https://doi.org/10.7840/kics.2025.50.2.357

고해상도 이미지 처리를 위한 PIM 기반 JPEG 디코더 병렬화 기법 구현

김 지 은◆, 남 덕 윤°

Implementation of Parallelization Techniques for PIM-Based JPEG Decoder in High-Resolution Image Processing

Jieun Kim◆, Dukyun Nam°

요 약

JPEG은 이미지 크기를 효과적으로 줄일 수 있어 다양한 응용 분야에서 널리 사용되는 디지털 이미지 형식이다. JPEG 디코딩은 압축된 이미지 데이터를 복원하여 디스플레이하거나 후처리하기 위해 필요한 과정으로, 실시간 애플리케이션과 딥러닝과 같은 대량 이미지 처리 분야에서 중요한 역할을 한다. 그러나 JPEG 디코딩은 성능의 병목 지점이 될 수 있어, 이를 극복하기 위한 다양한 가속기 기반 병렬 처리 기법들이 제안되었다. 최근에는 Processing-in-Memory (PIM) 아키텍처를 활용한 연구들이 주목받고 있으며, 기존 연구인 PIM 기반 JPEG 디코더는 하드웨어 스레드와 AAN 알고리즘을 적용하여 성능 개선을 도모하였으나, DPU당 제한된 메모리 용량 및 DPU 간 통신 부재로 처리 가능한 이미지 해상도에 한계가 있었다. 본 연구에서는 JPEG 디코딩 파이프라인을 수정하여 이러한 한계를 극복하는 방안을 제안한다. 허프만 디코딩을 호스트 CPU에서 수행함으로써 DPU 간 통신의 필요성을 없애고, MCU (Minimum Coded Unit) 간 데이터 의존성 없이 병렬 처리가 가능하도록 설계하였다. UPMEM PIM 서버에서 고해상도 이미지 데이터 세트를 사용하여 실험을 수행한 결과, 본 연구에서 구현한 디코더는 기존 PIM 디코더가 처리할 수 없었던 고해상도 이미지를 디코딩할 수 있음을 확인할 수 있었다.

키워드 : 병렬 JPEG 디코딩, Processing-in-Memory (PIM), 고화질 이미지 처리
Key Words : Parallel JPEG Decoding, Processing-in-Memory (PIM), High-resolution Image Processing

ABSTRACT

JPEG is a digital image format widely used in various applications due to its ability to effectively reduce image file size. JPEG decoding involves restoring compressed image data for display or post-processing and plays a crucial role in fields such as real-time applications and deep learning, where large-scale image processing is required. However, JPEG decoding can become a performance bottleneck, prompting the development of various accelerator-based parallel processing techniques. Recently, research utilizing Processing-in-Memory (PIM) architectures has gained attention. Existing studies on PIM-based JPEG decoders have applied hardware threads and the AAN algorithm to improve performance, but limitations such as constrained memory capacity per DPU and lack of inter-DPU communication have restricted the resolution of image that can be processed. In this paper, we propose a modified JPEG decoding pipeline to overcome these limitations. By performing Huffman

※ 이 논문은 2022학년도 경북대학교 신임교원 정착연구비에 의하여 연구되었음.
◆ First Author : Kyungpook National University, School of Computer Science and Engineering, kimjea1999@knu.ac.kr, 학생회원
° Corresponding Author : Kyungpook National University, School of Computer Science and Engineering, dynam@knu.ac.kr, 정회원
논문번호 : 202409-207-C-RE.R1, Received September 12, 2024; Revised November 27, 2024; Accepted December 5, 2024

decoding in the host CPU, the need for inter-DPU communication is eliminated, enabling parallel processing without data dependencies between MCUs (Minimum Coded Units). Experiments conducted on the UPMEM PIM server using a high-resolution image dataset show that the proposed decoder can successfully decode high-resolution images that the existing PIM decoder could not process.

Ⅰ. 서 론

JPEG (Joint Photographic Expert Group) 표준은 디지털 이미지의 손실 압축을 위해 1992년에 ISO/IEC 10918-1로 처음 개발된 이후, 웹 그래픽, 디지털카메라, 이미지 저장 및 전송 등 다양한 분야에서 널리 사용되고 있다[1]. JPEG은 허프만 인코딩과 양자화를 비롯한 여러 압축 기법을 활용하며, 이는 이미지의 시각적 품질을 유지하면서 파일 크기를 줄이는 데에 효과적이다. 압축된 이미지를 디스플레이 장치에 표시하거나 편집할 때는 디코딩이 선행되어야 하며, 이는 특히 대량의 이미지를 처리해야 하는 실시간 애플리케이션이나 딥러닝과 같은 분야에서 중요한 역할을 한다.

JPEG 디코딩은 압축된 이미지 데이터를 해제하여 각 픽셀의 RGB 값을 복원하는 작업이다. 실시간 애플리케이션이나 대량의 이미지를 처리해야 하는 환경에서는 JPEG 디코딩 속도가 성능의 병목 지점이 될 수 있다. 이에 따라 JPEG 디코딩 가속을 목적으로 한 연구들이 이루어졌으며, SIMD 인스트럭션을 사용하여 여러 데이터를 동시에 처리하거나, GPU, FPGA, ASIC 등 다양한 하드웨어 가속기를 활용하여 디코딩을 병렬로 수행하는 기법이 제안된 바 있다[2-7]. 가속기를 활용한 연구에서는 JPEG 디코딩 단계 중 허프만 디코딩과 DC 계수 조정 단계의 병렬화를 구현하는 것이 주요 과제이다. JPEG 이미지는 가변 길이 코드인 허프만 코드로 압축되어 있기 때문에 코드워드의 경계를 디코딩 전에 미리 알 수 없고, 이로 인해 병렬 처리 유닛에 임의로 할당된 데이터 블록은 온전히 디코딩될 수 없는 경우가 대부분이다. 따라서 가속기를 활용한 연구에서는 (1) 호스트 프로세서에서 사전 스캔을 통해 허프만 코드워드를 미리 식별하여 가속기에 분배하거나, (2) 허프만 코드의 자기 동기화 (Self-synchronization) 특성을 이용한다. 자기 동기화 특성이란, 허프만 코드의 어느 위치에서 디코딩을 시작하더라도, 동기화 지점부터는 올바른 디코딩 결과가 나오는 특성이다. 여러 병렬 처리 유닛이 서로 다른 위치에서 디코딩을 시작할 때, 동기화 지점은 두 처리 유닛이 같은 디코딩 결과를 산출하는 지점으로 식별할 수 있다. 따라서, 이를 구현하기 위해서 병렬 처리 유닛 사이의 통신이 필요하다. DC 계수 조정 단계는 허프만 디코딩을 통해 생성된 MCU의 DC 계수 값을 계산하는 단계로, 이미지의 MCU 순서대로 DC 계수를 누적시켜 구하기 때문에 병렬로 계산하기 어려운 특성을 가진다.

Nider et al.의 연구[2] (이하 'PIM-JPEG')는 현재 상용화된 유일한 제품인 UPMEM사의 PIM (Processing-In-Memory) 하드웨어[8]를 활용하여 JPEG 디코딩 병렬화를 시도한 연구이다. 이 연구에서 구현한 디코더는 허프만 디코딩을 병렬로, DC 계수 조정을 순차적으로 수행하는 방식을 채택했다. 또, PIM 코어 (DPU)의 제한적인 연산 능력을 극복하기 위해 하드웨어 스레드인 태스크릿 (tasklet)을 활용하고, AAN 알고리즘[9] 등 계산 최적화 기법을 적용하여 성능 향상을 도모하였다. UPMEM PIM은 DPU 간 직접적인 통신을 지원하지 않으므로, PIM-JPEG은 이러한 구조적 제약하에서 허프만 디코딩 병렬화를 구현하기 위해 이미지 1장에 최대 1개의 DPU만 할당하도록 구현되었다. 그러나, DPU가 접근할 수 있는 메모리 공간이 64MB로 제한되므로, 이러한 DPU 할당 방식은 처리할 수 있는 이미지 해상도가 한정적이라는 문제가 있다.

따라서, 본 논문에서는 JPEG 디코딩 파이프라인을 수정하여 고해상도 이미지를 처리하는 방법을 제안한다. 제안된 디코더는 PIM-JPEG과 달리, 허프만 디코딩과 DC 계수 조정을 호스트 CPU에서 순차적으로 수행한다. (그림 1) DC 계수 조정이 끝난 MCU는 다른 MCU의 값에 영향을 받지 않으므로, 여러 DPU에서 병렬적으로 처리될 수 있다. 이러한 구조에서 이미지에 여러 개의 DPU를 할당함으로써, PIM-JPEG의 메모리 용량 제약을 완화할 수 있다. 실험은 UPMEM PIM 서버에서 고해상도 이미지 데이터 세트를 사용하여 수행되었으며, 그 결과 처리 가능한 최대 해상도가 증가했음을 확인하였다. 또, PIM-JPEG과 본 논문에서 제안한 디코더의 실행 시간을 측정한 결과, 허프만 디코딩의 병렬화보다 여러 DPU를 활용하는 방식이 더 나은 성능을 제공하는 것으로 확인되었다. 마지막으로, 실행 시간에서 디코더의 각 실행 단계 (MCU 생성, 데이터 이동, DPU 실행, BMP 작성)가 차지하는 비율을 분석하여 주요 병목 지점을 파악하였고, 이를 기반으로 효율적인 PIM 기반 병렬 디코딩 방식을 논의했다.

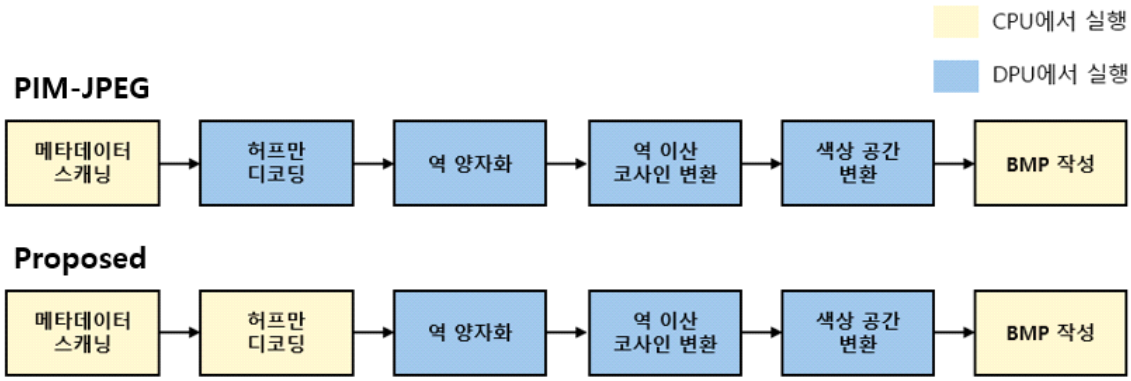

그림 1. PIM-JPEG과 제안된 디코더의 디코딩 파이프라인 비교
Fig. 1. Comparison of the decoding pipeline of PIM-JPEG and the proposed decoder

Ⅱ. 배경 및 관련 연구

2.1 허프만 디코딩과 DC 계수 조정

인코딩되지 않은 이미지는 R, G, B값을 갖는 픽셀들로 이루어져 있다. 이를 JPEG 포맷으로 인코딩하는 과정을 간략하게 설명하면 다음과 같다.

첫 번째, 이미지의 픽셀을 8x8 픽셀 블록인 MCU (Minimum Coded Unit) 단위로 나눈다. 두 번째, 각 픽셀이 가지는 RGB 값을 YCbCr 색상 공간 값으로 변환한다. 이때, MCU는 채널 개수만큼 늘어난다. (예를 들어, 세 가지 채널을 가지는 컬러 이미지는 8x8 픽셀 블록마다 3개의 MCU를 가진다) 세 번째, 모든 MCU에 이산 코사인 변환(DCT)을 수행하여 픽셀이 가지는 값을 주파수 영역으로 변환한다. 네 번째, 모든 MCU에 양자화를 수행하여 각 픽셀이 가지는 값의 크기를 줄인다. 다섯 번째, 허프만 인코딩을 통해 값을 표현하는 비트 수를 줄인다. 이 과정에서 사용된 양자화 테이블과 허프만 테이블은 JPEG 헤더에 저장된다.

JPEG 디코딩 과정은 인코딩의 역순으로 진행된다. JPEG 포맷으로 인코딩된 이미지는 허프만 인코딩된 비트스트림으로 나타난다. JPEG 디코더는 헤더에 저장된 허프만 테이블을 이용하여 허프만 코드를 정숫값으로 디코딩하며, 연속된 64개의 값이 1개의 MCU가 된다. 1개의 MCU는 DC 계수와 AC 계수로 구성되며, DC 계수는 블록의 평균 색상을 나타내고 AC 계수는 세부적인 변화, 즉 텍스처나 엣지 정보를 표현한다. DC 계수는 해당 MCU의 평균값을 의미한다. 이웃한 MCU는 서로 유사한 경우가 많아, DC 계수의 변동이 작다. 이러한 점을 이용하여 DC 계수에는 이전 값과의 차를 저장하는 방식인 차분 코딩(differential coding)을 적용하여 압축 효율을 높인다. 따라서, DC 계수를 계산할 때는 이전 MCU의 DC 계수 값을 더해야 하며, 첫 번째 MCU의 DC 계수 값에는 0을 더한다. 무선 통신 환경 등, 손실이 발생할 수 있는 상황에서의 오류 정정을 위해 재시작 간격(restart interval)을 설정하기도 한다. 재시작 간격은 DC 계수에 더하는 값을 0으로 초기화하는 주기를 의미하며, MCU 개수 단위이다. 그림 2는 재시작 간격에 따른 DC 계수 조정 방식의 차이를 나타낸 것이다.

2.2 UPMEM PIM

본 절에서는 UPMEM PIM 구조와 특성 분석 (characterization) 연구에서 제시되는 프로그래밍 권장 사항에 대해 간략히 설명한다. UPMEM PIM은 8개의 PIM 지원 칩(PIM-enabled chip)으로 구성된 랭크들로 이루어져 있으며, 각 칩에는 8개의 DRAM Processing Units (DPU)가 있다. DPU 코어는 순차(in-order) 실행 방식의 32비트 RISC 프로세서로, 24개의 하드웨어 스레드인 태스크릿을 통해 멀티스레딩을 지원하며, mutex, barrier, semaphore와 같은 동기화 메커니즘을 제공한다. 각 DPU에는 전담 메모리 영역인 Instruction RAM (IRAM), Working RAM (WRAM), Main RAM (MRAM)이 있다. IRAM은 DPU가 실행할 프로그램이 저장되는 영역이며, WRAM은 스택 및 힙 데이터를 저장하고 모든 태스크릿이 공유하는 영역이다. MRAM은 64MB 용량의 DRAM 슬라이스로, 호스트 CPU와 DPU 사이 통신에 사용된다. DPU 코어는 전담 메모리 영역에만 접근할 수 있으며, 다른 DPU의 메모리에 접근하기 위해서는 호스트 CPU를 거쳐야 한다. 따라서, DPU 간 통신은 상당한 오버헤드를 발생시킨다.

UPMEM PIM 프로그램은 호스트 측에서 실행되는 프로그램과 여러 DPU에서 SPMD (Single Program Multiple Data) 모델로 실행되는 DPU 프로그램으로 구

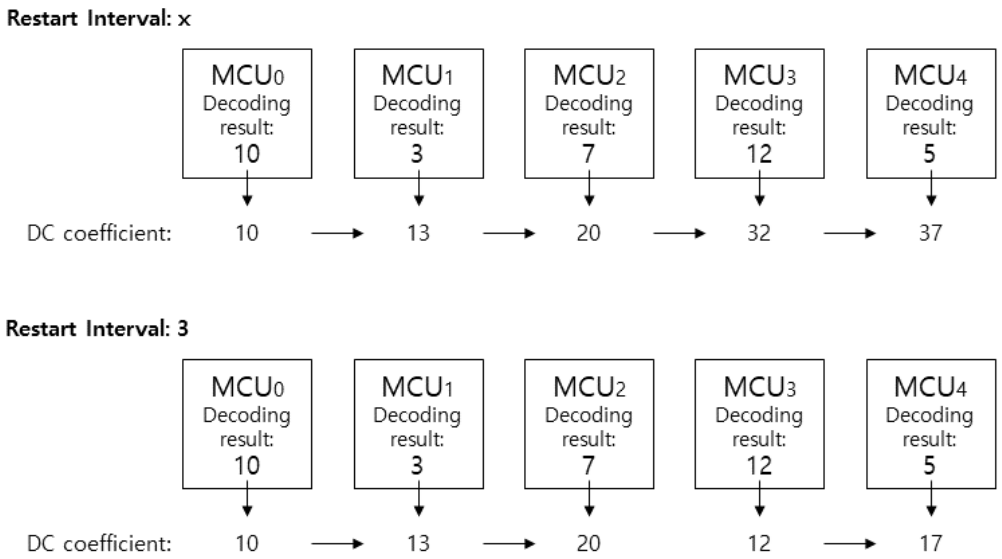

그림 2. 재시작 간격에 따른 DC 계수 조정 방식
Fig. 2. DC coefficient adjustment with and without restart interval

성된다[10]. 기존의 UPMEM PIM 특성 분석 연구에서는 실험을 통해 DPU 프로그램 작성 시의 권장 사항을 제시하고 있다. 권장 사항들은 DPU의 제한적인 연산 능력을 극복하기 위해 여러 연구에서 따르고 있으며, 그 중 핵심적인 두 가지 사항은 (1) 여러 개의 태스크릿 사용과 (2) 계산 최적화이다[11]. 첫째, 여러 태스크릿을 사용하는 것은 DPU 프로그램의 병렬성을 높여 실행 시간을 단축하는 데 기여한다. 알려진 최적의 태스크릿 개수는 11개로, 이는 DPU의 14단계 파이프라인 중 마지막 세 단계만 병렬적으로 실행할 수 있어 11개 이상의 태스크릿을 사용해야 파이프라인을 완전히 활용할 수 있기 때문이다. 둘째, 계산 최적화는 DPU 코어가 지원하지 않는 연산을 지원하는 연산으로 대체하는 것으로, 이 또한 실행 시간을 단축하는 데 도움을 준다. DPU 코어는 32/64비트 정수 곱셈 및 나눗셈, 부동소수점 연산을 하드웨어 수준에서 지원하지 않고 소프트웨어에서 에뮬레이션하는데, 이에 따라 연산 속도가 크게 저하된다. 따라서 UPMEM PIM을 활용한 연구에서는 부동소수점 연산을 정수 연산으로 대체하는 "adds and shifts" 기법이 자주 사용된다.

2.3 가속기를 활용한 병렬 JPEG 디코딩 연구

JPEG 디코딩 성능을 향상시키기 위해 가속기를 활용한 병렬 처리 기법 연구는 지속적으로 이루어져 왔다. Sodsong et al.는 CPU를 사용해 비트스트림에서 데이터 블록의 위치를 신속하게 스캔하고, GPU에서 각 블록을 병렬적으로 디코딩하는 방법을 제안했다[3]. Wang et al.의 연구는 두 가지 사전 스캔(pre-scan) 방법을 통해 JPEG을 임의의 블록 단위로 병렬 디코딩이 가능하게 변환하고, 이를 GPU에서 병렬 처리하는 방식을 제시했다[4]. Cheng et al.은 효율적인 FPGA 기반 JPEG 디코더를 설계하고, 딥러닝 워크로드의 데이터 전처리를 FPGA 디코더로 오프로딩함으로써 데이터 스톨(Data stalls)을 개선할 수 있음을 보였다[5]. 이들 연구는 공통으로 허프만 데이터의 병렬 디코딩 방법을 탐구했다. Klein과 Wiseman은 멀티 코어 프로세서에서 허프만 코드의 특징을 이용하여 허프만 인코딩된 비트스트림을 병렬적으로 디코딩하는 방법을 제안하고, 이를 JPEG 이미지 디코딩에 적용한 결과를 보였다[12]. 허프만 코드는 잘못된 위치에서 디코딩을 시작해도, 디코딩이 올바르게 수행되기 시작하는 지점(동기화 지점)이 존재하는데, 이를 허프만 코드의 자기 동기화 특성이라 부른다. 여러 개의 프로세서가 허프만 코드의 서로 다른 지점에서 디코딩을 시작하면, 각 병렬 처리 유닛 모두 동기화 지점에 도달하게 되고, 동기화 지점은 두 프로세서의 디코딩 결과가 동일한 지점으로 식별할 수 있다. 동기화 지점 이전까지의 디코딩 결과는 잘못된 것으로 간주하고, 프로세서는 동기화 지점에 도달한 뒤에도 디코딩을 계속 수행하여 올바른 결과를 생성한다. 즉, 실질적인 디코딩은 동기화 지점부터 시작되는 것이다. 이러한 방식으로 허프만 디코딩 병렬화를 구현하는 경우, (1) 서로 다른 병렬 처리 유닛이 통신하여 동기화 지점을 식별해야 하고, (2) 동기화 지점 이후의 올바른 디코

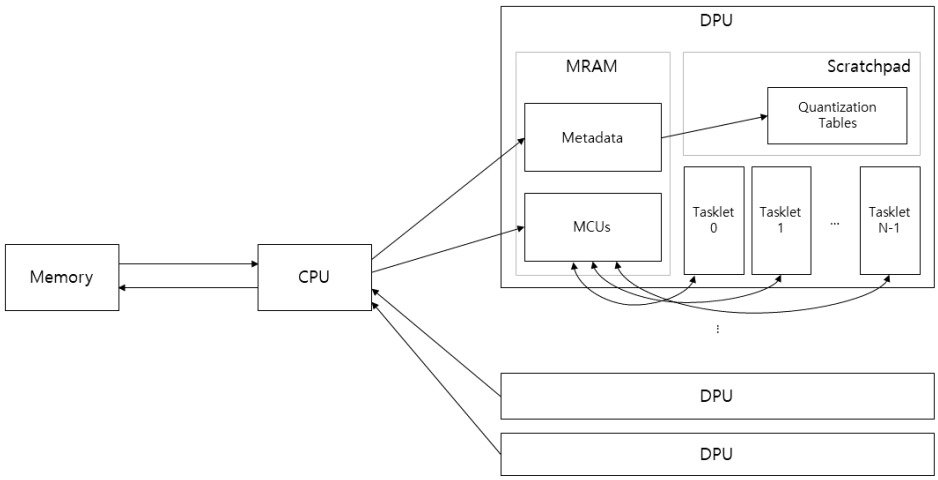

그림 3. 여러 DPU 코어를 활용하는 JPEG 디코더 구조
Fig. 3. JPEG decoder architecture utilizing multiple DPU cores

딩 결과를 원본 데이터와 같은 순서로 모아야 한다.

Nider et al.의 연구[2]는 현재 상용화된 유일한 PIM 하드웨어인 UPMEM PIM을 활용한 대규모 병렬 JPEG 디코딩의 타당성을 평가하였다. 이 연구에서 제안된 PIM-JPEG 디코더는 호스트 CPU에서 JPEG 메타데이터를 스캔한 후, JPEG 파일에 DPU 하나를 할당하여 처리한다. PIM-JPEG이 이러한 DPU 할당 방식을 채택한 이유는 허프만 디코딩을 병렬화하기 위해서이다. UPMEM PIM은 DPU 사이의 통신을 지원하지 않기 때문에, DPU 수준에서 허프만 디코딩을 병렬화하는 것은 부적합하다. 따라서, PIM-JPEG은 DPU의 하드웨어 스레드로 허프만 디코딩 병렬화를 구현하며, DC 계수 조정을 제외한 나머지 단계 또한 스레드 수준에서 병렬로 수행된다. 그러나 이러한 구조에서는 DPU가 처리할 수 있는 이미지 해상도가, DPU가 접근할 수 있는 메모리 영역인 MRAM의 크기로 제한되는 문제가 발생한다. 한 DPU에 부착된(attached) DRAM 크기는 64MB로, 대부분의 JPEG 파일보다 큰 용량을 가지고 있으나, 디코딩된 MCU는 64개의 2바이트 정수로 이루어져 있고 컬러 이미지는 3개의 채널을 가지므로, 8x8 픽셀 블록은 약 380바이트의 고정 크기를 차지하게 된다. 즉, JPEG 파일의 용량이 MRAM 용량보다 작더라도, 해상도가 일정 수준을 초과하면 처리할 수 없다.

본 연구에서는 이러한 한계를 해결하기 위해, 허프만 디코딩을 CPU에서 순차적으로 수행하도록 변경하여, 많은 DPU 코어를 효율적으로 활용하는 개선된 JPEG 디코더를 제안한다.

Ⅲ. 설계 및 구현

3.1 JPEG 디코딩 작업 분할 전략

JPEG 디코딩 과정을 호스트 CPU와 DPU 간에 분할하여 수행하도록 설계되었다. JPEG 디코딩 과정에서 전반부인 메타데이터 스캐닝과 허프만 디코딩은 호스트 CPU에서 수행하고, 후반부인 역 양자화, 역 이산 코사인 변환, 색상 공간 변환은 다수의 DPU에서 여러 태스크릿을 활용하여 병렬적으로 수행한다.

JPEG 디코딩의 초기 두 단계인 메타데이터 스캐닝과 허프만 디코딩을 DPU에서 병렬로 수행하지 않고 호스트 CPU에서 처리하는 이유는 이들 단계의 병렬화에 몇 가지 제약이 있기 때문이다. 메타데이터 스캐닝의 경우, 첫째로 각 데이터의 크기나 위치를 사전에 확인할 수 없고, 데이터의 형식과 처리 방법이 데이터 종류에 따라 상이하다. 둘째, 스캔한 데이터를 모든 프로세서가 공유해야 하지만, DPU 간의 직접적인 통신이 불가능하며, CPU를 경유해야 하므로 비효율적이다. 다음으로 허프만 디코딩의 병렬 처리 기법은 기존 연구[12]에 의해 제안되었으며, 여러 연구에서 이를 응용하고 있지만, 일부 이미지에서는 디코딩 오류가 발생하거나, 원본 이미지와 다른 결과를 출력하는 등 한계가 존재한다. 메타데이터 스캐닝과 허프만 디코딩 단계는 전체 실행 시간에서 차지하는 비중이 상대적으로 작을 뿐만 아니라 병렬 처리에 구조적 어려움이 있기 때문에, 본 연구에서는 이 단계를 호스트 CPU에서 처리하고, 이후 스캔된 비트스트림을 DPU에서 병렬로 디코딩하는 방식을 채택했다.

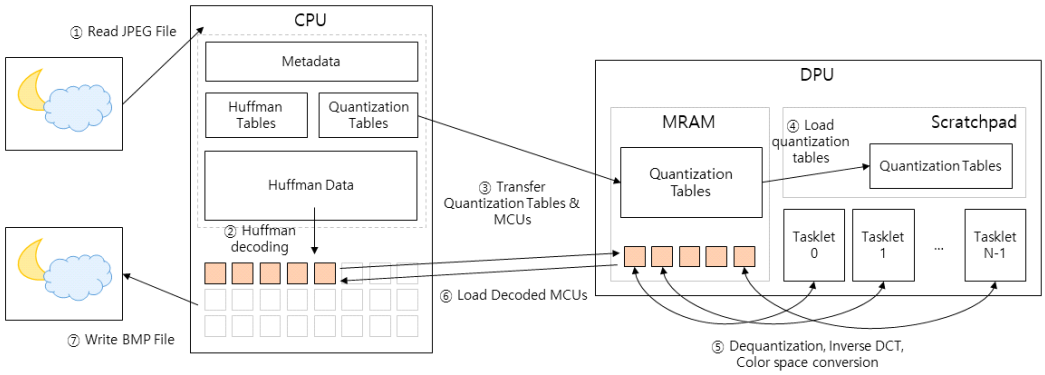

그림 4. 제안된 디코더의 실행 순서
Fig. 4. Execution sequence of the proposed decoder

JPEG 디코딩 과정의 후반부에 해당하는 역 양자화, 역 이산 코사인 변환, 색상 공간 변환은 데이터 병렬화(data parallelism)를 적용하기에 용이하고, 계산 복잡도가 높아 많은 연산량을 요구한다. 따라서 이들 단계를 다수의 DPU에서 여러 태스크릿을 활용하여 병렬로 수행함으로써 처리 속도를 개선할 수 있다.

그림 3은 본 논문에서 제안하는 PIM 기반 디코더의 구조도이다. 호스트 CPU는 JPEG 파일을 파싱한 후 DPU에 MCU를 분배하고, 디코딩이 완료된 MCU를 수집해 최종적으로 하나의 비트맵 파일로 재구성한다. PIM은 전송된 MCU에 대해 남은 디코딩 단계를 수행하며, 여러 태스크릿을 병렬로 활용하여 DPU 실행 시간을 최소화한다. 여기서, 이미지의 양자화 테이블과 같은 메타데이터는 WRAM에 로드되어 모든 태스크릿이 접근할 수 있도록 한다. 각 태스크릿은 자신에게 할당된 MCU에 대해 역 양자화, IDCT, 색상 공간 변환을 수행한 후, 연산이 끝난 MCU를 MRAM에 저장한다.

3.2 DPU 작업 최적화

DPU의 낮은 연산 능력을 극복하기 위해, 기존에 알려진 최적화 기법 두 가지를 적용했다[11]. 첫째, 여러 개의 태스크릿을 활용하여 디코딩 작업을 병렬적으로 수행하도록 구현했다. 각 DPU의 첫 번째 태스크릿은 MRAM에 전송된 메타데이터를 WRAM으로 로드하고, 각 태스크릿이 디코딩을 시작할 위치와 디코딩할 MCU 개수를 계산한다. 이 과정이 완료될 때까지 나머지 태스크릿들은 배리어(barrier) 동기화를 통해 대기 상태를 유지한다. 둘째, 계산 최적화를 통해 DPU가 지원하지 않는 연산을 간단한 연산으로 대체하고, 계산 횟수를 줄였다. 역 이산 코사인 변환과 색상 공간 변환은 많은 부동소수점 곱셈 연산을 포함한다. 따라서 'adds and shifts' 기법을 적용해 부동소수점 연산을 정수 연산으로 변환하고, 역 이산 코사인 변환의 최적화에 널리 사용되는 AAN (Arai, Agui, and Nakajima) 알고리즘[9]을 적용하여 곱셈 연산을 64회에서 11회로 줄였다.

3.3 디코딩 수행 순서

본 절에서는 제안한 디코더의 실행 순서를 그림 4를 통해 상세하게 설명한다. ① 디코더는 JPEG 파일을 읽고, 메타데이터와 이미지 데이터를 분리한다. ② 허프만 테이블을 이용하여 이미지 데이터에 허프만 디코딩과 DC 계수 조정을 수행하고, 그 결과로 MCU가 생성된다. ③ MCU 개수를 기반으로 할당할 DPU 개수를 결정하고, 각 DPU에 할당된 MCU와 메타데이터를 전송한다. 예를 들어, 이미지의 MCU 개수가 100이고, 시스템 상의 DPU 개수가 10이라면, 각 DPU에는 10개의 MCU가 전송된다. ④ DPU가 실행되면, 0번 태스크릿이 양자화 테이블을 WRAM에 로드하여 여러 태스크릿이 접근할 수 있도록 하고, 각 태스크릿이 처리할 MCU 범위를 계산한다. ⑤ 0번 태스크릿의 초기화 작업이 끝나면, 모든 태스크릿이 나머지 디코딩 단계(역 양자화, 역 이산 코사인 변환, 색상 공간 변환)를 병렬로 수행한다. ⑥ 모든 DPU의 실행이 끝나면, 호스트 CPU는 디코딩된 MCU를 모아 원래 순서대로 합친다. ⑦ 마지막으로, 디코딩이 끝난 MCU 값을 BMP 파일로 작성한다.

Ⅳ. 실 험

UPMEM SDK 2021.3.0을 이용해 구현하고 성능 평가를 수행했다. 실험 환경은 40개의 랭크(rank)로 구성된 UPMEM PIM 하드웨어 (64 DPUs per rank, 전체 2560 DPUs)와 Intel(R) Xeon(R) Silver 4215R CPU

그림 5. 처리 가능한 이미지 해상도 비교
Fig. 5. Comparison of decodable resolution

@ 3.20GHz로 구성되어 있다. 실험에 사용된 데이터 세트는 Kaggle에 공개된 'Clothing dataset'[13]으로, 5,762개의 JPEG 이미지로 이루어져 있으며 이미지의 평균 해상도는 2760x3349이다. DPU에서의 실행 시간을 분석하기 위해, UPMEM SDK에서 제공하는 perf-counter 함수를 사용하여 디코딩의 각 단계에서 소비되는 사이클 수를 측정하고[8], 이를 전체 소비 사이클에서 차지하는 비율로 환산했다.

4.1 처리 가능한 이미지 해상도 비교

데이터 세트의 모든 JPEG 파일에 대해 디코딩을 수행한 결과, 기존 논문에서 구현한 디코더는 643개(데이터 세트의 11.16%), 본 논문에서 구현한 디코더는 5,691개(데이터 세트의 98.77%)의 JPEG 이미지를 디코딩할 수 있었다. 디코딩이 불가능한 경우는 (1) 지원하지 않는 유형의 JPEG 파일인 경우 (2) JPEG 파일의 해상도 또는 용량이 큰 경우 (3) DPU fault가 발생한 경우의 세 가지 유형으로 나뉘었다.

그림 5는 두 디코더가 디코딩에 성공한 이미지 중 해상도가 가장 큰 이미지의 해상도 차이를 비교한 예시이다. 기존 PIM 디코더가 디코딩한 이미지의 최대 해상도는 1796x3070이고, 본 논문에서 구현한 디코더는 5120x3840으로 나타났다. 이를 MCU 개수로 환산하면 약 3.57배로, 해상도 제약이 크게 완화되었음을 확인할 수 있다.

4.2 실행 시간 분석

데이터 세트에서 두 디코더가 모두 디코딩할 수 있는 643개의 이미지를 각각 디코딩하여 실행 시간을 측정했다. 그림 6에서 볼 수 있듯이, 본 논문에서 구현한 디코더가 모든 경우에서 PIM-JPEG보다 실행 시간이

그림 6. 이미지 해상도에 따른 디코딩 시간
Fig. 6. Execution time according to image resolution

짧았으며, 실행 시간의 약 70~90%를 단축할 수 있었다. 이는 허프만 디코딩 병렬화보다 여러 개의 DPU를 활용한 디코딩 시간 단축이 더 효율적일 수 있음을 보여준다. 또, 이미지 해상도 증가에 따른 실행 시간 변화 폭도 차이를 보였는데, PIM-JPEG은 실행 시간의 최댓값이 최솟값의 약 3.7배였으나, 본 논문에서 구현한 디코더는 1.5배에 그쳤다. 즉, 해상도에 비례하여 DPU를 할당하는 방식이 실행 시간 측면에서 이점이 있다.

표 1은 기존 디코더와 본 논문에서 구현한 디코더의 평균 실행 시간 중 각 단계가 차지하는 시간을 정리한 것이다. MCU 디코딩은 메타데이터 스캐닝과 허프만 디코딩, DC 계수 조정에 걸린 시간을 합한 것이다. 기존 디코더는 이러한 작업을 DPU에서 수행하므로, 해당 시간이 DPU 실행 시간에 포함되었다.

기존 디코더는 DPU 실행 시간이 99.14%로 매우 크게 나타났는데, 이미지를 하나의 DPU로 처리하므로 DPU당 연산량이 비교적 크기 때문이다. 본 논문에서 구현한 디코더는 이미지 해상도에 비례하여 DPU를 할당했고, 그 결과 DPU 실행 시간이 전체 실행 시간의 8.5%로 크게 줄어들었다. MCU 디코딩 시간은 52.2%로 절반 이상을 차지하나, DPU 실행 시간과 합산하여 비교했을 때 전체의 약 60%로 기존 디코더에서 DPU 실행 시간이 차지하는 비율보다 작다. 여러 장의 이미지를 동시에 처리할 때는 MCU 디코딩 시간이 선형적으로 증가하여 병목 지점이 될 수 있으나, 이는 멀티스레딩을 통해 나머지 단계와 병렬적으로 실행하여 해결할 수 있다.

반대로, 데이터 이동 (CPU-to-DPU 전송 + DPU-to-CPU 전송) 시간이 전체의 27%를 차지하지만, 실제 소요 시간은 기존 디코더보다 더 길었다. 그 원인은 (1) 구현에 사용된 언어의 차이와 (2) 전송되

표 1. 디코딩 시간 분석 (단위: 밀리초)
Table 1. Execution time breakdowns (unit: milliseconds)

	PIM-JPEG	Proposed
MCU Decoding (Host CPU)	-	291.98
CPU-to-DPU Data Transfer	5.25	69.64
DPU Execution	3073.44	47.86
DPU-to-CPU Data Transfer	1.07	82.80
BMP Write	20.24	67.72
Total	3100	560

표 2. 디코딩 시간 비교 (단위: 초)
Table 2. Execution time comparison (unit: seconds)

	PIM-JPEG	Proposed	Image-Magick
Execution Time	3.10	0.56	0.44

는 데이터 용량의 차이로 보인다. 기존 디코더는 압축된 비트스트림을 1개의 DPU로 전송하지만, 본 논문에서 구현한 디코더는 압축이 해제된 MCU를 여러 개의 DPU로 나누어 전송한다. 허프만 디코딩이 완료되면 픽셀값을 표현하는 비트 수가 늘어나므로, 전송되는 데이터양에 상당한 차이가 발생한다. 대량의 이미지 처리 시에는 데이터 전송 횟수 또한 증가할 것이므로, 데이터 이동이 병목 지점이 될 가능성이 크다.

본 연구에서 제안한 디코더의 실행 시간을 기존의 CPU 기반 디코더와 비교하기 위해, 동일 서버에서 이미지 편집 소프트웨어인 ImageMagick을 사용하여 이미지를 디코딩하고, 그 실행 시간을 측정했다. 표 2는 세 디코더의 실행 시간을 정리한 것으로, 제안한 디코더의 실행 시간은 PIM-JPEG 대비 약 82%가 감소하여 PIM 기반 JPEG 디코딩을 효율적으로 수행하였으나, ImageMagick의 실행 시간 대비 27% 증가한 결과를 보였다. 본 논문에서 제안한 디코더는 디코딩 시간 단축을 목적으로 구현되었기에 BMP 작성 함수의 개선 여지가 남아 있으며, DPU의 클록 주파수가 CPU의 10배 이상 낮다는 점을 고려하면, 디코딩의 효율화를 충분히 이루었다고 볼 수 있다.

V. 결 론

본 연구는 PIM-JPEG에서 처리 가능한 이미지 해상도 제약이 발생하는 원인에 대해 설명하고, 허프만 디코딩이 실행되는 위치를 DPU에서 CPU로 변경하여 해상도 제약을 크게 완화할 수 있음을 보였다. UPMEM PIM 서버에서 고해상도 JPEG 데이터 세트를 사용하여 성능 평가를 수행하였고, 처리 가능한 해상도가 3배 이상 증가했음을 확인했다. 뿐만 아니라, 실행 시간을 측정한 결과, PIM-JPEG 대비 최대 90%의 성능 향상이 확인되었으며, 이미지의 해상도가 높을수록 더 좋은 성능을 보임을 확인했다. 실행 시간 분석 결과, 본 논문에서 제안한 디코더는 데이터 이동 시간이 길게 측정되었으며, 허프만 디코딩을 전송 전에 수행하여 이동하는 데이터의 용량이 커진 것이 원인으로 보인다. 향후 연구에서는 대량의 JPEG 이미지 디코딩을 지원하기 위해, 데이터 전송 오버헤드를 개선할 수 있는 효율적인 구조와 오프로딩 방식을 제안할 예정이다.

References

[1] Joint Picture Experts Group, Overview of JPEG 1. (https://jpeg.org/jpeg/index.html)

[2] J. Nider, J. Dagger, N. Gharavi, D. Ng, and A. Fedorova, "Bulk JPEG decoding on in-memory processors," in *Proc. 15th ACM Int. Conf. Syst. and Storage (SYSTOR '22)*, pp. 51-57, Jun3 2022. (https://doi.org/10.1145/3534056.3534946)

[3] W. Sodsong, M. Jung, J. Park, and B. Burgstaller, "JParEnt: Parallel entropy decoding for JPEG decompression on heterogeneous multicore architectures," in *Proc. 7th Int. Work. Programming Models and Applications for Multicores and Manycores (PMAM'16)*, pp. 104-113, Mar. 2016. (https://doi.org/10.1145/2883404.2883423)

[4] L. Wang, Q. Luo, and S. Yan, "Accelerating deep learning tasks with optimized GPU-assisted image decoding," in *Proc. 2020 IEEE 26th Int. Conf. Parallel and Distributed Systems (ICPADS)*, pp. 274-281, Dec. 2020. (https://doi.org/10.1109/ICPADS51040.2020.00045)

[5] Y. Cheng et al., "DLBooster: Boosting end-to-end deep learning workflows with offloading data preprocessing pipelines," in *Proc. 48th Int. Conf. Parallel Processing (ICPP'19)*, Article No. 88, pp. 1-11, Aug.

2019.
(https://doi.org/10.1145/3337821.3337892)

[6] K. Yan, J. Shan, and E. Yang, “CUDA-based acceleration of the JPEG decoder,” in *Proc. 2013 Ninth Int. Conf. Natural Computation (ICNC)*, pp. 1319-1323, Jul3 2013. (https://doi.org/10.1109/ICNC.2013.6818183)

[7] A. Weißenberger and B. Schmidt, “Accelerating JPEG decompression on GPUs,” in *Proc. 2021 IEEE 28th Int. Conf. High Performance Comput., Data, and Analytics (HiPC)*, pp. 121-130, Dec. 2021. (https://doi.org/10.1109/HiPC53243.2021.00026)

[8] UPMEM DPU SDK. (https://sdk.upmem.com/2024.1.0/index.html)

[9] Y. Arai, T. Agui, and M. Nakajima, “A fast DCT-SQ scheme for images,” *IEICE TRANSACTIONS* (1976-1990), vol. E71, no. 11, pp. 1095-1097, 1988.

[10] H. Choi and H. Kim, “Design of compiler for DRAM PIM and implementation in UPMEM environment,” in *Proc. 2022 Autumn Annual Conf. IEIE*, pp. 1034-1036, Gwangju-si, Korea, Nov. 2022.

[11] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, “Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system,” *IEEE Access*, vol. 10, pp. 52565-52608, May 2022. (https://doi.org/10.1109/ACCESS.2022.3174101)

[12] S.T. Klein and Y. Wiseman, “Parallel huffman decoding with applications to JPEG files,” *The Computer J.*, vol. 46, no. 5, pp. 487-497, Jan. 2003. (https://doi.org/10.1093/comjnl/46.5.487)

[13] Clothing dataset (full, high resolution) (https://www.kaggle.com/datasets/agrigorev/clothing-dataset-full)

김 지 은 (Jieun Kim)

2023년 2월 : 경북대학교 컴퓨터학부 졸업
2023년 3월~현재 : 경북대학교 컴퓨터학부 석사과정
<관심분야> Processing-in-Memory, 미디어 부호화·복호화
[ORCID:0009-0001-9282-3519]

남 덕 윤 (Dukyun Nam)

1999년 2월 : 포항공과대학교 컴퓨터공학과 졸업
2001년 2월 : KAIST 공학 석사
2006년 8월 : KAIST 공학 박사
2004년 7월~2022년 2월 : 한국과학기술정보연구원 슈퍼컴퓨팅본부 (선임연구원)
2022년 3월~현재 : 경북대학교 컴퓨터학부 조교수
<관심분야> 분산시스템, 고성능컴퓨팅
[ORCID:0000-0003-1023-7311]