논문 2023-18-21

엣지 디바이스에서의 병렬 프로그래밍 모델 성능 비교 연구

(A Performance Comparison of Parallel Programming Models on Edge Devices)

남 덕 윤*
(Dukyun Nam)

Abstract : Heterogeneous computing is a technology that utilizes different types of processors to perform parallel processing. It maximizes task processing and energy efficiency by leveraging various computing resources such as CPUs, GPUs, and FPGAs. On the other hand, edge computing has developed with IoT and 5G technologies. It is a distributed computing that utilizes computing resources close to clients, thereby offloading the central server. It has evolved to intelligent edge computing combined with artificial intelligence. Intelligent edge computing enables total data processing, such as context awareness, prediction, control, and simple processing for the data collected on the edge. If heterogeneous computing can be successfully applied in the edge, it is expected to maximize job processing efficiency while minimizing dependence on the central server. In this paper, experiments were conducted to verify the feasibility of various parallel programming models on high-end and low-end edge devices by using benchmark applications. We analyzed the performance of five parallel programming models on the Raspberry Pi 4 and Jetson Orin Nano as low-end and high-end devices, respectively. In the experiment, OpenACC showed the best performance on the low-end edge device and OpenSYCL on the high-end device due to the stability and optimization of system libraries.

Keywords : Edge computing, Heterogeneous computing, Edge devices, Parallel programming model

Ⅰ. 서 론

이기종 컴퓨팅 (Heterogeneous computing)은 서로 다른 아키텍처나 특성을 가진 여러 유형의 프로세서 또는 코어를 활용하여 병렬 처리를 수행하는 기술이다 [1]. CPU, GPU, FPGA 등 다양한 종류의 컴퓨팅 자원을 조합함으로써 응용 프로그램의 성능과 에너지 효율성을 극대화할 수 있다는 장점이 있다. 그러나 이기종 컴퓨팅 기술을 실제 활용하려면 여러 가지 현실적인 제약에 직면한다. 컴퓨팅 자원 간 하드웨어 호환성, 시스템 아키텍처 특성에 따른 성능 차이, 이로 인한 효율적 작업 분배 및 최적화의 어려움, 시스템 아키텍처에 따라 상이한 프로그래밍 모델 활용 등이 그것이다. 최근 관련된 하드웨어 및 소프트웨어 기술이 급속히 발전하면서 이기종 컴퓨팅 기술에 대한 제약사항이 점차 해소되고 있다. FPGA와 같은 유연한 하드웨어 장치를 사용한 다양한 아키텍처 지원 및 하드웨어의 통합, 가상화 기술 기반의 이기종 플랫폼 통합, OpenCL (Open Computing Language) [2]과 같은 개방형 표준 프로그래밍 모델 및 프로토콜의 개발로 인해, 이기종 컴퓨팅 기술 및 활용에 관한 연구가 상당한 진전을 보이고 있다.

*Corresponding Author (dynam@knu.ac.kr)
Received: Jun. 18, 2023, Revised: Aug. 7, 2023, Accepted: Aug. 17, 2023.
D. Nam: Kyungpook National University (Assis. Prof.)
※ 본 논문은 정부 (과학기술정보통신부)의 재원으로 한국연구재단의 지원 (No. RS-2023-00211606)과 정보통신기획평가원의 지역지능화혁신인재양성사업의 지원 (No. IITP-2023-RS-2022-00156389)으로 연구하였음.

ⓒ IEMEK J. Embed. Sys. Appl. 2023 Aug. 18(4) 165-172
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2023.18.4.165

한편, IoT (Internet of Things) 및 5G 기술의 발달과 함께 엣지 컴퓨팅 (Edge computing)에 대한 연구가 활발히 진행 중이다 [3]. 엣지 컴퓨팅은 중앙 서버의 컴퓨팅 성능을 분산시켜 클라이언트에 인접한 컴퓨팅 자원을 활용하는 분산 컴퓨팅 기술이다. 최근에는 인공지능 기술과 결합하여 지능형 엣지 컴퓨팅 (Intelligent Edge Computing)으로 진화하고 있다. 지능형 엣지 컴퓨팅이란 데이터가 생성되는 현장에서 직접 데이터에 대한 종합적인 분석과 처리를 수행하는 접근 방식이다. 데이터에 대한 단순 처리작업뿐 아니라 상황 인식, 추론, 예측 및 제어 등의 작업이 엣지에서 수행됨으로써 중앙 서버로부터의 대기 시간을 줄이고 작업효율을 극대화할 수 있다. 이를 위해 엣지 서버 혹은 엣지 디바이스는 대량으로 생성되는 데이터를 효율적으로 처리할 수 있어야 한다. 엣지 환경에서 이기종 컴퓨팅 기술을 활용할 수 있다면 중앙 서버에 대한 의존성을 최소화하면서 엣지 컴퓨팅 환경의 성능과 효율성을 크게 향상시킬 수 있다. 특히 엣지 디바이스에서 이기종 컴퓨팅 자원을 활용할 수 있다면 작업효율을 극대화할 수 있을 것으로 기대된다. 그림 1은 엣지 서버와 엣지 디바이스로 구성된 엣지 컴퓨팅 환경의 예로, 엣지 서버는 CPU, 내장 GPU (Integrated GPU), 외장 GPU (Discreate GPU) 등 3종의 계산 유닛을, 엣지 디바이스는 CPU와 GPU를 포함하고 있다.

한편 고성능 컴퓨팅 환경에서는 복잡한 계산작업과 대규모 데이터 처리를 효율적으로 수행하기 위해 CPU 기반 컴퓨팅 환경에서는 OpenMP, MPI와 같은 병렬 프로그래밍 모

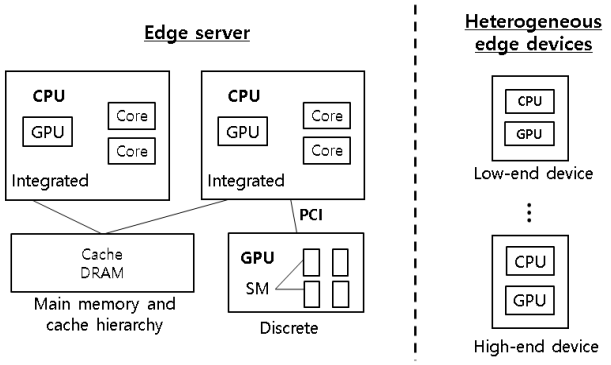


그림 1. 이기종 엣지 서버와 디바이스 예시
Fig. 1. Example of heterogeneous edge server and devices

델 [4]을, 엔비디아 GPU 기반 컴퓨팅 환경에서는 CUDA (Compute Unified Device Architecture) 프로그래밍 모델 [5]을 활용해 왔다. 최근 GPU 외에도 인공 신경망 학습과 추론작업을 위한 TPU (Tensor Processing Unit), NPU (Neural Processing Unit) 등 다양한 프로세서들이 개발되면서 이기종 컴퓨팅 자원을 활용한 병렬 처리에 관한 관심이 증가하고 있으며 기존의 병렬 프로그래밍 모델들이 이기종 컴퓨팅 지원을 위해 확장되거나 고수준 언어를 이용해 이기종 컴퓨팅을 지원하는 SYCL이 제안되었다 [6].

엣지 컴퓨팅 환경에서 이기종 컴퓨팅 기술을 실제로 활용하려면 고성능 컴퓨팅 환경을 대상으로 개발되었던 다양한 병렬 프로그래밍 모델들의 활용 가능성 및 성능에 대한 검토가 필요하다. 하드웨어 및 소프트웨어의 발전으로 최근 엣지 디바이스의 성능이 비약적으로 발전했다 하더라도 고성능 시스템과 비교할 때 메모리 용량 및 저장공간 등의 컴퓨팅 자원 및 전력 소비에 대한 태생적인 제약사항이 존재하기 때문이다. 따라서 본 논문에서는 고성능 컴퓨팅 환경에서 제안되었던 병렬 프로그래밍 모델들을 대상으로 엣지 디바이스에서의 활용 가능성에 대해 살펴보고, 엣지 디바이스에서 각 모델을 활용하여 성능 측정 및 분석을 수행하였다. 실험을 위해 가격과 성능을 고려해 엣지 디바이스를 저사양과 고사양으로 구분하고, 벤치마크 프로그램 [7]를 사용하여 성능 측정을 진행했다. 저사양 및 고사양 디바이스로서 각각 라즈베리 파이4 [8]와 젯슨 오린 나노 [9]를 대상으로, OpenSYCL [10], OpenCL [2], OpenMP [11], OpenACC [12], CUDA [5]의 총 5종을 활용해 병렬 프로그래밍 모델에 대한 실험 결과를 제시했다. 엣지 디바이스에서 병렬 프로그래밍 모델로 CPU와 GPU를 모두 활용할 수 있는 경우는 현재 고사양 디바이스에서의 OpenSYCL 조합만이 유일하게 가능했다. 성능 측면에서는 저사양 엣지 디바이스에서 OpenACC가, 고사양 디바이스에서는 OpenSYCL이 가장 우수한 성능을 보였다. 이는 저사양 디바이스의 경우 시스템 라이브러리의 안정성 차이로 인한 결과이며, 고사양 다비이스의 경우에서도 자원에 최적화된 런타임 라이브러리를 활용한 결과로 판단된다. 한편, 고수준 프로그래밍 언어로 제안된 SYCL이 엣지 디바이스에서도 우수한 성능을 보일 가능성을 확인할 수 있었으며, 실제로 SYCL 기반 응용 프로그램에서 CPU, GPU 등 다양한 컴퓨팅 자원들을 효율적으로 쉽게 활용할 수 있다면 [13], 다양한 분야에서 더 많이 활용될 수 있을 것이다.

본 논문의 구성은 다음과 같다. 2장에서는 이기종 컴퓨팅 환경을 지원하기 위해 제시되었던 프로그래밍 모델을 살펴보고, 3장에서 저사양 디바이스인 라즈베리 파이와 고사양 디바이스인 젯슨 오린 나노에 대해, 실험을 위한 하드웨어 및 소프트웨어 구성을 기술한다. 4장에서는 벤치마크 프로그램을 이용하여 프로그래밍 모델별 실행 성능 결과를 분석하고, 마지막으로 5장에서 결론을 맺는다.

Ⅱ. 이기종 컴퓨팅을 위한 병렬 프로그래밍 모델

본 절에서는 이기종 컴퓨팅을 위해 제시되었던 프로그래밍 접근법에 관해 기술한다. 이기종 컴퓨팅 연구 초기에는 CPU 이외의 GPU를 효율적으로 활용하기 위한 연구가 수행되었으나 [14], TPU, FPGA 등 다양한 이기종 프로세서가 등장함에 따라 GPU 외에도 다양한 이기종 프로세스를 수용하기 위한 연구로 확장되고 있다. 일반적으로 이기종 컴퓨팅을 위한 병렬 프로그래밍은 지시어 기반 프로그래밍, GPU용 저수준 프로그래밍, 고수준 프로그래밍의 세 가지 범주로 구분된다.

1. 지시어 기반 프로그래밍 언어

지시어 기반 프로그래밍 언어는 기존의 프로그래밍 언어에 확장된 지시문을 추가하여 병렬 처리를 명시적으로 지정함으로써 컴파일러에 해당 부분을 자동으로 병렬화하도록 안내하는 방식이다. 기존 코드의 수정을 최소화면서 상대적으로 쉽게 병렬 프로그래밍을 지원할 수 있다는 장점이 있다.

1.1 OpenMP

지시문 기반 프로그래밍의 대표적인 예로 1990년대 개발된 OpenMP가 있다. 당시 분산 시스템에서 여러 개의 노드가 상호작용하며 프로세스 통신을 기반으로 병렬처리를 수행하기 위한 MPI (Message Passing Interface)가 활용되었다. 그러나 단일 노드 내에서의 병렬 처리를 위해 보다 쉽게 사용할 수 있는 방법이 필요해졌고, 이에 대한 요구를 충족시키기 위해 1990년대 중반에 OpenMP가 개발되었다. OpenMP는 공유 메모리 시스템에서의 병렬 처리를 위한 프로그래밍 모델로서, 기존의 C, C++, Fortran과 같은 프로그래밍 언어에 pragma 지시문을 추가하여 병렬화를 지정한다. 하지만, 스레드 경쟁 조건이 존재하는 메모리 모델이기 때문에 최적화가 어려울 수 있다. OpenMP는 분산 메모리 아키텍처를 사용하는 여러 노드가 아닌 단일 노드 내 병렬처리를 지원하므로, 메모리 확장성은 노드의 메모리로 제한된다. 따라서, 더 큰 메모리 요구 사항이 있는 병렬 프로그램은 OpenMP를 MPI와 같은 분산 메모리 병렬 기술과 함께 사용해야 한다. 한편, OpenMP 초기 버전에서는 CPU 기반 시스템에서만 작동했으나, 이후 GPU 지원을 위한 OpenMP 디렉티브 및 라이브러리 함수가 추가되었다. GPU 지원을

위해서는 OpenMP 4.0 이상 버전이 필요하다.

1.2 OpenACC

OpenMP 외의 다른 지시어 기반 프로그래밍 모델로 OpenACC가 있다. 2011년 GPU에 대한 접근성에 대해 논의하는 과정에서 OpenMP가 CPU 병렬처리 지원에 집중하고 있는 관계로, NVIDIA와 당시 소규모의 컴파일러 공급업체 그룹들 (Cray, PGI 등)은 GPU 병렬처리에 대한 보다 편리한 방법을 제공하고자 OpenACC 표준을 제정하였다.

OpenACC는 병렬화 및 GPU 가속화를 위한 지시문을 제공하며, OpenMP처럼 C, C++, Fortran 등 기존의 프로그래밍 언어를 확장하여 사용한다. OpenMP는 단일 노드 내 CPU 병렬처리에 중점을 두는데 반해, OpenACC는 CPU와 GPU 간 효율적 상호작용을 위해 다양한 지시문을 지원한다. 예를 들면, 데이터를 GPU 메모리로 전송하거나 CPU와 GPU 메모리 간의 데이터 이동을 지정하는 데이터 지시문, 병렬로 실행될 코드 영역을 지정하는 병렬 지시문, 반복문을 병렬로 실행할 수 있도록 지정하는 루프 지시문 등을 다양한 지시문을 지원한다. OpenMP와 OpenACC는 선의의 경쟁 관계로, 현재 GPU 및 가속기를 지원하기 위한 기능을 확대하고 있다 [15].

2. GPU용 저수준 프로그래밍

GPU는 원래 효율적인 그래픽 처리를 위해 개발되었으나 점차 고성능 병렬 컴퓨팅을 위한 용도로도 활용되기 시작하였다. 이를 위해 GPU에 대한 저수준 프로그래밍이 요구되었는데 2000년대 초반에는 GPU 제조업체들이 독자적인 프로그래밍 모델과 API를 제공하였다. 2006년 NVIDIA가 CUDA를 발표하면서 GPU용 저수준 프로그래밍 모델이 등장하였으며 CUDA는 개발자들이 GPU를 활용한 고성능 병렬 처리를 수행할 수 있는 기술을 제공하였다. GPU용 저수준 프로그래밍은 GPU를 활용해 병렬 처리를 지원하기 위한 저수준 (low-level) API를 제공하며, 개발자가 직접 병렬 실행 단위와 데이터 전송 등을 제어하여 고성능 병렬 처리를 구현할 수 있도록 지원한다. 지시어 기반 프로그래밍 언어는 고수준에서 추상화하여 개발자가 프로그램의 병렬성을 표현하고 의도를 명시할 수 있는 동시에, 컴파일러는 하드웨어에 맞게 변환하여 최적화된 실행을 구현한다. 반면, 저수준 언어는 컴파일러에 대한 의존도가 낮고 개발자에게 더 많은 제어권을 주어 하드웨어의 세부 사항을 직접 콘트롤함으로써 성능을 극대화시킬 수 있다. GPU용 저수준 프로그래밍 언어로 널리 사용되고 있는 언어로는 CUDA 외에도 OpenCL이 있다.

2.1 NVIDIA GPU를 위한 CUDA 모델

CUDA는 NVIDIA GPU를 위한 프로그래밍 모델로서, 그래픽스 작업은 물론 고성능 컴퓨팅을 위한 병렬처리를 GPU에서 실행할 수 있도록 지원한다. CUDA를 사용해 CPU 작업의 일부를 GPU로 이동하여 병렬로 처리하여 계산 집약적인 작업을 효율적으로 수행하고 GPU의 다수의 코어를 활용하여 성능을 향상시킬 수 있다. CUDA 모델은 개발자에게 GPU 자원을 직접 제어할 수 있는 기능을 제공함으로써 성능을 최대한 활용할 수 있도록 지원한다.

개발자는 병렬로 실행될 코드 영역을 커널로 지정한다. 커널은 GPU에서 병렬로 실행되는 함수이다. CUDA 스레드 계층구조는 작업의 최소단위인 스레드, 하나 이상의 스레드로 구성된 스레드 블록, 최상위 개체인 그리드로 구성된다. 스레드 블록은 커널 내에서 동시에 실행되는 작업 단위이다. 커널은 스레드 블록들을 조정하고 스레드 블록들은 스레드 간의 협력을 통해 작업을 수행하며, 이를 통해 CUDA에서 병렬 처리를 구현한다. 그리드는 여러 스레드 블록들로 구성되며 독립적으로 실행될 수 있으며, 전체 CUDA 작업의 조직화와 병렬 실행을 담당한다.

2.2 OpenCL

CUDA는 NVIDIA에 의해 개발되어 NVIDIA GPU에 특화된 반면, OpenCL은 크로노스 그룹 (Khronos Group)에 의해 관리되고 있는 개방형 프로그래밍 표준으로서 벤더 중립적인 프로그래밍 모델로 다양한 벤더의 GPU 및 가속화 장치 지원을 표방한다. OpenCL은 C99 또는 C++로 디바이스 프로그래밍을 지원하는 라이브러리로, OpenCL 응용 프로그램은 호스트 프로그램 부분과 하나 이상의 커널부분으로 구성된다. 커널 부분은 연산 코어에서 병렬 방식으로 실행될 작업을 지정하며, 호스트 프로그램 부분은 실행 환경을 설정하고 데이터를 읽고 실행할 커널을 객체화하여 대기열에 추가하는 역할을 한다. 또한 OpenCL은 추상 실행 모델과 플랫폼 모델을 구분하여 정의하고, 표준을 따르는 런타임 라이브러리 구현을 제공함으로써 특정 하드웨어에 종속되지 않도록 이식성을 보장한다. OpenCL의 코드 이식성은 크로노스의 인증 프로그램에 의해 보장된다.

3. 고수준 프로그래밍

고수준 프로그래밍 언어는 이기종 컴퓨팅 자원을 고수준 (high level) 언어로 지원하기 위한 모델로 개발자가 복잡한 하드웨어 세부 사항을 알 필요 없이 병렬 처리를 수행할 수 있도록 추상화된 인터페이스를 제공함으로써 개발의 편의성을 향상시킨다. 컴파일러와 런타임 시스템에 더 많은 작업을 수행하게 하고 개발자의 역할을 재정의함으로써, 궁극적으로 프로그래밍 작업 시 저수준 API 및 아키텍처별 최적화로부터 자유롭게 하는 데에 목적이 있다.

OpenCL의 많은 장점에도 불구하고 저수준 언어의 복잡성과 장황한 추가기능으로 인해 활용이 어려웠다. 이에 프로그래밍 진입 장벽을 낮추고 추상화 수준 개선에 대한 필요성이 대두되었는데, 이에 제안된 표준이 SYCL이다. 2015년 Khronos Group에서 시작한 SYCL은 C++ 기반의 고수준 프로그래밍 모델로서, GPU뿐만 아니라 FPGA와 같은 이기종 컴퓨팅 자원을 지원하는데 초점을 두고 있다.

SYCL은 OpenCL의 C 언어 기반 프로그래밍 모델 위에 구축된 실험적인 C++ 구현이었다. OpenCL은 저수준의 GPU 프로그래밍을 위해 C 언어를 사용하는 반면, SYCL은 C++ 언어를 확장하여 추상적인 인터페이스를 통해 GPU 프

로그래밍을 지원한다. 또한 개발자들이 익숙한 C++ 문법과 기능을 활용하여 GPU 컴퓨팅을 수행할 수 있도록 지원한다. 또한, OpenCL의 플랫폼 모델과 추상 실행 모델을 유지하면서도 C++ 언어의 장점을 최대한 활용함으로써, 개발자는 높은 수준의 추상화와 유연성을 통해 GPU 가속화 작업을 보다 편리하게 수행할 수 있다. 현재 활용할 수 있는 대표적인 SYCL 구현으로는 Codeplay사의 computeCPP [16], Intel사의 DPC++ [17], 하이델베르그 대학의 OpenSYCL [18] 등이 있다.

Ⅲ. 성능 측정을 위한 실험 환경

본 논문에서는 고성능 컴퓨팅 환경에서 제안되었던 병렬 프로그래밍 모델들을 대상으로 엣지 디바이스에서의 활용 가능성에 대해 살펴보고, 엣지 디바이스에서 각 모델을 활용하여 성능 측정 및 분석을 수행하였다. 본 절에서는 실험에서 활용된 엣지 디바이스와 병렬 프로그래밍 모델, 벤치마크 응용에 대해 기술한다.

1. 실험 대상 엣지 디바이스 및 하드웨어 사양

엣지 디바이스는 성능과 가격에 따라 다양한 형태로 구성될 수 있다. 본 실험에서는 성능과 가격을 고려해 최소사양과 최고 사양 디바이스를 각 1종씩 선정해 실험을 수행함으로써 보편적인 엣지 디바이스에서의 이기종 컴퓨팅 자원활용 가능성을 제시하고자 하였다. 저사양 기기라 하더라도 최소한 병렬 프로그래밍 모델 수행이 가능해야 하므로, 우분투 및 기존 GNU 계열 소프트웨어의 설치가 가능한 사양을 선택하였다. 또한 보편적 활용을 위해 엣지 디바이스의 가격이 현실적이어야 한다는 점을 고려해 100만원 이하에서 고사양 디바이스를 선정했다. 이에 그림 2에서와 같이 최소사양 디바이스는 라즈베리 파이 4 모델 B를, 최고 사양 디바이스는 개방형 병렬 프로그래밍 모델은 아니지만 현재 병렬 컴퓨팅의 주요 활용 모델인 CUDA 프로그램 구동이 가능한 엔비디아 젯슨 오린 나노 모델을 선정하였다.

표 1은 실험 디바이스들의 주요 사양으로, 라즈베리 파이의 경우 CPU는 ARM Quad-core A72, GPU는 Broadcom VideoCore VI를 탑재하고 있으며, 젯슨 나노 오린의 경우 CPU는 ARM 6-core Cortex-A78AE, GPU는 1204-core

그림 2. 실험용 디바이스: 라즈베리 파이4와 젯슨 오린 나노
Fig. 2. Experimental devices: Raspberry Pi 4 and Jetson Orin Nano

표 1. 실험 디바이스 주요 사양
Table 1. Main specifications of experimental devices

	Low-end device	High-end device
Machine	Raspberry Pi 4 Model B	Jetson Orin Nano Dev Kit
CPU model	4-core Cortex-A72 64-bit	6-core Cortex-A78AE 64-bit
CPU speed	1.5 GHz	1.5 GHz
GPU	VideoCore VI, 250 MHz	1204-core Ampere, 625 MHz
AI accel.	-	32 Tensor cores
FLOPS(FP64)	13.5 GFLOPS (CPU)	640 GFLOPS (GPU)
Memory	LPDDR4 8GB	LPDDR5 8GB
Networking	Gigabit Ethernet 1 Gbps	Gigabit Ethernet 1 Gbps
Storage	32G microSD	32G microSD
Power Input	5V, 15W	19V, 7W to 15W
Size (mm)	100 x 76 x 30	103 x 90.5 x 35
Release date	June 2019	March 2023
Price	$35	$499

Ampere가 장착되어 있다. 두 디바이스의 이론적인 계산 성능은 라즈베리 파이는 13.5 GFlops, 젯슨 오린 나노는 640 GFlops이다. 이후 실험 결과에서 설명하겠지만, 라즈베리 파이는 CPU만을 계산 자원으로 활용할 수 있었던 반면, 비교적 최근에 출시된 젯슨 나노 오린은 NVIDIA GPU를 활용할 수 있었다.

2. 실험 대상 병렬 프로그래밍 모델 및 소프트웨어 환경

예전과 달리 엣지 디바이스에서도 다양한 리눅스 배포판이 정식으로 지원됨에 따라, 기본적으로 많이 사용되는 소프트웨어 패키지들은 APT (Advanced Packing Tool)와 같은 소프트웨어 관리 도구를 이용할 수 있게 되었다. 라즈베리 파이의 경우 전용의 라즈베리 파이 OS가 존재하지만 우분투를 설치하여 실험 환경을 동일하게 설정하였다.

본 실험에서는 지시어 기반 프로그래밍, GPU용 저수준 프로그래밍, 고수준 프로그래밍의 세 가지 범주에 속하는 다양한 병렬 프로그래밍 모델을 대상으로 CPU와 GPU 기반 이기종 컴퓨팅 기술의 적용 가능성을 검증하고 성능 측정을 수행하였다. 지시어 기반 프로그래밍 모델은 OpenMP와 OpenACC를, GPU용 저수준 프로그래밍 모델은 CUDA와 OpenCL을, 고수준 프로그래밍 모델은 OpenSYCL을 대상으로 하였다.

OpenSYCL을 제외한 대부분의 병렬 프로그래밍 모델은 GNU 계열의 GCC와 같은 기본적인 패키지에 포함되어 있으나, SYCL 관련 구현은 아직 일반적인 배포판에 포함된 버전이 없어 3rd party 제조사들이 자체적으로 개발한 다양한 소프트웨어를 검토해야 했다. 현재 리눅스에서 활용할 수 있는 SYCL 구현은 Codeplay사의 ComputeCpp, 하이델베르그 대학의 OpenSYCL, Intel사의 DPC++, triSYCL 등이다. ComputeCPP는 ARM 기반 바이너리만 제공하고 DPC++는 x86 기반 Intel CPU/GPU를, triSYCL은 Xilinx FPGA 지원을 우선으로 하고 있어, 본 실험에서는 OpenSYCL을 활용하였다. 한편 ARM 계열 GPU인 Mali에 관해 집약적 계산으로 활용하기 위한 시도들이 진행 중이며,

표 2. 실험 디바이스별 소프트웨어 목록
Table 2. List of software on experimental devices

	Low-end device	High-end device
OS	Ubuntu 20.04.6 LTS	Ubuntu 20.04.6 LTS
Kernel	5.4.0-1069-raspi aarch64	5.10.104-tegra aarch64
gcc/g++	9.4.0	9.4.0
cmake	3.16.3	3.16.3
clang/llvm	12	12
OpenACC	Not specified	Not specified
OpenCL	1.4	1.4
OpenMP	201511	201511
OpenSYCL	0.9.4	0.9.4
nvcc	Not installed	cuda_11.4.r11.4

현재 Codeplay사는 ARM Mali GPU에 대해 SYCL을 공식적으로 지원한다. 실험용 엣지 디바이스들에 설치된 소프트웨어는 NVIDIA 컴파일러인 nvcc 유무 외에는 모두 동일하게 구성하였으며, 상세 정보는 표 2와 같다.

3. 활용 벤치마크 프로그램

계산 집약적 특성에 기반한 과학응용 프로그램들의 컴퓨팅 성능은 컴퓨팅 자원 자체의 성능뿐 아니라 메모리 대역폭에 의해 성능이 제한된다. 병렬 프로그래밍 모델의 성능 또한 메모리 대역폭에 크게 의존하며, 메모리 대역폭이 낮으면 데이터를 효율적으로 전송하지 못해 고성능 시스템의 연산 처리 능력을 효율적으로 활용할 수 없다. 이에 본 논문에서는 메모리 대역폭 측정을 통해, 각 병렬 프로그래밍 모델에 대한 병렬화 기법 및 메모리 액세스 패턴에 따른 성능 차이를 살펴보고자 하였다. STREAM [19]은 주로 고성능 컴퓨팅 시스템에서 메모리 대역폭을 측정하고 메모리 지연시간을 평가하기 위한 벤치마크 프로그램이다. BabelStream [7]은 Stream을 기반으로 확장된 벤치마크로서, 보다 다양한 메모리 액세스 패턴과 작업 부하를 포함하며 명시적으로 CPU와 GPU를 모두 지원하도록 설계된 벤치마크이다. 최근에는 SYCL, OpenCL, OpenMP, OpenACC 등 다양한 병렬 프로그래밍 모델을 통합하여 지원한다. 이에 다양한 개방형 병렬 프로그래밍 모델을 비교하고자, 본 논문에서는 BabelStream을 이용해 엣지 디바이스에서의 병렬 프로그래밍 모델 성능을 측정하였다.

커널은 GPU에서 병렬적으로 수행하기 위한 연산들로 구성된 함수로, BabelStream은 아래 제시된 바와 같이 복사 (Copy), 곱셈 (Multiply), 덧셈 (Add), 트라이어드 (Triad), 내적(Dot product)의 5개의 커널함수를 이용해 성능을 측정한다. (여기서, a, b, c는 배열이며, α는 스칼라 상수이다.)

1) Copy: c[i] = a[i]
2) Multiply: b[i] = αc[i]
3) Add: c[i] = a[i] + b[i]
4) Triad: a[i] = b[i] + αc[i]
5) Dot: sum = sum + a[i] * b[i]

벤치마크 프로그램을 통해 엣지 디바이스에서 병렬 프로그래밍 모델에 대한 현실적인 성능 기대치를 추정해 볼 수 있다. 벤치마크 실험 결과 성능이 만족스럽지 않다면, 현실적으로 엣지 디바이스에서 이기종 컴퓨팅 자원을 활용해 과학 응용 프로그램은 수행하기 어려우며 해당 분야에 대한 연구가 더욱 절실하다고 할 수 있다.

Ⅳ. 실험 결과 및 분석

저사양 엣지 디바이스인 라즈베리 파이와 고사양 엣지 디바이스인 젯슨 오린 나노를 대상으로 5종의 병렬 프로그래밍 모델을 벤치마크 프로그램을 통해 이기종 컴퓨팅 자원에 대한 활용 가능성을 검증하였다. 이후 각 엣지 디바이스마다 활용 가능한 병렬 프로그래밍 모델 간의 성능을 비교하는데에 있어서, 대역폭 측정 결과를 보인다. 실험의 기본값들은 벤치마크 프로그램의 기본값을 활용했으며, 데이터는 더블 타입, 배열 크기는 33,554,432 (약 0.3GB), 커널의 실행 횟수는 100회로 설정한 후 실험하였다.

1. 저사양 엣지 디바이스에서의 실험

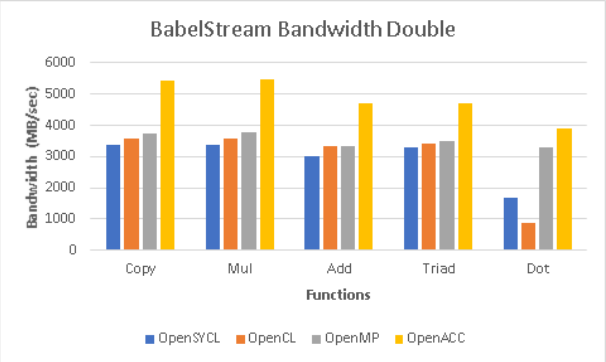
저사양 엣지 디바이스와 고사양 엣지 디바이스에서 벤치마크 프로그램을 이용해 이기종 컴퓨팅 자원에 대한 활용성을 검증한 결과는 표 3에 도시되어 있다. 라즈베리 파이 모델4에 탑재된 Videocore VI GPU는 주로 그래픽스 연산을 위해 제공되는 GPU로, 해당 GPU는 계산에 사용될 만큼 충분한 코어를 갖고 있지 않을 뿐만 아니라, 제조사 또한 그래픽스 외의 계산작업을 위한 공식적인 지원을 하지 않고 있다. 라즈베리 파이 모델3에서 OpenCL을 수행할 수 있는 드라이버는 존재하여 [20], 라즈베리 파이 모델 4에서도 OpenCL과 관련한 기본적인 정보는 읽어올 수 있으나, 벤치마크 프로그램을 이용하여 병렬 계산을 수행할 수는 없었다. 한편 CUDA는 NVIDIA GPU 전용이므로 라즈베리 파이 4에서 활용할 수 없었다.

결국 저사양 디바이스에서는 CPU만을 이용한 병렬 프로그래밍 모델만 실행할 수 있었으며, 성능은 표 4의 대역폭 결과에서 보듯 OpenACC, OpenMP, OpenCL, OpenSYCL 순으로 측정되었다. 특히, OpenACC는 5개의 함수, Copy, Mul, Add, Triad, Dot 함수 모두에 대해 OpenMP보다 18~45% 가량 좋은 성능을 보였다.

표 3. 디바이스 자원별 프로그래밍 모델 실행 여부
Table 3. Availability of programming model by device resource

	Low-end device		High-end device	
	CPU	GPU	CPU	GPU
OpenSYCL	○	×	○	○
CUDA	×	×	×	○
OpenCL	○	△	○	△
OpenMP	○	×	○	×
OpenACC	○	×	○	×

표 4. 저사양 디바이스에서의 대역폭 측정 결과 (MB/sec)
Table 4. Bandwidth results measured on a low-end device



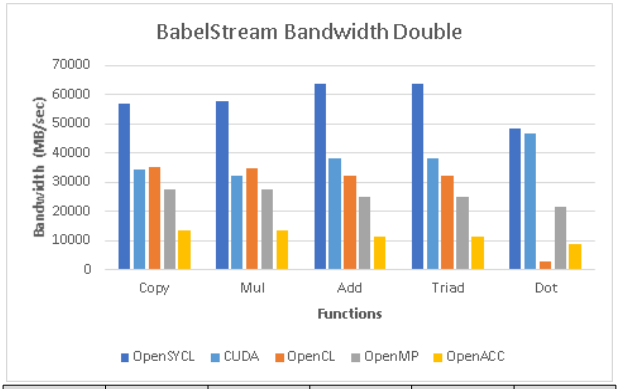
	Copy	Mul	Add	Triad	Dot
OpenSYCL	3383.777	3374.130	3019.343	3310.555	1686.508
OpenCL	3589.839	3588.305	3356.826	3410.185	888.057
OpenMP	3749.969	3776.202	3353.711	3489.555	3314.591
OpenACC	5443.040	5466.124	4699.787	4698.746	3902.857

리눅스에서 시스템 호출을 추적할 때 사용하는 strace [21]로 시스템 호출 통계인 표 5를 보면, OpenMP, OpenCL, OpenSYCL에서는 futex (fast user space mutex)로 인한 대기 시간이 많이 발생하고, OpenACC에서는 futex로 인해 걸리는 시간이 거의 존재하지 않았다.

한편 futex 다음으로 많이 불리는 시스템 호출은 메모리 매핑 관련 인터페이스인 munmap 호출로 모든 모델에서 거의 동일한 시간이 소요되었다. 호출 프로세스의 가상 주소 공간에 새로운 메모리 매핑을 생성한 mmap 호출의 역인 munmap 호출은 호출 프로세스의 가상 주소 공간에서 매핑을 제거하는 데에 많은 시간을 소모하였다.

마지막으로 특이한 결과는 Dot 커널의 경우로, OpenSYCL과 OpenCL의 결과가 다른 모델에 비해 매우 낮았다. 이는 두 모델에는 리덕션 API가 내장되어 있지 않기 때문에, 사용자 코드에서 리덕션 기능이 구현될 필요가 있었고, 이로 인해 다른 병렬 모델에서의 결과보다 낮은 성능을 보인 것으로 판단된다.

표 6. 고사양 디바이스에서의 대역폭 측정 결과 (MB/sec)
Table 6. Bandwidth results measured on a high-end device



	Copy	Mul	Add	Triad	Dot
OpenSYCL	56815.57	57617.92	63622.76	63611.99	48296.08
CUDA	34523.00	32233.27	38338.03	38175.59	46929.70
OpenCL	35244.70	34959.15	32381.42	32353.73	2778.50
OpenMP	27485.45	27461.16	25150.16	25146.92	21806.62
OpenACC	13479.16	13454.44	11465.21	11481.55	8875.74

2. 고사양 엣지 디바이스에서의 실험

젯슨 오린 나노 디바이스는 엔비디아의 제트팩 (JetPack) SDK [22]를 이용하여 소프트웨어 구성을 설치한다. 제트팩은 엔비디아의 종단 간 가속 AI 응용을 지원하기 위한 솔루션으로, 리눅스 커널, 우분투 데스크탑 환경, GPU 컴퓨팅 등 관련 라이브러리를 제공한다. 이에 처음 구동 시에는 빠르고 편하게 설치하여 활용할 수 있는 반면, CUDA 프로그래밍 모듈 외에는 최적화를 위한 노력이 필요한 것으로 보인다.

표 6의 대역폭 측정 결과를 분석해보면, OpenSYCL이 전반적으로 우수한 성능을 나타내는 것으로 확인된다. 이러한

표 5. 저사양 디바이스에서의 병렬 프로그래밍 모델별 주요 시스템 콜 현황
Table 5. Statistics of system calls by parallel programming frameworks on a low-end device

System call	OpenSYCL				OpenCL				OpenMP				OpenACC			
	% time	milli sec.	usec /call	calls	% time	milli sec.	usec /call	calls	% time	milli sec.	usec /call	calls	% time	milli sec.	usec /call	calls
brk	0.0	0.03	5	5	0.0	0.37	14	25	0.0	0.03	11	3	0.0	0.03	11	3
clone	0.0	0.00	0	4	0.0	0.33	82	4	0.1	0.27	89	3	-	-	-	-
close	0.0	0.39	14	27	0.1	2.07	15	133	0.0	0.10	14	7	0.0	0.10	14	7
faccessat	0.0	0.00	0	1	0.0	0.33	41	8	0.0	0.00	0	1	0.0	0.00	0	1
futex	**95.6**	5447.8	5284	1031	**91.2**	2721.8	778	3496	**10.1**	27.6	42	649	0.0	0.03	15	2
getdents64	0.0	0.31	51	6	0.0	0.22	27	8	0.1	0.24	117	2	0.1	0.23	113	2
mmap	0.0	1.02	18	54	0.1	1.38	13	100	1.6	4.47	144	31	0.2	0.53	19	28
mprotect	0.0	0.43	16	26	0.0	0.78	15	50	0.1	0.33	21	15	0.1	0.27	22	12
munmap	4.2	**240.5**	6682	36	7.8	**232.1**	3934	59	86.9	**238.4**	14024	17	98.6	**227.2**	13366	17
newfstatat	0.0	1.19	18	63	0.0	1.08	15	68	0.1	0.39	24	16	0.2	0.36	22	16
openat	0.1	4.22	25	168	0.3	8.71	28	307	0.5	1.24	28	43	0.5	1.18	27	43
read	0.0	0.53	24	22	0.2	5.81	48	119	0.0	0.10	20	5	0.0	0.10	20	5
renameat	-	-	-	-	0.1	1.48	1482	1	-	-	-	-	-	-	-	-
unlinkat	0.0	0.06	64	1	0.1	2.20	1101	2	-	-	-	-	-	-	-	-
write	0.0	0.51	32	16	0.1	2.28	7	324	0.2	0.41	31	13	0.2	0.40	18	22
total	100	5697.5	-	1504	100	2983.2	-	4816	100	274.3	-	820	100	230.5	-	165

성능 향상은 OpenSYCL이 내부적으로 OpenMP와 CUDA 모델의 런타임 라이브러리를 효과적으로 연동하여 작업을 처리하기 때문으로 판단된다. 특히, NVIDIA가 제공하는 최적화된 런타임인 CUDA 런타임 라이브러리(cudart.so)를 활용하는 OpenSYCL이 가장 높은 성능을 보였다.

한편, OpenSYCL 외의 모델들은 주로 젯슨 제조사인 NVIDIA가 제공하는 제트팩 소프트웨어 개발 키트에 포함된 컴파일러 및 라이브러리를 기반으로 실행 파일이 생성되었다. 이러한 모델들은 CUDA 런타임을 활용하는 OpenSYCL과 비교해 볼 때 낮은 성능을 보여주었다. OpenCL의 경우에는 제트팩에 기본적으로 포함되지 않기 때문에, PoCL [23]을 활용하여 디바이스를 인식하였다.

또한, 저사양 디바이스에서 우수한 성능을 보여주었던 OpenACC 모델도 고사양 디바이스에서는 기대한 성능을 발휘하지 못한 것으로 확인되었다. 향후 엔비디아 HPC SDK [24]가 제트팩에 포함되어 지원된다면 OpenACC도 좋은 성능을 보여줄 것으로 기대된다.

실험 결과 고사양 디바이스에서의 병렬 프로그래밍 모델별 주요 시스템 콜 통계인 표 7을 보면, CPU에서 호출되는 함수들만 취득되어 아쉽기는 하지만, 프로그래밍 모델 별로 활용하는 자원에 따라 이와 관련된 결과가 측정된 것으로 생각된다. OpenSYCL은 CPU와 GPU를 모두 활용하는 형태로 스레드 스케줄링과 관련한 함수가 많이 호출된 것으로 보이며, CUDA에서는 I/O 컨트롤 관련, OpenCL과 OpenMP는 futex에서 많은 시간이 소요되었다. 한편 저사양 디바이스 결과에서 많은 시간을 소요했던 munmap 함수의 경우 OpenACC 외에는 모두 개선된 결과를 보였다. 이는 최신의 시스템 소프트웨어 라이브러리 최적화가 반영된 결과로 생각된다.

본 실험 결과에서 CPU 자원만 활용할 수 있었던 저사양 디바이스에서는 OpenACC가 OpenMP에 비해 평균 36.9% 더 우수한 성능을 보였으며, GPU 자원을 활용할 수 있었던 고사양 디바이스에서는 OpenSYCL이 제조사에서 제공해 주는 최적화된 런타임 커널을 사용함으로써 CUDA에 비해 약 52.5% 더 우수한 성능을 보였다. 이 결과에서 병렬 프로그래밍 모델별 성능 향상의 여지를 확인할 수 있었다.

V. 결 론

이기종 컴퓨팅은 CPU 외에 다른 종류의 프로세서를 활용하여 성능과 에너지 효율성을 향상시키는 시스템으로, 특히 GPU와 같은 이기종 프로세서는 병렬 계산이 필요한 분야에서 CPU에 비해 뛰어난 처리 속도를 보인다. 최근에는 엣지 디바이스에서도 이기종 컴퓨팅 기술이 발전하였으며, 경량화된 소프트웨어 기술을 통해 기존의 대용량 서버에서 활용되던 프로그램들을 활용할 수 있게 되었다. 이기종 컴퓨팅을 위해 기존에는 OpenMP, MPI, CUDA와 같은 병렬 프로그래밍 모델이 사용되었으며, 최근에는 OpenCL과 SYCL이 이기종 컴퓨팅을 위한 효율적인 프로그래밍 모델로 주목받고 있다.

본 논문에서는 기존에 고성능 컴퓨팅 환경에서 제안된 병렬 프로그래밍 모델들이 엣지 디바이스에서 활용 가능한지 조사하고, 각 모델의 성능을 분석하고자 하였다. 엣지 디바

표 7. 고사양 디바이스에서의 병렬 프로그래밍 모델별 주요 시스템 콜 현황
Table 7. Statistics of system calls by parallel programming frameworks on a high-end device

System call	OpenSYCL				CUDA				OpenCL				OpenMP				OpenACC			
	% time	milli sec.	usec /call	calls	% time	milli sec.	usec /call	calls	% time	milli sec.	usec /call	calls	% time	milli sec.	usec /call	calls	% time	milli sec.	usec /call	calls
brk	0.01	0.21	6	33	0.23	0.28	8	33	0.02	0.08	6	12	-	-	-	-	0.4	0.012	4	3
clone	0	0.07	14	5	0.05	0.06	56	1	0.07	0.25	42	6	-	-	-	-	-	-	-	-
close	0.23	5.92	10	587	3.39	4.15	7	587	0.49	1.87	9	207	0	0	0	9	1.22	0.037	4	9
connect	-	-	-	-	0.03	0.03	31	1	-	-	-	-	-	-	-	-	-	-	-	-
faccessat	0.01	0.15	10	14	0.05	0.06	4	14	0.05	0.21	10	19	0	0	0	1	-	-	-	-
fcntl	0.14	3.65	9	371	2.06	2.53	6	379	0.02	0.06	7	8	-	-	-	-	-	-	-	-
fstat	0.01	0.34	8	38	0.11	0.13	4	27	0.03	0.12	2	43	-	-	-	-	1.38	0.042	4	10
futex	**28.33**	736.01	20	36134	0.09	0.12	9	11	**94.71**	359.34	113	3155	**86.87**	41.55	46	885	0.53	0.016	5	3
getdents64	0	0.12	15	8	0.03	0.04	9	4	0.01	0.05	6	8	0	0	0	2	1.09	0.033	16	2
ioctl	1.08	28.14	20	1354	**84.05**	103.1	54	1889	0	0.01	11	1	-	-	-	-	-	-	-	-
lseek	0.05	1.26	9	133	0.64	0.78	5	137	0	0.02	3	5	-	-	-	-	-	-	-	-
mmap	0.08	2.04	13	147	1.2	1.47	12	118	0.41	1.54	22	68	0	0.001	0	33	5.5	0.167	5	28
mprotect	0.03	0.74	13	57	0.26	0.31	8	37	0.07	0.25	5	44					4.48	0.136	8	16
munmap	0.16	4.20	220	19	3.62	4.44	341	13	1.87	7.09	472	15	13.12	6.276	896	7	**76.7**	2.327	332	7
newfstatat	0.02	0.50	8	61	0.06	0.08	8	9	0.02	0.08	3	23	-	-	-	-	-	-	-	-
openat	0.08	2.14	15	138	0.8	0.99	15	62	1.22	4.64	16	278	0	0	0	9	3	0.091	10	9
prlimit64	-	-	-	-	0.01	0.01	5	2	-	-	-	-	-	-	-	-	0.16	0.005	5	1
read	0.03	0.76	14	53	0.32	0.39	9	40	0.56	2.11	10	195	0	0	0	7	1.12	0.034	4	7
readlinkat	-	-	-	-	0.02	0.02	11	2	0.03	0.10	11	9	-	-	-	-	-	-	-	-
rt_sigaction	-	-	-	-	0.01	0.01	5	2	0.03	0.11	6	18	-	-	-	-	0.3	0.009	4	2
sched_yield	**69.71**	1811.2	21	85368	2.61	3.20	10	305	-	-	-	-	-	-	-	-	-	-	-	-
write	0.01	0.134	4	28	0.23	0.29	10	27	0.32	1.20	4	241	0	0.01	0	13	3.39	0.103	4	22
total	100	2598.1		124584	100	122.6	3728	41	100	379.42		4405	100	47.83		1013	100	3.034		125

이스에서의 이기종 컴퓨팅 활용은 아직 제한적이지만, 향후 SYCL과 같은 모델에 대한 요구가 증가할 것으로 예상됨에 따라, SYCL을 포함한 다양한 병렬 프로그래밍 모델에 대해 살펴보았다. 본 논문에서 고수준 프로그래밍 언어로 제안된 SYCL이 엣지 디바이스에서도 우수한 성능을 보일 가능성을 확인할 수 있었으며, 실제로 SYCL 기반 응용 프로그램에서 CPU, GPU 등 다양한 컴퓨팅 자원들을 효율적으로 쉽게 활용할 수 있다면, 다양한 분야에서 더 많이 활용될 수 있을 것이다.

SYCL 프로그래밍 모델은 최신 C++ 구성과 함께 OpenCL의 이식성을 지원하며, 데이터 관리 및 동기화 같은 저수준의 세부 사항을 암묵적으로 처리하여 개발자의 생산성 향상을 기대하게 한다. SYCL의 최신 버전인 SYCL2020에 대해 많은 연구 그룹에서 지원을 진행 중인 만큼, 앞으로 다양한 이기종 프로세싱 유닛에서의 SYCL 지원이 계속 확대될 것으로 예상된다. 향후 엣지 컴퓨팅 환경에서 고성능 서버, 엣지 서버, 저사양 단말 등에서의 SYCL 기반 프로그램에 대한 계산 오프로드 연구 [25]를 진행할 계획이다.

References

[1] V. G. Cerf, “On Heterogeneous Computing,” Communication of the ACM, Vol. 64, No. 21, pp. 9, 2021.
[2] OpenCL, https://www.khronos.org/opencl/
[3] S. Mendez, “Edge Computing Systems with Kubernetes,” Packt Publishing, 2022.
[4] J. Diaz, C. Munoz-Caro, A. Nino, “A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era,” IEEE Transactions on Parallel and Distributed Systems, Vol. 23, No. 8, pp. 1369-1386, 2012.
[5] CUDA, https://developer.nvidia.com/cuda-toolkit
[6] SYCL, https://www.khronos.org/sycl/
[7] T. Deakin, J. Price, M. Martineau, S. McIntosh-Smith, "Evaluating Attainable Memory Bandwidth of Parallel Programming Models Via BabelStream," International Journal of Computational Science and Engineering, Vol. 17, No. 3, pp. 247-262, 2018.
[8] Raspberry Pi, https://www.raspberrypi.com/products/raspberry-pi-4-model-b/
[9] Jetson Orin Nano, https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/
[10] A. Alpay, V. Heuveline, "One Pass to Bind Them: The First Single-Pass SYCL Compiler with Unified Code Representation Across Backends," in Proceedings of the 2023 International Workshop on OpenCL, Article 7, 2023.
[11] OpenMP, https://www.openmp.org/
[12] OpenACC, https://www.openacc.org/
[13] D. Angus, S. Georgiev, H. A. Gonzalez, J. Riordan, P. Keir, M. Goli, "Porting SYCL Accelerated Neural Network Frameworks to Edge Devices," in Proceedings of the 2023 International Workshop on OpenCL, Article No. 4, 2023.
[14] J. Y. Park, J. H. Hong, K. S. Chung “Parallel LDPC Decoder for CMMB on CPU and GPU Using OpenCL," IEMEK J. Embed. Sys. Appl., Vol. 11, No. 6, pp. 325-334, 2016 (in Korean).
[15] S. Memeti, L. Li, S. Pllana, J. Kołodziej, C. Kessler, "Benchmarking OpenCL, OpenACC, OpenMP, and CUDA: Programming Productivity, Performance, and Energy Consumption," in Proceedings of the 2017 Workshop on Adaptive Resource Management and Scheduling for Cloud Computing, pp. 1-7, 2017.
[16] Codeplay ComputeCpp, https://developer.codeplay.com/products/computecpp/ce/home
[17] Intel oneAPI DPC++, https://www.intel.com/content/www/us/en/developer/tools/oneapi/dpc-compiler.html
[18] OpenSYCL, https://github.com/OpenSYCL/OpenSYCL
[19] STREAM Benchmark, https://www.cs.virginia.edu/stream/
[20] VC4CL: OpenCL for VideoCore IV GPU, https://github.com/doe300/VC4CL
[21] strace: Linux syscall tracer, https://strace.io/
[22] JetPack SDK, https://developer.nvidia.com/embedded/jetpack
[23] Portable Computing Language, http://portablecl.org/
[24] NVIDIA HPC SDK, https://developer.nvidia.com/hpc-sdk
[25] C. Feng, P. Han X. Zhang, B. Yang, Y. Liu, L. Guo, “Computation Offloading in Mobile Edge Computing Networks: A Survey,” Journal of Network and Computer Applications, Vol. 202, 103366, 2022.

Dukyun Nam (남 덕 윤)

1999 Computer Science and Engineering from POSTECH (B.S.)
2001 Engineering from KAIST (M.S.)
2006 Engineering from KAIST (Ph.D.)
2004~2022 Korea Institute of Science and Technology Information (Senior Researcher)
2018~2021 National Center for Supercomputing Applications (Visiting Scholar)
2022~ Kyungpook National University (Assis. Prof.)
Field of Interests: High performance computing, distributed system, heterogeneous computing
Email: dynam@knu.ac.kr