NPU 환경에서의 영상 전처리 연산 구현 및 효율성 분석

AI-Chip 응용분야

The Architects - 김연재 · 김민서 · 이건하 · 김호준

2025.10.17

Table of Contents

- **Overview**
  - 과제 소개 및 목표
  - 도전 과제와 기술적 중요성
- **Approach**
  - 추가 구현 대상 함수 선정
  - NPU 아키텍처별 최적화 전략
- **Evaluation**
  - 성능 측정 환경 및 지표
  - NPU vs. GPU 성능 심층 비교
- **Conclusion**
  - 핵심 결과 요약 및 향후 활용 방안

Team *The Architects* 소개

팀장: 김연재

✓ 서울대 컴공 20
✓ 학부 연구인턴 경험
✓ 관심사: ML Systems, Training Optimization

팀원: 김민서

✓ 서울대 컴공 20
✓ 학부 연구인턴 경험
✓ 관심사: ML Systems, Inference efficiency

팀원: 이건하

✓ 서울대 컴공·통계 19
✓ 학부 연구인턴 경험
✓ 관심사: ML system, parallel programming

팀원: 김호준

✓ 서울대 전기 20
✓ 학부 연구인턴 경험
✓ 관심사: ML Systems, AI Agents

Overview

과제 소개 및 목표

FuriosaAI NPU

Rebellions NPU

Nvidia GPU

과제 소개 및 목표

- 대표적인 **영상 전처리 연산**들을 **3개의 아키텍처(GPU, Furiosa Warboy, Rebellions ATOM)** 환경에 각각 구현 및 이식
- GPU를 기준으로 NPU의 성능을 다각적으로 비교 분석
  - **정확성 분석**: 연산 결과의 정밀도 비교
  - **효율성 분석**: 처리 속도(Latency) 효율성 비교

과제 소개 및 목표

- 대표적인 **영상 전처리 연산**들을 **3개의 아키텍처(GPU, Furiosa Warboy, Rebellions ATOM)** 환경에 각각 구현 및 이식
- GPU를 기준으로 NPU의 성능을 다각적으로 비교 분석
  - **정확성 분석**: 연산 결과의 정밀도 비교
  - **효율성 분석**: 처리 속도(Latency) 효율성 비교

정량적 분석 결과를 바탕으로, 실제 AI 응용 환경에 최적화된 하드웨어 활용 전략 제시

도전 과제와 기술적 중요성

- **근본적인 아키텍처의 차이**
  - **GPU:** 대규모 병렬 처리(SIMD)에 강점, 유연한 프로그래밍 가능
  - **NPU:** Matmul, Conv 등 특정 AI 연산에 극도로 최적화됨
- **제한된 연산자 및 데이터 타입**
  - **GPU:** 대부분의 연산을 지원
  - **NPU:** 하드웨어적으로 내장된 특정 연산자만 고속으로 처리 가능, 지원되지 않는 연산은 CPU offloading 혹은 GPU 에서 처리(병목 발생), 정수형 연산 중심으로 Quantization이 필요함
- **컴파일 기반의 실행 모델**
  - **GPU:** 커널 실행 병목을 줄임, Graph 기반 실행 가능
  - **NPU:** 사전에 정의된 고정 크기의 입력에 대해 모델을 컴파일하여 사용

도전 과제와 기술적 중요성

- **근본적인 아키텍처의 차이**
  - **GPU:** 대규모 병렬 처리(SIMD)에 강점, 유연한 프로그래밍 가능
  - **NPU:** Matmul, Conv 등 특정 AI 연산에 극도로 최적화됨
- **제한된 연산자 및 데이터 타입**
  - **NPU:** 사전에 정의된 고정 크기의 입력에 대해 모델을 컴파일하여 사용

NPU 아키텍쳐 한계 극복을 위해 알고리즘 수준의 재설계와 최적화가 필수적이다

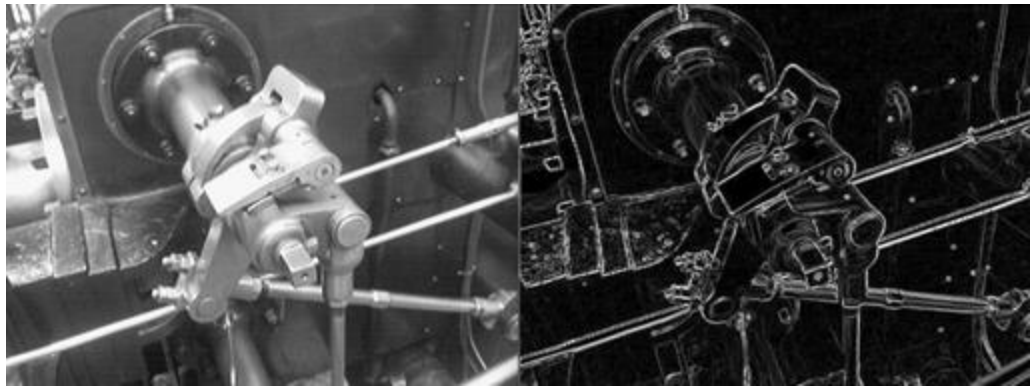
도전 과제와 기술적 중요성

- **차세대 AI 응용을 위한 필수 기술 확보**
  - 자율주행, 스마트팩토리 등 실시간성이 필수적인 AI 응용에서 영상 전처리는 성능의 핵심
  - 저전력·고효율 특성을 가진 NPU 활용 가능
- **국산 AI 반도체의 실용성 검증 및 생태계 활성화**
  - 국산 NPU(Furiosa, Rebellions)의 실제 응용 환경에서의 성능을 정량적으로 검증.
  - NPU 이식 및 최적화 사례를 확보하여, 향후 다른 개발자들이 국산 AI 반도체를 쉽게 활용할 수 있는 생태계의 기반을 마련
- **미래 AI 시스템 설계의 기반 마련**
  - GPU와 NPU의 처리 효율을 비교 분석하여, 가장 효율적인 하드웨어 선택 기준과 최적화 방향을 제시

Approach

구현 대상 함수 - 기본 전처리 연산 5종

- **기하학적 변환 (Geometric Transformation)**
  - Resized Crop: 이미지 크기를 0.5x, 1x, 2x로 조절 및 잘라내기
  - Horizontal / Vertical Flip: 이미지를 수평 또는 수직으로 뒤집는 연산
- **픽셀 단위 변환 (Pixel-wise Transformation)**
  - Color Jitter: 이미지의 밝기(Brightness)와 대비(Contrast)를 조절
- **컨볼루션 기반 필터링 (Convolution-based Filtering)**
  - Gaussian Blur: 3x3, 5x5 등의 커널을 이용해 이미지를 부드럽게 처리
  - Sobel Edge Detection: 이미지의 경계선을 검출하는 대표적인 엣지 검출 필터



Sobel Edge Detection Example

구현 대상 함수 - 심층 전처리 연산 Edge Enhancement

Edge Enhancement:

영상 내 경계(에지) 정보를 강조하여 **선명도와 시각적 대비를 향상**시키는 전처리 기법

- 주로 고주파 성분(경계, 세부 구조)을 강화하고 저주파 성분(평탄한 영역)을 억제함으로써 이미지의 디테일을 부각
- Kernel_size와 Alpha 값에 따라 사용 목적이 다름

활용 방안

- **선명도 개선**: 흐릿한 영상의 경계를 뚜렷하게 만들어 객체 인식률 향상
- **후처리 대비 향상**: 이후 단계(검출·분할 등)의 특징 추출 성능 개선
- **NPU 적합성**: convolution 기반 필터링으로 구조 단순, 병렬 처리 효율 높음

구현 대상 함수 - 심층 전처리 연산 Zero-DCE

Zero-DCE (Zero-Reference Deep Curve Estimation):

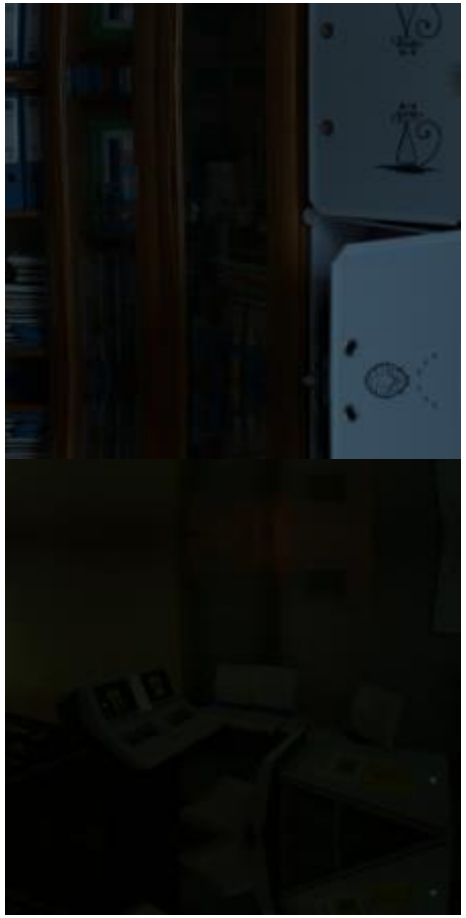
저조도 이미지를 개선하는 경량 Convolution 기반 모델

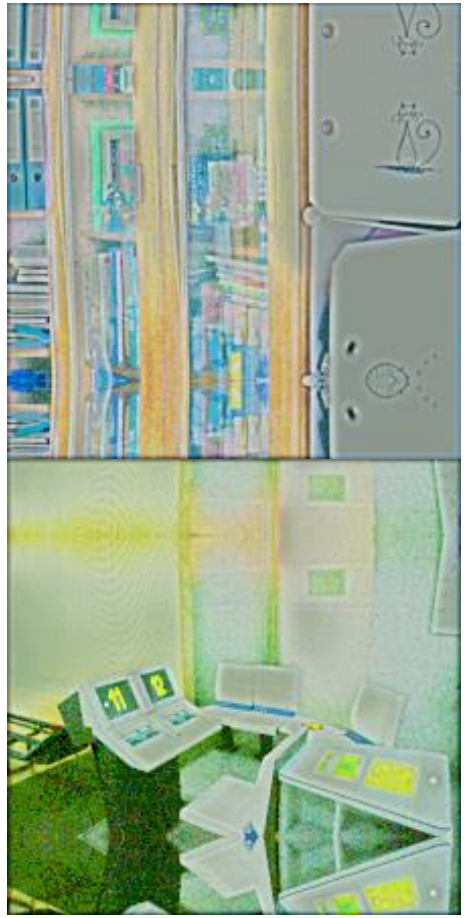
- 저조도 환경에서 얻은 이미지를 객체 인식하기 위해 많이 사용

활용 방안

- **선명도 개선**: 흐릿한 영상의 경계를 뚜렷하게 만들어 객체 인식률 향상
- **후처리 대비 향상과 높은 실용성:** 이후 단계(검출·분할 등)의 특징 추출 성능 개선 및 자율주행·감시 등 저조도 환경 전처리에 유용
- **NPU 적합성**: convolution 기반 필터링으로 구조 단순, 병렬 처리 효율 높음

구현 대상 함수 - 심층 전처리 연산 Zero-DCE





Model결과 Dataset - geekyrakshit/LoL

Challenge 1: 다양한 Input Shape 처리

핵심 과제: GPU와 달리, NPU는 사전 컴파일된 고정된 크기의 Input에 대해서만 동작함.

해결 방안 1: Dynamic Padding

방법: 기준 모델 크기에 맞춰 모든 입력을 Padding으로 채움.

장점: 하나의 모델로 모든 크기 대응 가능.

단점 1: CPU에서 수행되는 전/후처리로 인한 Overhead 발생.

단점 2: Padding으로 인해 Accuracy 저하 가능

해결 방안 2: Multi-Model Compilation

방법: 자주 사용되는 여러 Input Shape에 맞춰 각각의 모델을 컴파일.

장점: Padding Overhead 감소.

단점 1: 컴파일된 모델들이 NPU 메모리를 많이 차지함.

단점 2: 여전히 Compile 되지 않은 새로운 Input이 들어오면 해결 불가

Challenge 2: Supported Ops의 한계

핵심 과제: NPU는 최적화를 지원하는 Operation들이 정해져있다.

해결 방안: Supported Ops를 보고 CPU와 NPU에서 처리할 함수를 나눈다.
구현 예시:
- Resized Crop : Padding은 CPU, Interpolate은 NPU에서
- Zero-DCE: Curve Add는 CPU, Convolution은 NPU에서

cf) Add도 가속을 지원하지만 Curve add를 실험적으로 측정했을때 NPU보다 CPU가 최적

Challenge 3: CPU-NPU 통신 오버헤드 최소화

핵심 과제: NPU는 연산 시마다 CPU-NPU 간 데이터 전송이 발생하며, 이 통신 비용이 병목이 됨.

해결 방안: 연산 통합 (Operation Fusion)
전략: 여러 전처리 함수를 하나의 NPU 모델로 묶어 CPU-NPU 왕복 횟수를 최소화.
구현 예시:
- H-Flip과 V-Flip을 동시에 처리하는 단일 모델.
- 3x3, 5x5 Gaussian Blur를 한번에 적용하는 단일 모델.

추가 전략: 비동기 실행 (Asynchronous Execution), Batch 전략
전략: NPU 연산과 다른 작업(e.g., 이전 결과 전송)을 중첩시켜, 연속적인 데이터 스트림 환경에서의 전체 처리량(Throughput) 극대화.

참고: 이번 대회는 이미지별 지연 시간(Latency) 측정을 위해 개별 이미지 단위 동기화(sync) 방식을 사용하므로, 비동기 전략과 Batch 전략 적용 X

NPU-specific Approach: ATOM NPU 최적화 전략

주요 특징:

- 여러 모델 동시 로딩 가능 (메모리 제약 존재).
- 다양한 Precision 지원 (e.g., INT8, FP16).

최적화 전략:

- Input 처리: Multi-Model과 Padding 전략을 혼용.
- 가속 가능 연산 사용: NPU에서 최적인 함수만 사용
- 연산 통합: Padding Overhead와 모델 교체 IO 비용을 상쇄하기 위해, 가능한 많은 연산을 하나의 모델에 통합하여 호출당 작업 효율을 극대화.

NPU-specific Approach: Warboy NPU 최적화 전략

주요 특징:

- 한 번에 하나의 모델만 로딩 가능.(런타임 하나당 하나의 러너(모델)만 할당가능)
- INT8 Quantization 필수 적용.
- 지원되는 연산이 비교적 적으며 지원되지 않는 연산은 CPU로 Fallback

최적화 전략:

- Input 처리: Padding을 사용하여 하나의 Model만 이용
- 정밀도 손실 최소화: INT8 양자화에 실제 image data 사용하여 scale 결정
- 가속 가능 연산 사용: CPU Fallback을 줄임
- 연산 통합: Padding Overhead와 모델 교체 IO 비용을 상쇄하기 위해, 가능한 많은 연산을 하나의 모델에 통합하여 호출당 작업 효율을 극대화.

NPU-specific Approach

	ATOM	Warboy
Flip	Multi Model, Fusion	Padding, Fusion
Gaussian Blur	Padding, Fusion	Padding, Fusion
Color Jitter	Padding, Fusion	Padding, Fusion
Resized Crop	Padding, Fusion	Padding, Fusion
Sobel	Padding	Padding
Zero-DCE	Padding	Padding
Edge Enhancement	Padding, Fusion	Padding, Fusion
전체	Supported Ops로 구현	

Evaluation

성능 측정 환경 (Experimental Settings)

- Latency 측정 범위: 메모리에 이미지가 로드된 시점부터 해당 함수 실행이 완료될 때까지의 지연 시간을 측정
- 출력 데이터 형식: 모든 함수의 최종 출력 데이터 형식은 Float으로 고정 (Skeleton Code)
- NPU의 경우에는 필수적으로 compile이 필요한데 latency가 측정되는 시점에서 compile을 하는 것은 옳지 않으므로 미리 compile 후 메모리에 Load

추가 구현 함수

- Zero-DCE의 경우 모델학습을 할 때 (3,224,224)로 padding을 했기 때문에 CPU에서도 Padding 적용
- Edge Enhancement의 경우는 Gaussian과 같이 여러 argument 들에 대해서 측정

Kernel size	3	3	5	5
alpha	1.0	0.8	0.5	1.2

성능 측정 환경 (Experimental Settings)

고려사항: CPU 성능이 GPU, ATOM, Warboy 세 환경 모두 다름

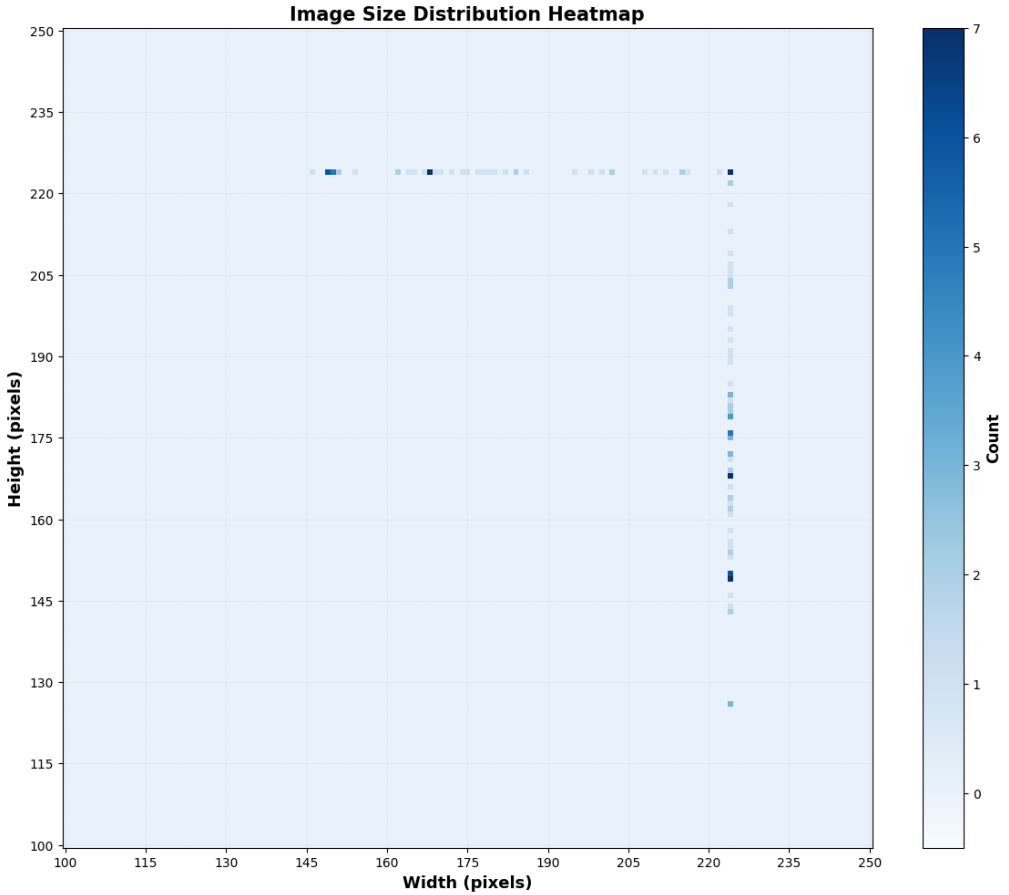
- CPU [INTEL(R) XEON(R) GOLD 6542Y]
- NVIDIA-A100 + Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz
- ATOM + INTEL(R) XEON(R) GOLD 6542Y
- Warboy + Intel(R) Xeon(R) Gold 5420+

	# of CPUs	CPU freq[MHz]	PCIe Bandwidth
INTEL(R) XEON(R) GOLD 6542Y	96	800/4100	128GB/s
Intel(R) Xeon(R) Gold 5420+	112	800/4100	32GB/s
Intel(R) Xeon(R) Gold 6338 CPU @ 2.00GHz	128	800/3200	32GB/s

성능 측정 대상 특성 (Workload Characteristics)

고려사항: Test Image가 대부분 작음

- HtoD/DtoH data transfer > Compute
- PCIe Bound



Quantitative Result - MSE

- ATOM의 경우 MSE가 모두 굉장히 낮음
- Warboy의 경우에는 Quantization이 필수적이라 상대적으로 MSE가 높음

	CPU(Baseline)	ATOM	Warboy	GPU
Flip	-	0.00000000	0.00111226	0.00000000
Gaussian Blur	-	0.00000014	0.00000529	0.00000000
Color Jitter	-	0.00000213	0.00023436	0.00004064
Resized Crop	-	0.00000115	0.00012706	0.00004064
Sobel	-	0.00000115	0.02926639	0.00004064
Zero-DCE	-	0.00000044	0.00000637	0.00004064
Edge Enhancement	-	0.00000211	0.00039226	0.00004064

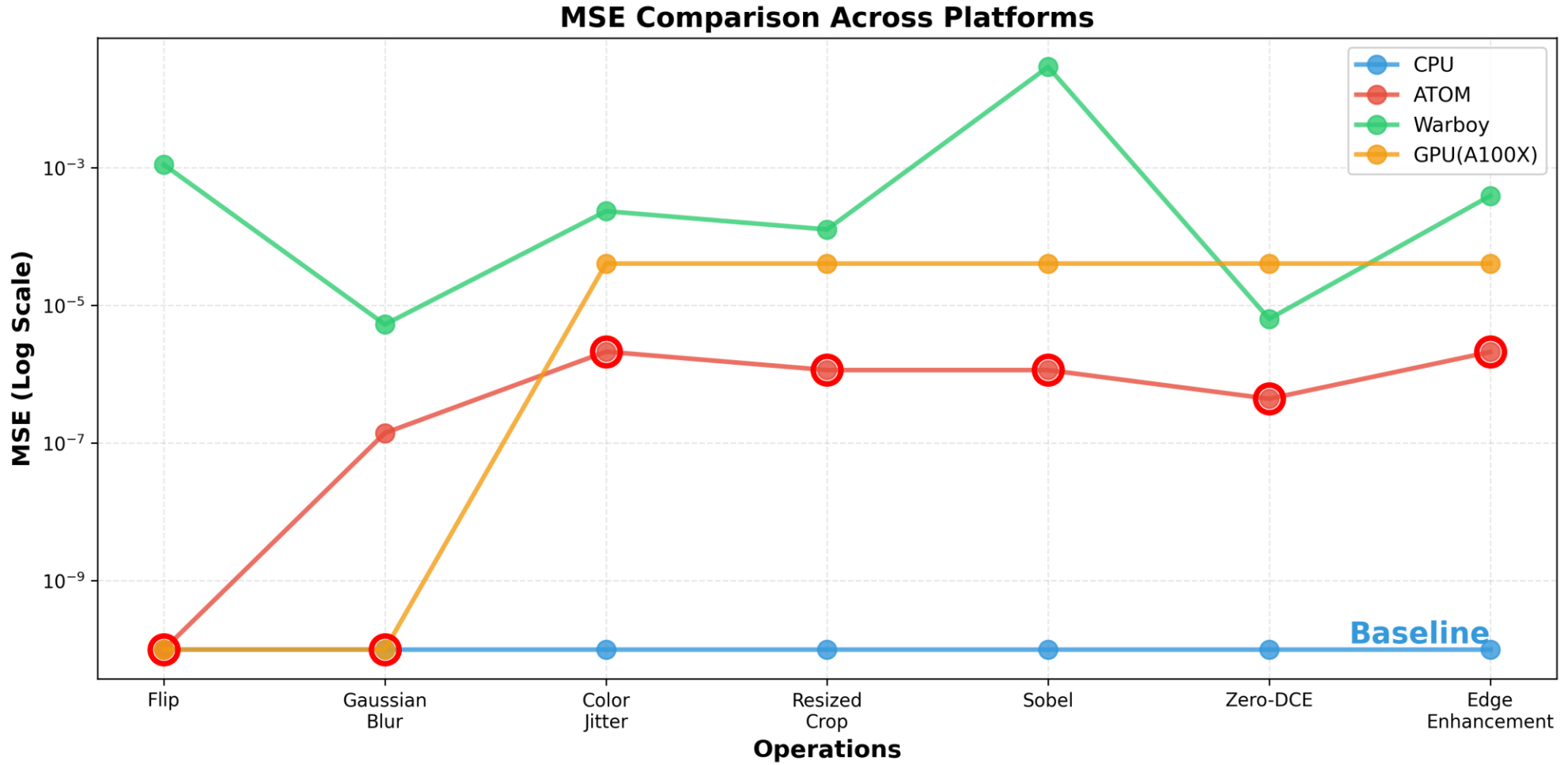
Quantitative Result – Latency(ms)

- Gaussian Blur 같은 함수는 CPU 전/후처리를 포함해도 1ms 이내로 처리가 완료됨
- 대부분의 기본 전처리 함수들은 10ms 안쪽으로 모든 작업이 끝나는 성능을 보임

	CPU(Baseline)	ATOM	Warboy	GPU
Flip	**0.067**	0.483	**18.517**	1.295
Gaussian Blur	1.471	0.962	**1.704**	**0.712**
Color Jitter	**0.670**	2.189	**5.380**	2.035
Resized Crop	**0.859**	2.007	**10.201**	1.418
Sobel	**0.797**	1.623	**2.366**	1.449
Zero-DCE	**9.267**	3.509	7.255	**1.973**
Edge Enhancement	**2.496**	1.828	2.004	**1.219**

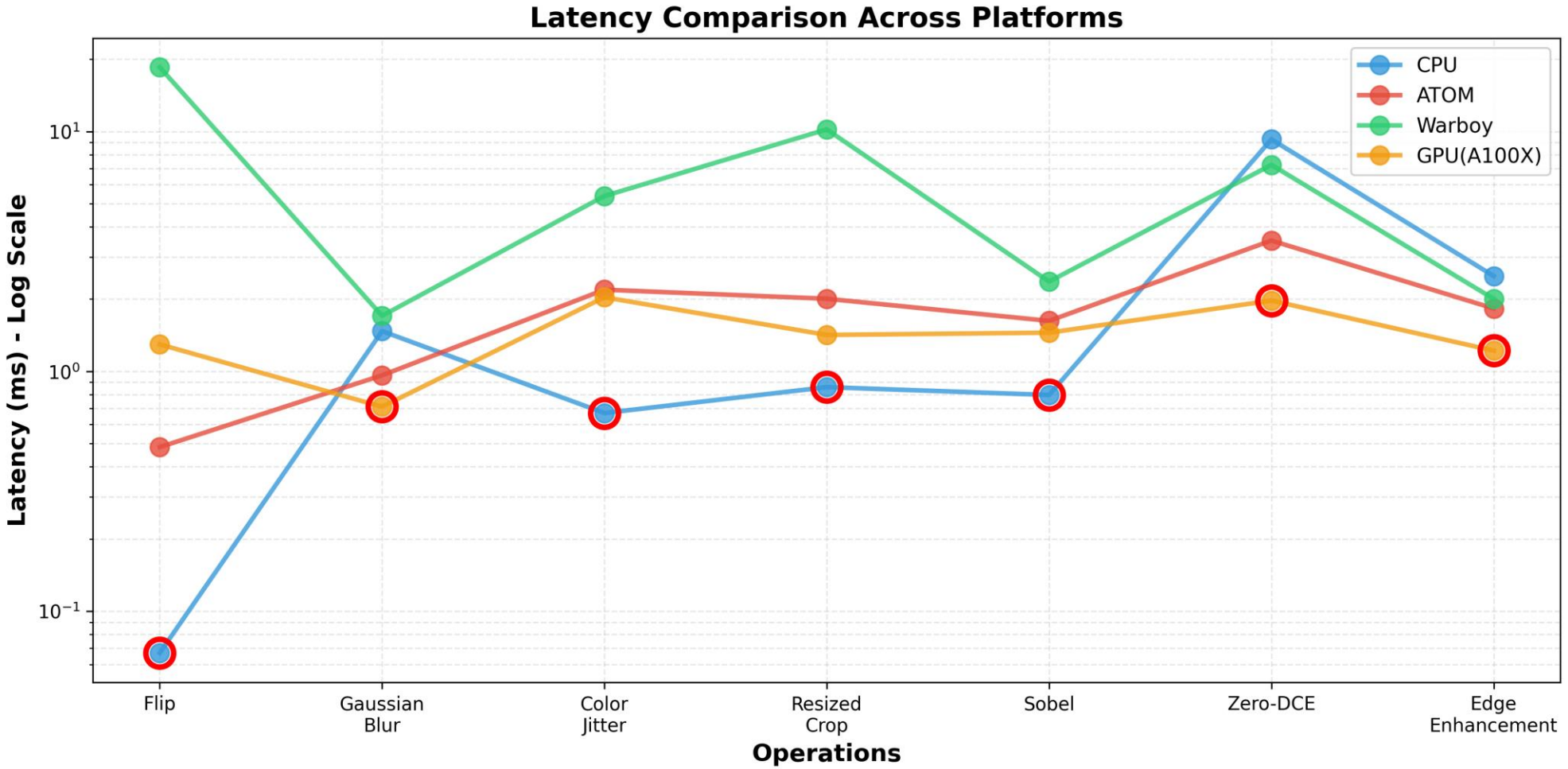
Quantitative Result - MSE

- ATOM의 경우 MSE가 굉장히 낮음 - FP16 지원
- Warboy의 경우 상대적으로 MSE가 높음 – Integer Quantization 필수



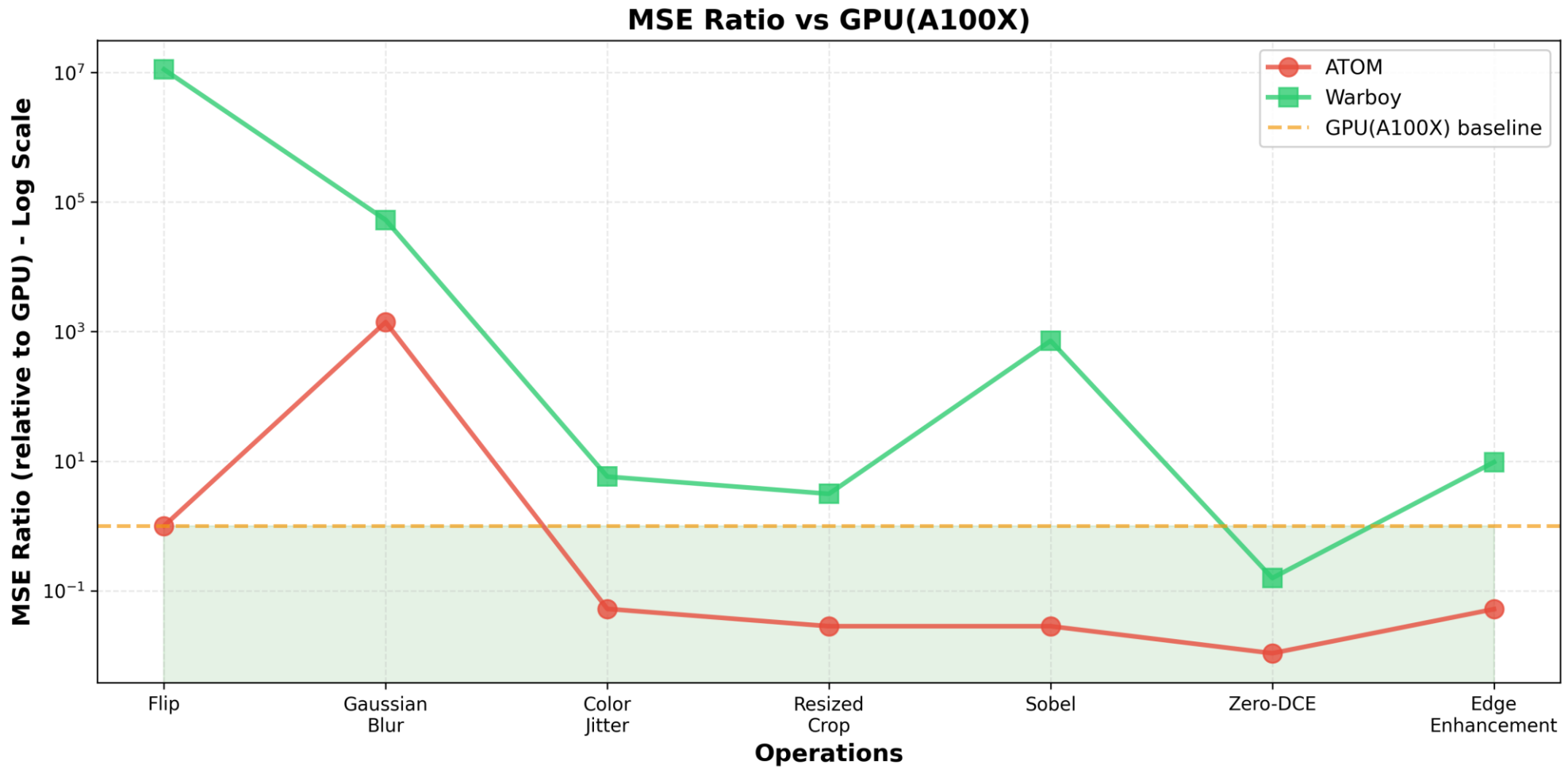
Quantitative Result - Latency

- 대부분의 Test Image가 작아 CPU에서도 큰 시간차이가 없음
- 연산량이 적기때문에 HtoD/DtoH 데이터 전송이 오래 걸림



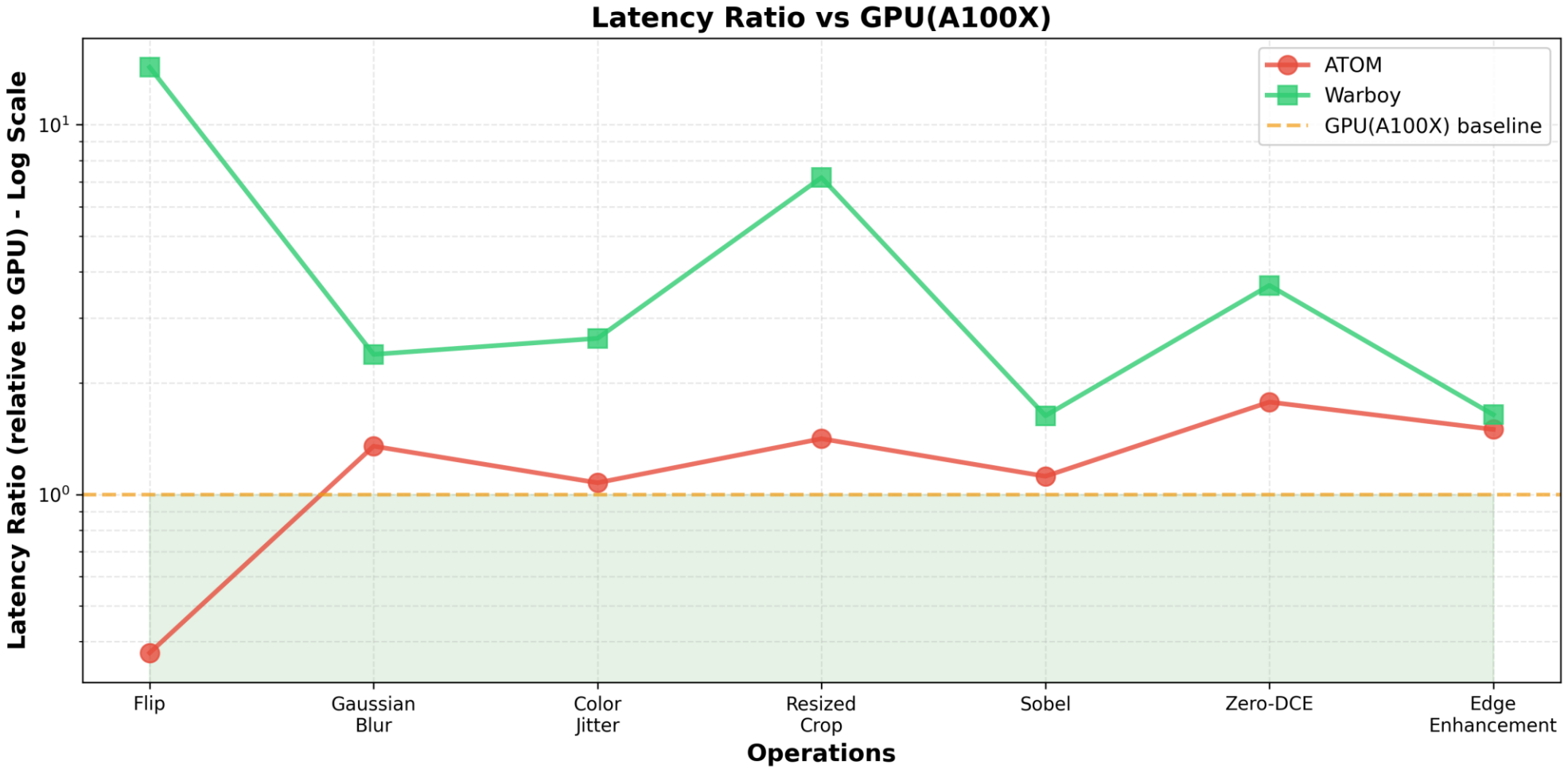
GPU relative Result - MSE

- Warboy의 경우에는 Integer Quantization이 필수적이라 Flip에서 MSE가 높음
- ATOM의 경우 FP16을 지원하여 Flip에서 CPU, GPU와 차이가 없음



GPU relative Result - Latency

- ATOM, Warboy 모두 GPU 대비 Throughput은 낮음
- NVIDIA-A100와 ATOM, Warboy의 가격차이는 10배

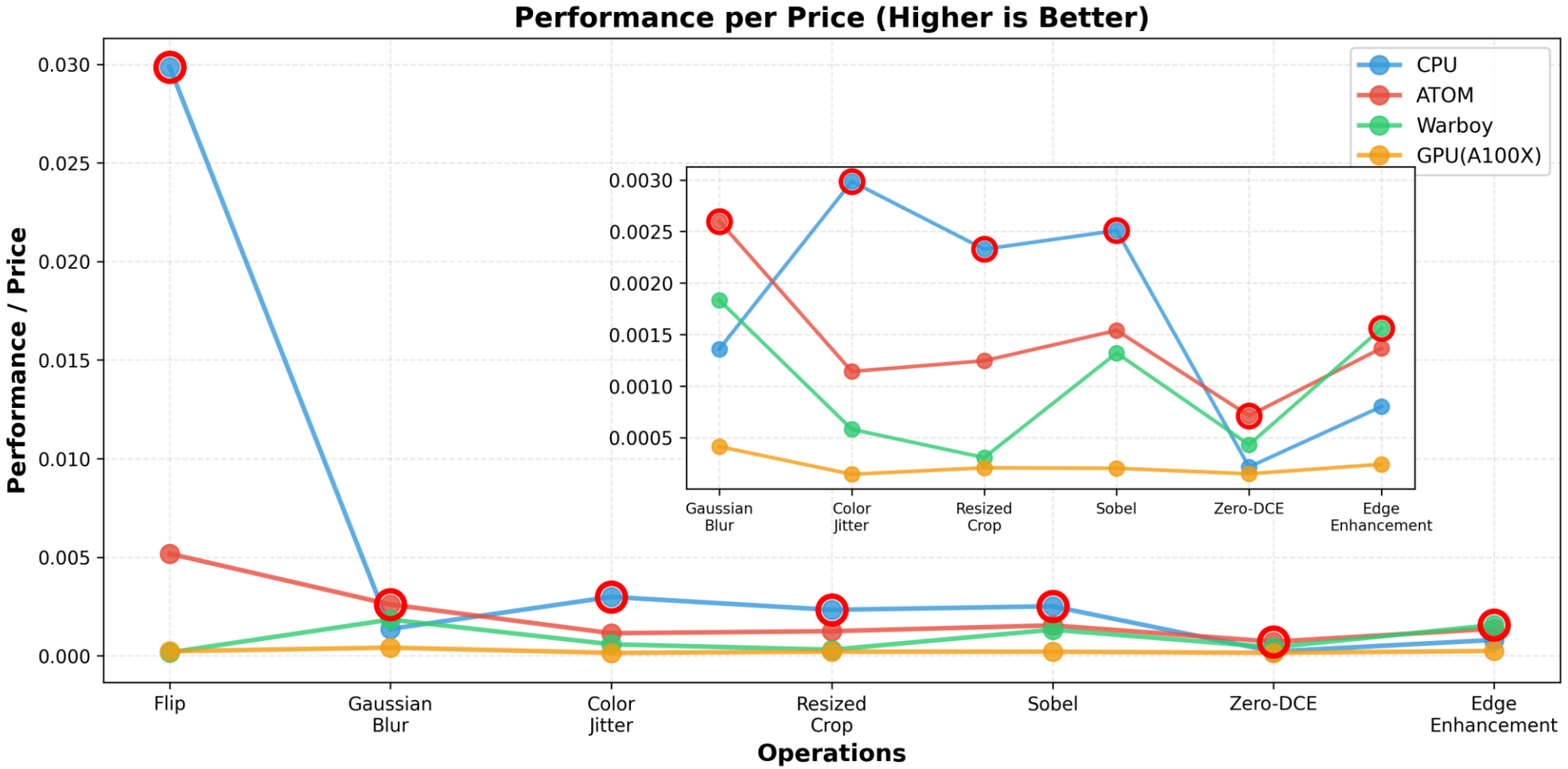


Cost Efficiency - MSE

- 대부분의 연산에서 비용 효율은 ATOM이 가장 좋음
- Warboy는 Integer 연산만 지원하는 것의 영향이 큼

Cost Efficiency - Latency

- Flip의 경우 단순한 연산이기때문에 CPU에서 수행하는 것이 유리함.
- Test 대상의 이미지가 작아 CPU에서 연산을 진행하는 것이 비용효율적임

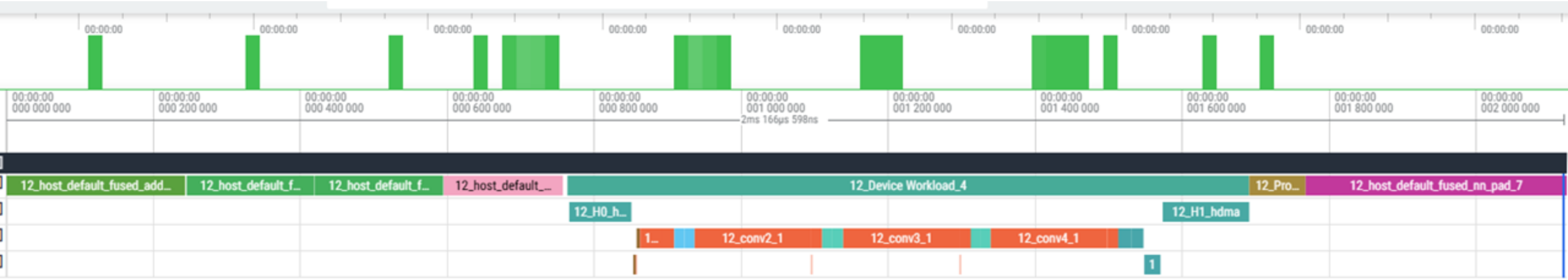


Conclusion

Result Analysis: ATOM (Zero-DCE)

cpu 런타임: 1.213ms

npu 런타임: 0.927ms(IO포함)



성능 병목 원인

- CPU 런타임의 주요 병목: image padding & transpose
- NPU 런타임의 주요 병목: convolution

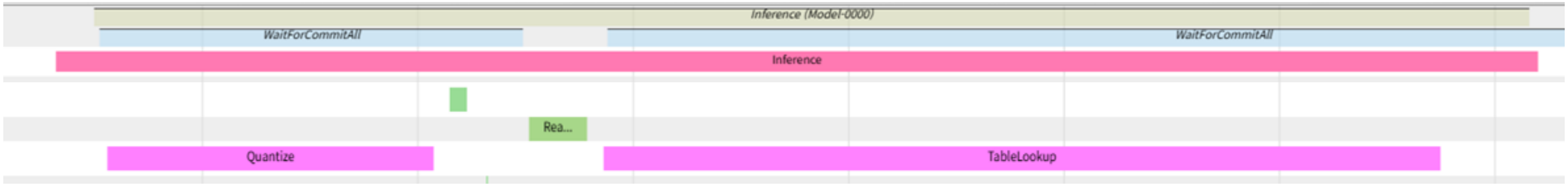
Result Analysis: Warboy (Zero-DCE)

cpu 런타임: 6.880ms

npu 런타임: 0.158ms(IO 포함)

Dequantization cost: 3.884 ms

Quantization cost: 1.515ms



성능 병목 원인

- CPU 런타임의 주요 병목: Quantization/Dequantization + IO + Unsurpported Opperation
- Warboy의 경우 CPU에서의 Quantization/Dequantization cost가 주요 병목

핵심 결과 요약 및 향후 활용 방안

연산 유형	예시	최적 Hardware
Memory Bound	Flip, Resize	CPU
Compute Bound	ZeroDCE, Gaussian Blur	GPU, NPU

- 10ms 이내의 빠른 처리 속도는 SLO (Service Level Objective)를 손쉽게 충족시키며, 이는 시스템이 Real-Time 목표를 달성할 수 있음을 의미
- 가벼운 연산들은 CPU에서 하고 무거운 연산(딥러닝 모델을 통한 전처리 등)을 NPU에서 활용
- 자율주행, 드론, 스마트 팩토리, 로봇 등 지연 시간에 매우 민감한 (Latency-sensitive) 분야에서의 활용도가 극대화