Co-design 관점의 이기종 환경을 위한 병렬 프로그래밍 디버거 아키텍처 설계

(Design of a debugger architecture for parallel programming in a heterogeneous environment from a co-design point of view)

정현미*, 정기문**
(Hyun Mi Jung, Ki Moon Jeong)

요 약

병렬프로그래밍용 가속기 기능을 포함한 RISC-V 칩용 OpenCL 디버거 개발을 통하여 이기종 환경을 위한 H/W, S/W Co-design 개발사례를 소개한다. 디버거 개발 목표는 LLDB(Low-Level Debugger)를 기본으로 하며 칩 및 호스트에서 실행되는 OpenCL 애플리케이션을 디버깅 하는 것이다. RISC-V 코어로 구성된 칩은 FP64 데이터 유형을 지원하고, 새로운 인스트럭션이 포함된 행렬/벡터 연산 가속기로 기본적인 RISC-V 디버그 서브 시스템이 포함되어 있다. 우리는 디바이스에서 실행되는 OpenCL 애플리케이션 디버깅을 위하여 RISC-V 디버그 서브 시스템을 목표에 맞게 분석하여 디바이스 개발 요구사항을 도출하였다. 이와 함께 프로그래밍 실행 모델 분석 등을 통하여 디버거 아키텍처를 설계하였다.

■ 중심어 : 디버거 ; 오픈소스 ; RISC-V, LLDB, Co-design

Abstract

This paper presents the case of H/W and S/W co-design development for heterogeneous environments through the development of OpenCL debugger for RISC-V chip with accelerator function for parallel programming. The aim of the debugger development is to debug OpenCL applications running on the chip and host based on LLDB (Low-Level Debugger). The chip consists of RISC-V cores, supports FP64 data types, is a matrix/vector arithmetic accelerator with new instructions, and includes a basic RISC-V debug subsystem. We performed a targeted analysis of the RISC-V debug subsystem for debugging OpenCL applications running on the device to derive device development requirements. We also designed the debugger architecture by analysing the programming execution model.

■ keywords : Debugger ; Open Source ; RISC-V, LLDB, Co-design

Ⅰ. 서 론

현재 HPC(High Performance Computing) 시스템은 이기종(heterogeneous) 아키텍처 형태로 진화하고 있으며 NVIDIA, Intel, AMD 등 해외 대형 벤더(Vendor) 들은 자체 가속기 및 응용프로그램 사용자를 위한 성능개선의 도구를 개발하여 제공 중이다. 소프트웨어 측면에서의 디버거는 애플리케이션의 문제를 확인하고 수정하게 도와주어 전체 노드의 성능 향상에 큰 도움을 준다. 하드웨어 측면에서는 디버깅 수행이 하드웨어 성능향상을 위한 소프트웨어 개발에서 매우 중요한 도구이다. 이에 병렬프로그램용 디버거의 기능적 한계점을 개선하고 개발되는 가속기 하드웨어의 특성에 최적화된 디버거를 개발하기

* 정회원, 한국과학기술정보연구원 슈퍼컴퓨팅기술개발센터
** 정회원, 한국과학기술정보연구원 슈퍼컴퓨팅기술개발센터
1. 이 논문은 2024년도 한국과학기술정보연구원(KISTI)의 기본사업으로 수행된 연구입니다.(과제번호: (KISTI)K24L2M1C6)
2. 이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. 2020M3H6A1084853)
접수일자 : 2024년 10월 31일 게재확정일 : 2024년 11월 12일
교신저자 : 정현미 e-mail : hmjung@kisti.re.kr

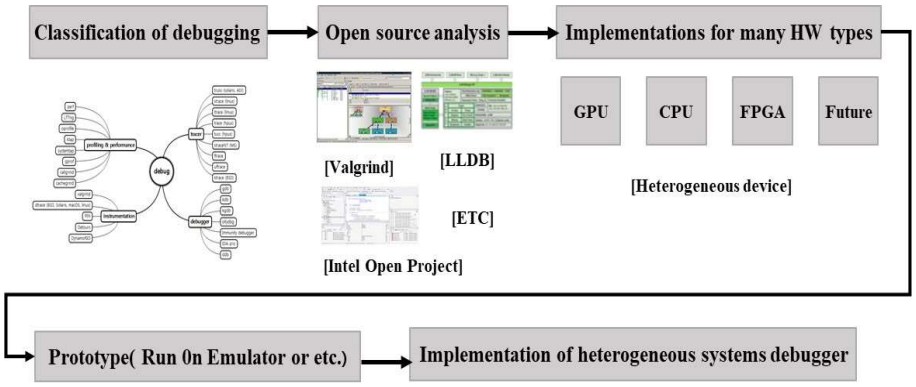

그림 1. 이기종시스템을 위한 디버거 개발 계획

위한 노력이 진행 중이다. 본 한국과학기술정보연구원(이하 KISTI) 슈퍼컴퓨팅기술개발센터는 이기종 아키텍처를 위한 OpenCL 디버거 개발을 진행하였다[1]. 그림 1은 이기종 아키텍처를 위한 디버거 개발 계획이다. 이것을 바탕으로 이기종 아키텍처 기반 시스템 사용자가 더욱 쉽게 OpenCL 커널(타겟 프로그램)의 오류를 찾을 수 있도록 도와주기 위해 아래와 같은 목표를 가지고 있다[1].

1) Open source 기반(LLDB)으로 개발되어 최대한 가속기 구조에 영향을 받지 않는 범용적인 디버거
2) OpenCL(Open Computing Language) 프로그램을 대상으로 호스트 및 기기의 커널 부분을 모두 디버깅
3) 디바이스 문제 발생 시 OpenCL 실행 디바이스에서만 디버깅 가능
4) 적용하고자 하는 가속기 모델에 대하여 RISC-V를 대상 디바이스로 설정

특히 개발 조건 '4)'에 해당하는 사항은 본 센터에서 수행하고 있는 슈퍼컴개발선도사업[3]에서 개발 중인 슈퍼컴 SoC의 기본 구조가 되는 RISC-V 칩을 대상으로 하였고 새로 개발되는 하드웨어 대상의 디버거를 이기종 시스템에 적용하기 위하여 HPC에서의 Co-Design 방법론을 바탕으로 개발하였다[4, 5]. 그러므로 본 논문의 2장에서는 HPC에서의 Co-design 개발 방법론에 대하여 분석하고 3장에서는 디버거 개발을 위하여 Co-design 방법론을 적용한 아키텍처를 설계하고 4장에서는 결론을 내린다.

Ⅱ. 관련 연구

1. HPC 에서의 Co-design 개발 방법론

Co-design은 하드웨어와 소프트웨어가 따로 개발되는 것이 아니라, 상호 최적화를 목표로 긴밀하게 협력하는 설계 방법론이다. 이는 하드웨어 성능 최적화, 응용프로그램 중심 설계 및 하드웨어 소프트웨어의 긴밀한 피드백 루프 개발을 가능하게 한다. HPC(High-Performance Computing)에서 'Hardware-Software Co-design'은 하드웨어와 소프트웨어가 서로 독립적으로 설계되는 대신, 이들이 긴밀하게 상호작용하며 최적의 성능을 발휘하도록 함께 설계되는 프로세스이며 이는 복잡한 시스템에서 성능, 에너지 효율성, 확장성 등을 극대화하기 위해 필수적이다. 구체적으로 다음과 같은 이점이 있다[4, 5].

1) 성능 최적화: 하드웨어는 특정 소프트웨어 작업량에 맞게 커스터마이즈 될 수 있고, 반대로 소프트웨어는 하드웨어의 구조에 맞춰 최적화될 수 있다. 예를 들어, AI/ML 작업량에 맞춘 특정 가속기(예: TPU(Tensor Processing Unit))소프트웨어 스택을 포함한다[6, 7].
2) 병렬성 극대화: HPC는 대규모 병렬처리가

필요하다. 하드웨어와 소프트웨어가 병렬처리에 최적화되도록 함께 설계되면, 처리 속도와 효율성이 높아진다. 이를 통해 수많은 코어에서 동시 실행되는 작업이 더 원활하게 이루어질 수 있다.

3) 에너지 효율성 향상: Co-design을 통해 특정 연산에 필요한 하드웨어 자원만을 사용하도록 하여 불필요한 에너지 소비를 줄일 수 있다. 이는 특히 대규모 데이터센터에서 중요한 이슈이기도 하다.
4) 시스템 확장성 및 유연성: 하드웨어와 소프트웨어가 협력적으로 설계되면, 시스템이 더 쉽게 확장되고 다양한 애플리케이션에 적응할 수 있다. 예를 들어, 특정 애플리케이션을 위해 하드웨어가 재구성 가능하다면, 다양한 계산 요구사항을 충족할 수 있다.
5) 디버깅 및 최적화 용이성: 하드웨어와 소프트웨어가 통합되어 설계되면, 성능 병목이나 비효율성을 더 쉽게 발견하고 해결할 수 있다.

대표적인 Co-design 사례를 보자면 ARM과 NVIDIA의 협력이나, Intel의 특정 연산을 가속화하려는 FPGA와 소프트웨어 최적화 등이 있다[4, 5]. HPC Co-design에 관한 여러 논문 중에서 2013년 'On the Role of Co-Design in High Performance Computing[4]'에 따르면 HPC 성능은 거의 10년마다 3배씩 향상되었으며, 이는 진화적 접근 방식에 따른 기술 확장과 시스템 설계(하드웨어 및 소프트웨어)의 발전으로 가능했다.

차세대 하드웨어는 소프트웨어 개발자에게 제공되었고, 소프트웨어 개발자는 제공된 하드웨어에 소프트웨어 스택을 포팅 하고 최적화하는 개발과정을 반복하고 있으며 엑사 스케일은 전체 시스템 아키텍처가 크게 달라질 것이므로 엑사 스케일 성능 목표를 실현하기 위해서는 강력한 소프트웨어 및 하드웨어 Co-design 전략을 중심으로 한 혁신적인 접근 방식이 필요하다. 다음 그림 2는 HPC에서의 Co-design 전략이다[4].

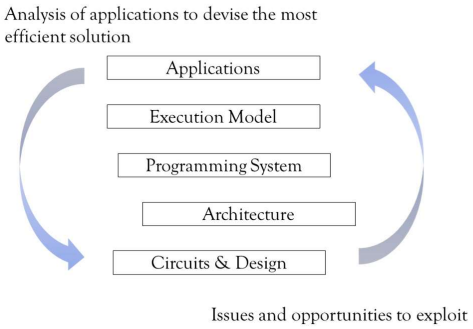

그림 2. Co-design strategy[4]

그림 2에서 볼 수 있듯이 co-design은 기반 기술부터 애플리케이션까지 전체 시스템 스택을 고려한다[4]. 애플리케이션은 컴퓨팅 패턴과 데이터 이동 패턴에 대한 인사이트를 제공하여 시스템을 최적화하고, 시스템 소프트웨어에 구현된 실행 모델은 서비스 제공 및 자원관리에 맞게 조정되며, 프로그래밍 시스템은 기반 하드웨어를 숨기고 프로그래밍 생산성을 제공한다. 아래로부터 기술적인 문제와 기회를 이해하고, 중간에서 시스템 아키텍처를 고안하는 노력이 필요하다. 2021년 'Co-design In High Performance Computing Systems[5]' 논문에 의하면 미국 에너지부(DOE)는 2014년 1월 차세대 슈퍼컴퓨터 시스템 조달을 위한 제안요청서(RFP)를 발표하면서 CORAL(Collaboration among Oak Ridge, Argonne and Livermore) 프로그램을 공식적으로 시작하였다. 이 프로그램에 따르면 대규모 고성능 컴퓨터 시스템을 구축하려면 향후 의도한 시스템의 목표 기능을 정의하고, 시스템이 개발될 때 시스템 사양을 검증하며, RFP에 명시된 사전 정의된 벤치마크 세트에 따른 성능 예측을 추적하는 철저한 개발 프로세스가 필요하다. 이러한 예측은 새로운 아키텍처를 갖춘 미개발 시스템의 경우 가장 엄격하게 적용되는데, 가장 큰 슈퍼컴퓨터의 경우 미래의 요구에 대응하여 설계 및 개발되고 새로운 기술과 아키텍처를 주도하기 때문에 Cc-Design이 중요한 요소로 작용한다.

III. Co-design을 통한 디버거 개발과정

본 장에서는 Co-design을 통한 디버거 개발과정을 보여준다. 디버거 개발 목표는 타겟 애플리케이션(OpenCL)의 문제를 확인하고 수정하게 도와주어 타겟 디바이스는 물론 전체 노드의 성능을 향상시키는 것이다. 타겟 디바이스는 앞서 언급한 것처럼 슈퍼컴퓨터개발선도사업에서 개발하고 있는 SoC의 기반인 머신 모드 전용의 RISC-V 칩을 이용하였고 다음과 같은 프로세스로 디버거 아키텍처를 설계하였다.

- RISC-V 구조분석 및 설계 요구사항 도출
- 프로그래밍 실행 모델 분석 및 설계
- LLDB Extension 프로토타입 개발
- 프로토타입 개선사항을 포함한 하드웨어 맞춤형 디버거 아키텍처 설계 및 구현

1. RISC-V 구조분석 및 설계 요구사항 도출

디버거 개발을 위하여 첫 번째로 RISC-V 칩에 대한 구조를 분석하였다. 대상 디바이스 설계 시 여러 가지 기능을 고려한 공간 효율화를 위하여 칩 내에 전체 디버거 유닛 검토를 통해 디버깅 기능이 엑세스되는 경로를 파악하였다. 다음 그림 3은 RISC-V 의 기본 디버그 유닛이다[10].

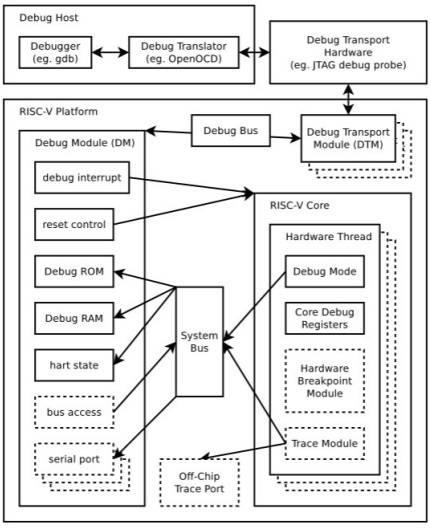

그림 3. RISC-V 디버그 유닛[10]

검토 결과 디버깅 수행을 위한 디버그 유닛 모듈 엑세스 방안은 다음과 같다.

1) Debug Module Access 방안

DM Register를 직접 엑세스 해서 Halt/Resume 및 기타 Debug 기능을 수행하는 방안이다. 이는 반드시 하드웨어 설계 시 반영되어야 하며 다양한 Debug 기능 제공된다.

2) Trigger Module (Option) 추가 방안

만약 칩 설계 시 다른 필수 기능 구현을 위하여 공간 효율성을 고려한다면 구현 불가능하다. 이 유닛은 DM 없이도 기본적은 Debug 동작(Halt/Resume)을 수행할 수 있는 유닛이다. 또한 Watch Point, Conditional break을 수행하는 데 필요한 유닛이다. 트리거는 특별한 명령어를 실행하지 않고도 중단점 예외, 디버그 모드 진입 또는 트레이스가 가능하다. 따라서 ROM에서 코드를 디버깅 시 유용하다. 또한 지정된 메모리 주소에서 명령어가 실행되거나 로드/저장된 주소/데이터에서 트리거가 가능하다. 이러한 기능은 모두 디버그 모듈이 없어도 유용할 수 있는 기능이지만 별도 추가가 필요하다. 두 번째로 애플리케이션 실행 최적화를 위하여 디바이스 설계 요구사항을 도출하였다. 타겟 디바이스는 머신 모드 전용의 32비트 연산할 수 있는 RISC-V 칩이다. 이에 64비트 연산이 필요한 병렬프로그래밍용 디버거 개발 시 수정이 필요하다. 64 비트 데이터 및 주소 처리가 가능하게 디버그 서브 시스템의 모든 레지스터를 수정 하여야 한다. 또한 소프트웨어가 레지스터에 접근하기 위한 경로 생성은 필수이다.

2. 프로그래밍 실행 모델 분석 등을 통한 디버거 아키텍처 설계

디버깅 대상인 OpenCL 커널을 디버그하기 위하여 LLDB를 이용하였다. LLDB는 LLVM 프레임워크의 세부 기능을 직접 활용할 수 있어 다양한 기능을 빠르게 구현하고 쉽게 확장할 수 있는 공개 프로그램이다. LLDB와 OpneCL은 상용

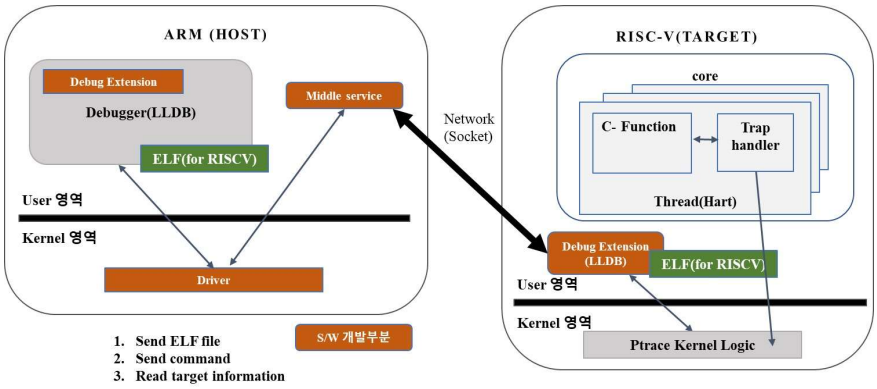

그림 4. 디버거 개발 시나리오

디바이스에서는 동작한다. 그러나 새로 설계되는 디바이스에는 바로 적용할 수 없으므로 Extension 부분과 디바이스 드라이버와 연결되는 Low-level 인터페이스 개발이 꼭 필요하다. 이에 오픈소스 디버거인 LLDB 동작 방식을 상세 분석하고 확장하여 디버거 시나리오 작성하였다. 다음 그림 4은 요구사항을 반영한 1차년도 디버거 개발 시나리오 이다.

3. LLDB Extension 개발

그림 4의 구현을 위하여 상용 RISC-V 보드를 구매해서 Target 보드로 사용하고 Host쪽 디버거와 Driver를 개발하였다. 향후 middle service에 해당하는 Daemon 제거하고 Driver와 개발하고자 하는 병렬프로그래밍용 RISC-V칩 연동으로 바꿀 수 있도록 프로토타입을 개발하였다. 이 버전에서 커널 드라이버는 LLDB에서 전달받은 데이터를 middle service로 전달하는 역할을 하지만 추후 타겟 RISC-V 가속기로 직접 데이터를 전달하고 middle service는 삭제한다. 다음 그림 5는 프로토타입 결과물이다.

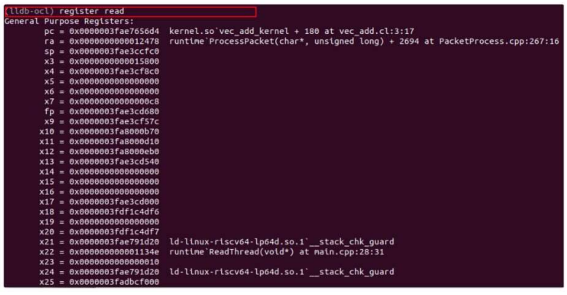

그림 5. 프로토타입 구현 화면

4. 프로토타입 개선사항을 포함한 하드웨어 맞춤형 디버거 아키텍처 설계 및 구현

개발하고자 하는 디버거는 LLDB 기반이기 때문에 모든 LLDB 기본 기능을 포함하고 있다. 앞서 보여준 것과 같이 프로토타입에서는 통신을 위하여 middle service를 구현하였으나 타겟 하드웨어의 구조가 완성되면 이 부분을 삭제한다. 또한 타겟 하드웨어의 기능적 구현이 완성되지 않았기 때문에 타겟 보드와의 디버깅 인터페이스 구현 검증을 1차 목표로 다음과 같이 디버거 요구사항을 도출하였다.

우선 오픈 소스 기반의 LLDB와 OpneCL은 일반적인 상용보드에서는 동작하지만 새로 설계되는 RISCV 기반의 가속기에는 바로 적용할 불가 Extension 부분과 디바이스 드라이버와 연결되는 디바이스 라이브러리 Interface 개발 필요하다. 이에 새로운 가속기와 Linux Kernel Driver를 통해서 컨트롤 할 수 있도록 Interface와 protocol 설계 및 구현하는 것이 필요 하다. 기 구현된 Debug extension 부분을 가속기 전용 block으로 새로 추가해서 구현한다. 이는 RISC-V를 기반으로 구현되는 신규 가속기에 맞춰 새로운 instruction, register, 컴파일러 구현이 필요하다. 위의 과정들로 도출된 디버거 기능 요구사항은 다음과 같다.

- 모든 표준 LLDB 디버깅 기능 구현
- 단일 세션 내에서 원활한 CPU 및 타겟

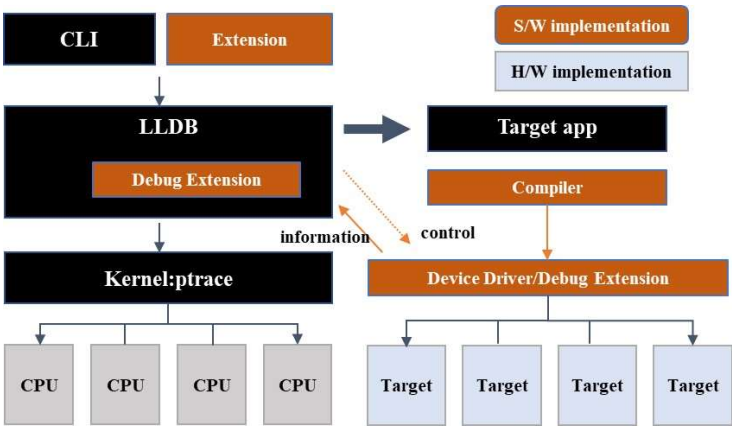

그림 6. 디버거 아키텍처

디바이스 디버깅 기능 구현

- Attaching/Detaching 기능
- Breakpoints 및 Wachpoints 설정 기능
- 프로그램 상태 검사 기능(메모리, 변수, 레지스터, 로컬/공유/글로벌 변수 검사 등)
- 다중 타겟 디바이스, 다중 컨텍스트, 다중 커널 지원 기능
- 이벤트 알림 기능
- 소스 및 어셈블리 수준 디버깅 기능
- 자동 오류 검사 기능(스택 오버플로와 같은 메모리 오류 검사) 등

지금까지 분석한 기능 요구사항 등을 토대로 최종적으로 설계된 디버거 아키텍처는 다음 그림 6과 같다. 그림은 H/W, S/W Co-design 시 각각 영역에서 개발하여야 하는 부분을 포함하고 있다.

Ⅳ. 결론

우리는 새로운 병렬프로그래밍용 디바이스를 대상으로 Co-design의 관점으로 디버거를 설계하고 구현하였다. 가속기마다 서로 다른 프로그래밍 모델을 사용하기에 디버거 역시 새로운 하드웨어와 프로그래밍 모델의 특성을 반영해서 모두 다른 형태로 만들어져야 한다. 이에, 복잡한 이기종 환경을 위한 디버거는 서로 다른 가속기의 특성에 따른 디버깅 결과 값에 대한 신뢰도 문제 등이 발생한다. 이를 극복하기 위하여 새로운 하드웨어 (타겟 디바이스)설계를 위한 디버깅 요구사항 도출 및 반영, 타겟 디바이스 디버깅 요구사항을 소프트웨어 개발에 반영 하는 등 순환적 구조를 통하여 H/W와 S/W를 최적화하면서 동시개발 하였다. 향후 이러한 개발 경험 축적을 통하여 좀더 다양한 디바이스에서 실현 가능한 다양한 S/W 및 디버거 개발이 가능할 것으로 예상 된다.

REFERENCES

[1] 정현미, 구기범, 오광진, 초고성능 컴퓨터 가속기(GPU)를 위한 디버거 분석, 한국과학기술정보연구원, 45쪽, 2022년
[2] 정현미, 구기범, 초병렬프로세서의 OpenCL 커널 디버거를 위한 SCR1의 Debug Subsystem 설계 변경, 한국과학기술정보연구원, 113쪽, 2022년
[3] 슈퍼컴퓨터개발선도사업,(2020). https://nrf.re.kr/biz/info/info/view?menu_no=378&biz_no=445 (accessd Sep., 15, 2024).
[4] 정현미, 여준기, 구기범, 슈퍼컴퓨터 SoC를 위한 OpenCL Debugger 개발, 대한전자공학회 2023 하계종합학술대회, 2023년
[5] 정현미, 구기범, 오광진, 박희민, LLDB를 이용한 병렬프로그래밍 디버깅 기술분석, 114쪽, 2023년
[6] "Cloud Tensor Processing Units (TPUs)", Google Cloud. Retrieved 20 Jul. 2020.
[7] Jouppi et al, "In-Datacenter Performance Analysis of a Tensor Processing Unit", arXiv:1704.04760 2017.
[8] Richard F. BARRETT a et al. (Eds.), On the Role of Co-design in High Performance Computing,T ransition of HPC Towards Exascale Computing, IOS Press, 2013.
[9] Jaime H. Moren et al. , Co-design In High Performance Computing Systems, 2021 IEEE International Electron Devices Meeting (IEDM), 2022.
[10] RISC-V Debug Module Implementation,

https://gist.github.com/brabect1/a39c5470b4cf49524919bfb3e3f20a5c (accessd Sep., 15, 2024).
[11] Syntacore, SCR1 External Architecture Specification, v1.2.9, 2022.
[12] Tim Newsome and Megan Wachs, RISC-V External Debug Support Version 0.13.2, 2022.
[13] Baris Aktemuret al., Debugging SYCL Programs on Heterogeneous Intel Architectures, International Workshops on OpenCL, 2020.
[14] Christopher Erband Joseph L. Greathouse, clARMOR: A Dynamic Buffer Overflow Detector for OpenCL Kernels, International Workshops on OpenCL, 2018.
[15]B en Ashbaugh, Debugging and Analyzing Programs Using the Intercept Layer for OpenCL Applications, International Workshops on OpenCL, 2018.
[16] Anshuman Verma et al., Developing Dynamic Profiling and Debugging Support in OpenCL for FPGAs, Proceedings of the 54thAnnual Design Automation Conference, 2017.
[17] James Price and Simon McIntosh-Smith, Oclgrind: An Extensible OpenCL Device Simulator, International Workshops on OpenCL, 2015.

저 자 소 개

정현미(정회원)

2010년 한남대학교 컴퓨터공학과 석사 졸업.
2014년 한남대학교 컴퓨터공학과 학과 박사 졸업.
<주관심분야 : HPC, 이기종컴퓨팅 , 병렬컴퓨팅, 클라우드>

정기문(정회원)

2001년 전남대학교 전산통계학 석사 졸업.
2009년 전남대학교 정보보호학 박사 졸업.
<주관심분야 : HPC 클라우드, 이기종 컴퓨팅, 클라우드 보안>