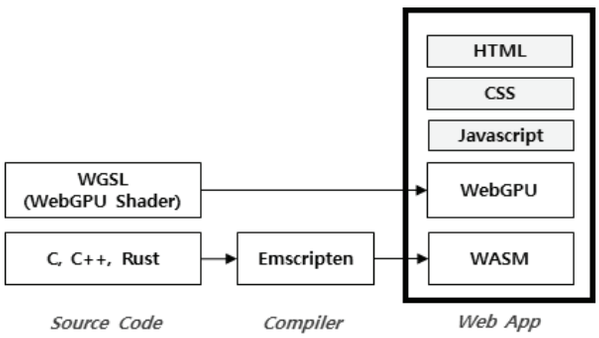

Fig. 2. High-performance Web Application Packaging

[12], Metal[13], Vulkan[14]과 같은 Native API 기반 위에서 실행된다. WGSL 코드는 일반적으로 텍스트 파일 형태로 배포되기 때문에 보안상의 문제점을 보완하기 위해 난독화 및 압축을 통해 보안성을 향상 시킬 수 있다[15].

다음으로 WASM은 C언어를 사용하여 작성하였으며 Emscripten 컴파일러를 이용하여 플랫폼 독립적인 바이너리 파일 형태로 WASM 코드를 생성한다. 이후 생성된 WASM 코드는 Web App에 함께 패키징되어 배포된다.

결론적으로 일반 웹 페이지 코드인 HTML, CSS, Javascript 코드와 WebGPU 및 WASM 코드를 하나의 Web App 으로 패키징하였다. 그리고 이와 같은 패키징 과정을 통해 CPU, GPU 이종 실행 환경에서 최적 실행을 위하여 하나의 웹 애플리케이션으로 구성된다.

3.3 이미지 프로세싱 알고리즘 실행 구조

본 논문은 각 이미지 필터 알고리즘들을 각기 다른 웹 표준 기술 타입에 따라 개별적으로 제작한 후 성능을 비교하였으며, 세부 구성 내역은 Fig. 3과 같다. 각 구성 형태에 따라 총 실행 성능을 측정하고자 하며 크게 3가지 단계로 각 필터 알고리즘을 세분화 할 수 있다.

본 연구의 실험에서는 3가지 종류의 이미지 필터 알고리즘들을 JS, WASM, WebGPU 코드 모듈로 개발하였으며, 세부적으로는 DOM 영역에 있는 img 태그의 이미지 Raw 데이터에 대한 전처리/후처리 작업에 해당하는 Convert Image Data 단계 그리고 실제 이미지 필터 알고리즘이 수행되는 Image Filter 단계로 나뉜다.

먼저 HTML 문서의 〈img〉 태그의 이미지 데이터를 로드하고 이미지 프로세싱을 위한 RGB 포맷으로 변환하는 Convert Image Data 단계의 로직은 JavaScript와 WASM 모듈만을 사용할 수 있었다. 이는 WebGPU가 웹 페이지의 이미지 데이터에는 직접적으로 접근할 수 없어 해당 단계의 로직 구현이 불가능하기 때문이다.

다음으로 주요 이미지 프로세싱 작업이 이뤄지는 Image Filter 단계에서는 3가지 필터 알고리즘(GrayScale, Invert, Sepia)을 JavaScript, WASM, WebGPU 코드 모듈로 구현하였다. 각 커널 코드들은 독립적인 이미지 영역에 대하여 연산을

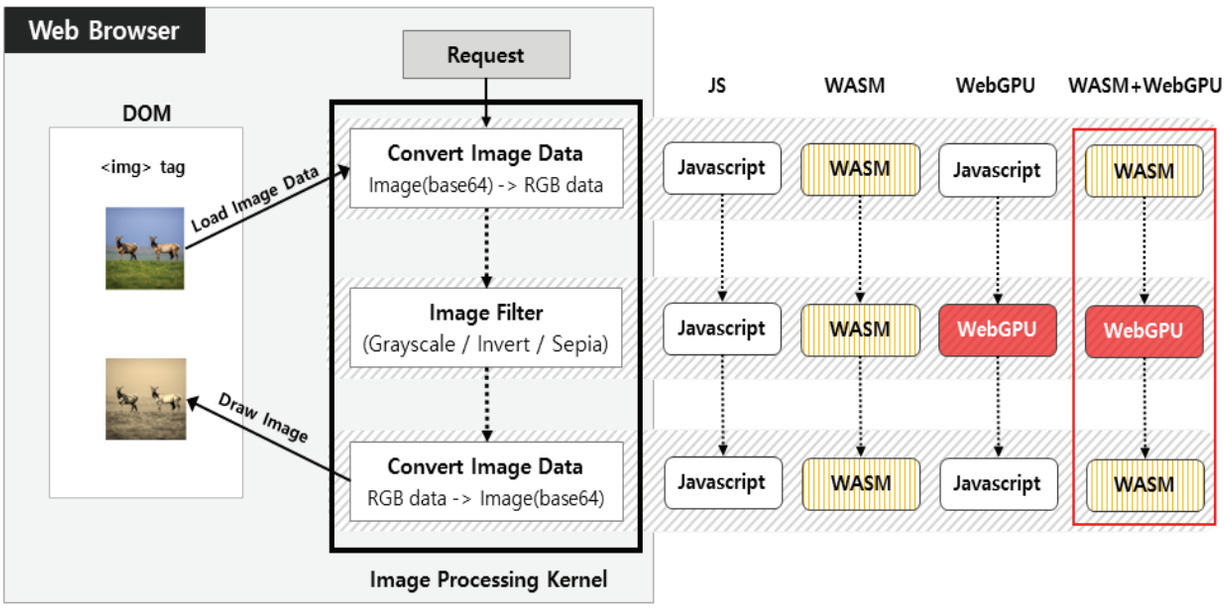

Fig. 3. Image Processing Step

수행함으로써 동기화 이슈가 없도록 구현되었으며 알고리즘 구성에 따라 선택적으로 적용해가며 실험하였다.

기본적으로 WebGPU 코드는 GPU에서 커널 코드들을 병렬로 실행 가능하지만 JS 및 WASM은 단일 CPU에서 순차 실행만 가능하다. 또한 WASM+WebGPU 실험 구성과 같이 이종 환경에서 상이한 코드들을 동시에 실행하기 위하여 Web-Worker를 적용하였다. 또한 각 코드에서 사용되는 입/출력 데이터들은 SharedArrayBuffer 객체를 사용하여 데이터를 공유함으로써 성능을 개선하였다.

4. 실험 및 성능 평가

실험에서 성능 비교를 위하여 3가지 이미지 필터 알고리즘들을 Javascript, WASM, WebGPU 및 WASM+WebGPU 실험 구성에서 성능을 측정하였다. 실험 방법으로는 이미지 크기를 달리해가며 전체 실행 시간을 측정하였으며 실험 결과는 Fig. 4 및 Fig. 5와 같다.

먼저 Fig. 4는 Sepia 필터 알고리즘 1개를 이미지 크기별로 처리 시간을 로그 스케일 단위로 측정한 결과이다. 결과를 보면 가장 작은 400x300 이미지 크기에서 WASM이나 WebGPU를 사용한 경우보다 오히려 Javascript로 실행하였을 때 전체 실행 시간이 작았다. 이는 WebGPU 코드의 로드 단계에서 셰이더 코드를 컴파일하고 데이터를 송/수신하는 통신 작업과 같은 오버헤드로 인한 결과이며, WASM 코드의 경우 파일의 크기가 JS 보다는 커서 초기 로딩 시점에 지연이 발생하기 때문이다. 따라서 작업 크기가 작은 경우에는 오히려 속도가 느려질 수 있기 때문에 고성능이 요구되는 1280x1920 크기 이상의 큰 이미지를 대상으로 Fig. 5와 같이 전체적인 성능 측정

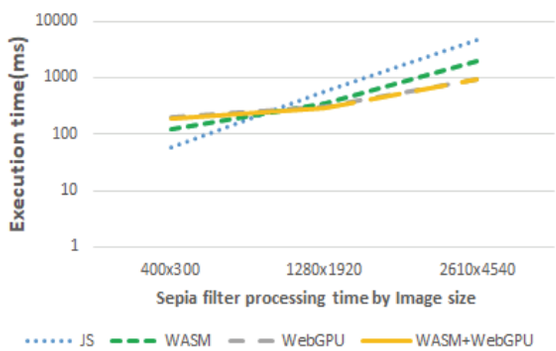

Fig. 4. Measurement of Sepia Filter Processing Time

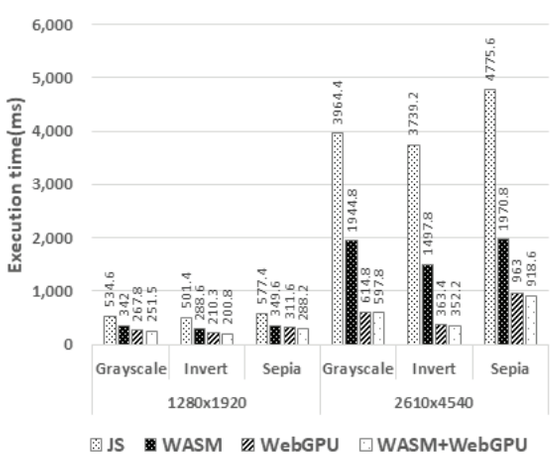

Fig. 5. Measurement of Image Processing Execution Time

을 진행하였다. 본 실험 결과를 통해 WASM 및 WebGPU의 적용 여부가 무조건 성능이 개선되지 않다는 것을 알 수 있으며, 작업 크기에 따라 각 코드의 적용 여부가 결정되어야 한다는

것을 확인하였다.

Fig. 5의 첫 번째 실험은 일반 모니터 해상도(1280x1920) 크기의 이미지 프로세싱 결과이며 WebGPU를 사용하는 경우 성능이 가장 우수했다. GPU를 사용하는 경우 CPU 단독 실행하는 JS보다 약 2~2.5배 향상되었으며 WASM과 비교하면 1.2~1.4배 정도 성능이 개선되었다. 그리고 WebGPU 단독으로 실행했던 경우와 WASM+WebGPU를 동시 적용한 경우를 비교하면 두 코드를 동시에 활용하였을 때 최대 7% 정도의 성능 개선이 있었다.

두 번째 실험은 더 큰 2610x4540 크기의 이미지 프로세싱 결과이며 GPU를 사용하는 경우 이전 실험 결과보다 성능 개선 효과가 더 커졌다. 이는 작업 크기가 증가 할수록 GPU의 병렬 효율성이 증가하여 성능 개선 효과가 더 뚜렷하게 나타난 것으로 JS나 WASM에 비해 약 5.2배~10배 성능이 향상되었으며, WebGPU와 대비하여 WASM+WebGPU의 실험 결과에서 최대 4% 정도 성능이 개선되었다. 또한 본 실험 결과로 유추해보면 이미지의 크기가 커질수록 JS 대비 WASM 또는 WebGPU를 사용할 경우 성능 개선 효과가 더욱 커질 것으로 예상할 수 있다.

결론적으로 작업 크기가 커질수록 WebGPU, WASM 순으로 성능이 높아지며, 종합적으로는 CPU, GPU를 모두 활용하는 WASM+WebGPU 코드 조합의 실험에서 이미지 프로세싱 알고리즘의 속도 개선 효과가 가장 크다는 것을 확인하였다.

5. 결 론

본 논문은 웹 애플리케이션 개발 언어인 Javascript 대신 고성능 실행을 위한 WASM 및 WebGPU 표준을 활용하여 이미지 프로세싱 알고리즘을 구현하였으며, 실험을 통해 웹 표준 기반 이미지 프로세싱 알고리즘의 성능을 개선하였다. 이를 통해 비표준 웹 브라우저 플러그인이나 서버 처리 방식이 아닌 웹 표준 기술만으로 성능 요구사항을 만족할 수 있음을 검증하였다. 이를 위해 최신 웹 표준 기반의 고성능 프로세싱을 위한 코드 구성 및 패키징 방안 그리고 CPU, GPU 동시에 활용할 수 있는 실행 방안을 제안하였다.

향후 연구로는 현재 적용한 웹 이종 환경 기반 이미지 프로세싱 알고리즘의 추가적인 최적화 연구를 진행할 것이며 또한 연구 결과를 범용적인 활용이 가능하도록 프레임워크 형태로 확장해 나갈 계획이다.

References

[1] J. T. Park and I Y. Moon, “Development trends in web convergence service implementation technology,” *The Journal of The Korean Institute of Communication Sciences*, Vol.38, No.4, pp.3-9, 2021.

[2] D. Kim, “Cloud Computing to Improve JavaScript Processing Efficiency of Mobile Applications,” *Journal of Information Processing Systems*, Vol.13, No.4, pp.731-751, 2017.

[3] C. W. Kim and H. J. Cho, “Profiler Design for Evaluating Performance of WebCL Applications,” *KIPS Transactions on Computer and Communication Systems*, Vol.4, No.8, pp.239-244, 2015.

[4] H. W. Nam and N. S. Park, "Accelerating AES Algorithm using WebGPU," *The transactions of The Korean Institute of Electrical Engineers*, Vol.71, No.7, pp.1008-1014, 2022.

[5] H. K. Kim, “Web assembly technology trend research,” *The Journal of The Korean Institute of Communication Sciences*, Vol.40, No.2, pp.3-9, 2023.

[6] C. H. Shin, J. H Yeo and S. M Moon, “Analysis of Process and Performance in WebAssembly,” in *The Korean Institute of Information Scientists and Engineers*, pp.1546-1548, 2018.

[7] opencv.js, https://github.com/TechStark/opencv-js

[8] Photon, https://silvia-odwyer.github.io/photon/

[9] wgimage, https://github.com/neka-nat/wgimage

[10] WebGPU Shading Language-W3C Working Draft, https://www.w3.org/TR/WGSL/

[11] N. D. Matsakis, F. S. Klock, “The Rust Language,” ACM SIGAda Ada Lett. 103-104, 2014

[12] Direct3D 12 programming guide, https://learn.microsoft.com/en-us/windows/win32/direct3d12/

[13] Metal, https://developer.apple.com/kr/metal/

[14] Vulkan-Cross platform 3D Graphics, https://www.vulkan.org

[15] YUI Compressor-Javascript Compressor, https://yui.github.io/yuicompressor/