# HSA 런타임 API를 이용한 hipSYCL 런타임 백엔드 개발

명훈주, 구기범
한국과학기술정보연구원 슈퍼컴퓨팅 기술개발센터
hjmyung@kisti.re.kr, gibeom.gu@kisti.re.kr

# Development of hipSYCL Runtime Backend Using HSA Runtime API

Hunjoo Myung, Gibeom Gu
Center for Supercomputing Technology Development, Korea Institute of Science and Technology Information

요 약

SYCL은 OpenCL 디바이스를 위해 추상화한 C++ 프로그래밍 모델이다. OpenCL에 비해 SYCL은 높은 생산성 등 C++이 가지고 있는 강점을 보유하며, 인텔이 이기종 컴퓨팅을 위한 개발 언어로 SYCL 기반의 DPC++을 출시함에 따라 많은 주목을 받고 있다. 우리는 여러 SYCL 구현물들 중에서 NVIDIA, AMD 등 다양한 GPU를 지원하고, 코드의 수정 및 추가가 용이한 hipSYCL를 채택하여 여러 연구를 진행하고 있다. 본 논문에서는 hipSYCL 구조 내에 AMD GPU를 위한 HIP 백엔드 플러그인을 대체할 수 있는 새로운 백엔드 플러그인을 제안한다. 이 플러그인은 HSA 런타임 API를 사용하여 기존의 플러그인보다 계층 구조를 줄이고 경량화하였다.

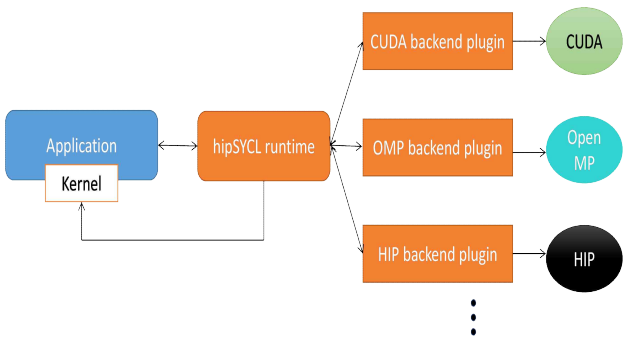
## 1. 서론

SYCL[1]은 OpenCL[2] 디바이스를 위한 표준 C++ 기반의 프로그래밍 모델이다. SYCL은 OpenCL와 마찬가지로 크로노스 그룹[3]에서 표준을 제정하므로, 벤더에 종속적이지 않은 코드를 개발할 수 있다. 반면에 C++ 템플릿, 람다 함수, 추상화 등의 표준 C++의 특성 등을 사용하여 프로그램 개발에 있어서 보다 높은 생산성을 제공한다. 또한, 인텔이 이기종 컴퓨팅을 위한 기본 개발 언어로 SYCL 기반의 DPC++[4]를 출시하면서, SYCL이 점차 많은 주목을 받게 되었다.

SYCL의 구현물들은 DPC++ 이외에도, hipSYCL[5], 자일링스의 triSYCL[6], CodePlay의 ComputeCpp[7] 등이 존재하는데, 우리는 2장에서 설명할 hipSYCL의 여러 가지 장점들에 주목하여 이것을 기반으로 여러 연구를 수행하고 있다. 본 연구에서는 hipSYCL의 AMD GPU를 위한 HIP 백엔드 플러그인보다 경량화한 새로운 백엔드 플러그인을 제안한다.

본 논문은 다음과 같이 기술되어 있다. 2장에서는 hipSYCL 개요 및 구조에 대해 설명하고, 3장에서는 새로운 백엔드 플러그인의 주요기능에 대해 구현 내용을 기술한다. 4장에서는 추후 계획에 대해 언급하고 끝맺는다.

## 2. hipSYCL 개요 및 구조

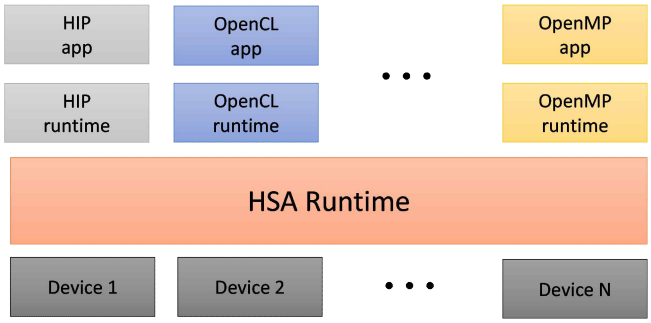



<그림 1> hipSYCL 런타임 구조

hipSYCL은 현재 하이델베르그 대학교에서 개발 중인 SYCL 구현물로, CPU와 NVIDIA, AMD 등 주요 GPU을 타겟으로 한다. hipSYCL은 SMCP (single-source multiple compiler-passes) 기법과 함께 기존 개발도구를 최대한 활용하는 것을 기본 방향으로 삼고 있다. 이 부분이 프로그램의 안정성과 더불어 우리가 연구에 사용할 SYCL 구현물로 hipSYCL을 선정한 이유이다. 참고로 인텔의 DPC++은 인텔 컴파일러와 통합되어

있어, hipSYCL과 비교하면 상당히 무거운 편이다.

hipSYCL 런타임은 <그림 1>에서 보는 바와 같이, 각각의 백엔드마다 독립적이고 모듈형태의 플러그인 라이브러리와 연동하고 있다. 이러한 구조는 새로운 디바이스에 대한 백엔드 플러그인을 추가하기 쉬운 점 등의 높은 확장성을 가진다.




<그림 2> AMD GPU를 사용하는 애플리케이션 계층 구조

한편, AMD GPU를 이용하는 애플리케이션은 <그림2>와 같은 계층 구조를 갖는데, hipSYCL을 이용하여 빌드를 하게 되면 HIP[7] 런타임을 사용하게 되므로, <그림2>의 OpenCL app등 다른 애플리케이션과 다르게, "HSA (Heterogenous System Architecture)[8] 런타임-HIP 런타임-SYCL 런타임"의 계층구조를 갖게 된다. 다시 말해서, hipSYCL로 빌드된 애플리케이션은 AMD GPU를 사용하는 다른 애플리케이션에 비해 한 계층이 더 많다.

## 3. HSA 런타임 API를 이용한 백엔드 플러그인 구현

2장에서 언급한 문제를 개선하기 위해, 우리는 HSA 런타임을 API를 활용하여, HIP 백엔드 플러그인을 대체하는 AMD GPU를 위한 새로운 백엔드 플러그인을 구현하고 있다. 이를 통해, AMD GPU를 이용하는 SYCL 애플리케이션은 보다 경량화될 것으로 기대한다.

백엔드 플러그인에서 제공해야 할 주요 기능은 메모리 할당 및 데이터 복사, 커널 코드의 적재, 커널 수행 등이다. HSA 런타임 API를 사용하기 위해서는 우선 hsa_init()함수를 호출하여 HSA 런타임을 초기화하여야 한다.

### 3.1 메모리 할당 및 데이터 복사

GPU의 메모리 할당을 위해서 hsa_memory_allocate() 함수를 사용한다. 그런데 이를 호출하기 전에 수행하려는 GPU의 메모리가 어떤 영역(region)들로 구성되어 있는지 등의 정보를 획득한 후에, 파라미터로 메모리 할당을 위한 영역을 명시해야한다. 이를 위해서는 hsa_agent _iterate_regions() 함수 사용 및 콜백 함수 작성이 필요하다. 일반적으로 AMD GPU는 global region이 대부분이므로 이 영역에서 메모리를 할당한다.

한편, 호스트 메모리와 GPU 메모리 간의 데이터 복사는 hsa_memory_copy()함수를 사용하고 사용 방법은 memcpy()함수와 동일하다.

### 3.2 커널 코드의 적재

HSA에서는 HSAIL[8]이 기본적으로 디바이스 코드로 사용되나, hipSYCL를 이용하여 소스 코드를 컴파일 하게 되면 실행 파일이 만들어지므로, 이것을 직접 다루는 것이 훨씬 효율적이다.

hipSYCL이 빌드한 실행 파일은 HIP과 동일하게 멀티 아키텍처 바이너리이며, 이것에서 AMD GPU의 디바이스 코드인 code object[9]를 추출하여 적재해야 한다. 즉, 실행파일에서 code object의 오프셋과 크기 등의 정보를 알아야 하는데, 이것은 AMD에서 제공하는 roc-obj-ls 명령어를 통해서 얻을 수 있다.

실행 파일로부터 실행해야할 커널 함수의 핸들러를 구하기까지의 과정과 그에 필요한 함수는 <표 1>와 같다.

<표 1> 커널 핸들러를 얻는 작업 순서 및 그에 따라 사용되는 HSA 런타임 API

| 순서 | 작업내용 | 사용되는 HSA 런타임 API |
|---|---|---|
| 1 | code object 읽기 | hsa_code_object_reader_create_from_memory() |
| 2 | code object를 executable 객체에 적재 | hsa_executable_load_agent_code_object() |
| 3 | executable의 잠금 (수정제한) | hsa_executable_freeze() |
| 4 | 실행할 커널함수 검색 | hsa_executable_get_symbol() |
| 5 | 커널 함수의 파라미터 영역 크기 구하기 | hsa_executable_symbol_get_info() |
| 6 | 커널 함수의 핸들러 구하기 | hsa_executable_symbol_get_info() |

### 3.3 커널 수행

HSA 런타임을 이용한 커널 수행은 커널을 수행할 커널 에이전트에 큐를 생성하고, HSA Architected Queuing Language (AQL)[8]를 사용하여 명령 패킷들을 생성, 큐에 적재하고 수행한다.

## 4. 결론

우리는 본 연구를 통해 hipSYCL에서 사용할 수 있는

새로운 백엔드 플러그인을 제안하였다. 이 플러그인은 HIP 런타임을 사용하지 않고, HSA 런타임 API를 사용하여 HIP 백엔드 플러그인보다 단순하고 가벼울 것으로 기대한다.

앞으로 이벤트 처리, 동기화 처리 등을 구현하여 새로운 백엔드 플러그인을 완성할 계획이다.

이 논문은 대한민국 정부(과학기술정보통신부)의 재원으로 한국과학기술정보연구원 초고성능컴퓨팅 공동활용을 위한 통합 환경 개발 및 구축 사업과 한국연구재단 슈퍼컴퓨터개발선도사업의 지원을 받아 수행된 연구임 (과제번호 : 2020M3H6 A1084857)

## 참고문헌

[1] Khronos SYCL working group. 2020. SYCL 1.2.1 specification. Standard. Khronos Group, Inc, Beaverton, OR, USA. https://www.khronos.org/registry/SYCL/specs/sycl-1.2.1.pdf

[2] https://khronos.org

[3] Khronos OpenCL working group. 2019. OpenCL 2.2 specification. Standard. Khronos Group, Inc, Beaverton, OR, USA. https://www.khronos.org/registry/OpenCL/specs/2.2/pdf/OpenCL_API.pdf

[4] Intel Corporation. [n.d.]. SYCL* Compiler and Runtimes. https://github.com/intel/llvm

[5] Aksel Alpay. [n.d.]. hipSYCL. https://github.com/illuhad/hipSYCL

[6] The triSYCL project. [n.d.]. triSYCL. https://github.com/trisycl/trisycl.

[7] Codeplay Software. [n.d.]. ComputeCpp. https://codeplay.com/products/computesuite/computecpp.

[8] HSA Foundation. 2018 HSA Platform System Architecture Specification (Version 1.2). http://hsafoundation.com/wp-content/uploads/2021/02/HSA-SysArch-1.2.pdf

[9] AMD Inc. ROCm Code Object Format. https://rocmdocs.amd.com/en/latest/ROCm_Compiler_SDK/ROCm-Codeobj-format.html