ezSYCL:SYCL을 위한 파이썬 랩퍼

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

ezSYCL:Python Wrapper for SYCL

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

요 약

SYCL은 크로노스 그룹(Khronos Group)이 개발한 C++ 기반의 이기종 아키텍처를 위한 공개 표준 언어이다. SYCL은 벤더에 종속되지 않는 개방형 표준이며, 특히 인텔이 이기종 아키텍처의 주력 언어로 채택하고 있다는 장점이 있다. 그러나 SYCL의 템플릿, 람다 함수 등 C++의 복잡한 기능들과 태스크 그래프 기반 수행 방식 등의 여러 특징은 개발자들에게 높은 진입 장벽으로 작용하고 있다. 본 논문에서는 이러한 문제를 해결하기 위해 파이썬 기반 래퍼인 ezSYCL의 설계를 제안한다. ezSYCL은 SYCL보다 간결한 표현을 제공하여 개발자들이 보다 빠르게 프로토타입을 개발할 수 있을 것으로 기대된다.

1. 서론

GPU 컴퓨팅으로 중심으로 하는 이기종 컴퓨팅은 AI 기술이 다양한 분야에 적용됨에 따라, 중요한 흐름으로 자리잡고 있다. 이를 지원하는 개발 도구로는 CUDA, OpenCL, HIP 등이 널리 사용되고 있지만, CPU 기반의 순차 프로그래밍에 비해 난이도가 높아 진입 장벽이 높은 것도 사실이다. 이를 해결하고자 지시어 (directive) 기반의 OpenACC[1], OpenMP[2]이 제안되었으며 파이썬 랩퍼인 pyCUDA[3], pyOpenCL[4]도 등장하였다. SYCL[5] 역시 표준 C++을 기반으로 하는 이기종 프로그래밍 모델로 여러 가지 장점을 지니고 있지만, C++에 익숙하지 않은 개발자에게는 템플릿, 람다함수 등 다양한 개념들이 SYCL의 진입장벽으로 작용하고 있다.

본 논문에서는 SYCL의 진입 장벽을 낮추고, 프로토타입을 빠르게 할 수 있는 ezSYCL을 소개한다. 2장에서는 관련 연구를 다루고 3장에서는 설계 목적과 주안점, 주요 기능을 설명한 후, 4장에서 결론을 제시한다.

2. 관련 연구

2.1 Data Parallel Control

Data Parallel Control (dpctl)[6]은 인텔에서 제작한 파이썬 라이브러리로 사용자가 이를 통해 계산 커널 실행할 수 있다. dpctl는 SYCL 표준 기반에서 제작되었으며, 이 라이브러리의 주요 특징은 다음과 같다.

- 커널 코드는 numba-dpex*를 이용해 파이썬으로 작성하거나, oneMKL와 같이 라이브러리의 일부가 될 수 있다.

  *인텔에서 제작한 numba[7] 컴파일러의 확장으로, 이를 이용해 CPU에서 프로그래밍하듯이 GPU와 같은 데이터 병렬 장치에서 프로그래밍을 할 수 있도록 지원하는 JIT (Just-In-Time) 컴파일러임
- SYCL의 통합공유메모리 (USM: Unified Shared Memory)를 지원한다.
- 디바이스 찾기, 컨텍스트 생성, 큐 생성 등과 같은 표준 SYCL 런타임 클래스의 일부를 바인딩하여 파이썬 환경에서 제공한다.

dpctl은 Intel OneAPI 소프트웨어 기반에서 동작한다. 그리고 앞서 언급했듯이, 커널 코드는 numba 컴파일러를 이용해 파이썬 코드로 작성되므로, 개발자 입장에서는 SYCL과 연관성을 직접 체감하지 못할 수도 있다.

2.2 pyCUDA

pyCUDA는 NVIDIA CUDA API를 파이썬으로

접근가능하게 해주는 라이브러리이다. pyCUDA의 특징은 다음과 같다.

• RAII[8] 개념을 채택하여, CUDA 자원의 관리와 정리 (cleanup)를 자동적으로 수행한다.

• GPU 디바이스의 관리 및 계산을 위해 GPUArray 클래스를 제공하며, 이 클래스는 numpy.ndarray[9]와 동일한 기능을 갖추고 있다.

• 자동 오류 체크 기능이 있다.

• CUDA 드라이버 API 수준으로 프로그래밍이 가능하다.

3. ezSYCL의 설계

3.1 목적 및 설계 방향

ezSYCL의 목적은 앞서 언급한 SYCL의 복잡한 구조와 문법들을 단순화하여 SYCL의 진입 장벽을 낮추고, 사용자로 하여금 빠른 프로토타입핑을 할 수 있도록 돕는 것이다. 이러한 맥락으로 ezSYCL은 dpctl과는 다르게 SYCL의 기본 개념을 알고 있는 개발자라면 쉽게 작성할 수 있도록 SYCL의 단순화에 주력하였다. 이를 통해 ezSYCL로 빠른 프로토타입 작성을 가능하게 하며, 이후 최적화 작업 혹은 SYCL 프로그래밍을 통해 프로그램의 완성도를 높이는 것을 목표로 한다.

3.2 설계 주안점

• USM 메모리 이용한 데이터 이동 방식 채택

SYCL에서는 데이터 이동에 있어서 3가지 방법을 제공한다. 첫 번째는 copy 함수를 이용하는 명시적인 방법이며, 두 번째는 USM 메모리를 활용하는 암시적인 방법, 그리고, 세 번째는 Buffer 클래스를 사용하는 방법이다. ezSYCL에서는 명시적인 방법에 비해 성능은 다소 떨어질 수 있지만, 프로그래머에게 더 직관적이고 신속한 프로그래밍이 가능한 암시적인 방법을 채택 데이터 이동을 설계하였다.

• ezSYCL 작업 실행 모델

SYCL은 태스크 그래프 (task graph) 기반 실행 모델을 채택하고 있는데, 이를 통해 런타임 동안 작업의 의존성과 시스템 자원을 고려하여 최적의 실행 경로를 선택할 수 있다. 이러한 접근 방식은 병렬 실행 성능을 최적화하는 데 유리하지만, 개발자가 작업의 의존성과 실행 순서를 명확히 이해해야 하는 점에서, 도전이 될 수 있다. 복잡한 태스크 그래프와 동시 실행의 조합은 디버깅을 더욱 어렵게 만들 수 있다. 이에 반해, ezSYCL은 단순한 큐 기반 병렬 처리 모델을 채택하여 커널 함수를 제출하는 submit() 함수와 동기화를 시키는 wait()함수를 제공한다. 이 모델은 개발자가 단순하고 직관적으로 프로그래밍 환경을 제공한다. 추후에는 커널 함수 간의 의존성을 표현하여 태스크 그래프 기반으로 실행할 수 있는 방법도 제공할 계획이다.

• 데이터 선처리 클래스 제공

파이썬은 과학 계산을 위한 강력한 라이브러리인 numpy를 제공하며, ezSYCL에서는 이 라이브러리와 자연스러운 연동을 통해 손쉽게 데이터 전처리를 수행할 수 있게 한다. 아울러 USM 메모리와의 심리스한 연동을 통해 개발자가 보다 직관적으로 프로그래밍을 할 수 있도록 한다.

3.3 설계 주요내용

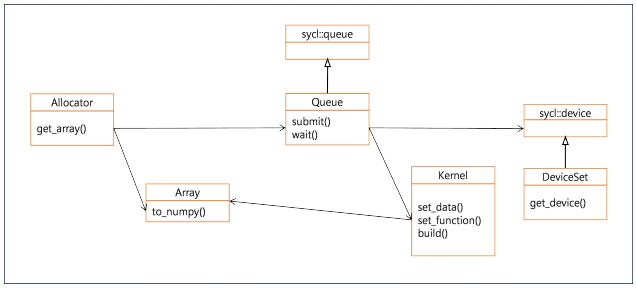
ezSYCL은 앞서 언급한대로 C++기반인 SYCL을 개발자가 파이썬 언어로 작성할 수 있게 해 주는 도구로, 이를 위해 pybind11[10] 라이브러리를 사용하여 구현하였다. pybind11을 사용하여 주요 SYCL 클래스들을 파이썬과 연동하였으며, ezSYCL 설계에서 중점을 둔 기능들을 구현하기 위해 C++ 클래스와 python 클래스를 추가로 작성하였다. 다음은 ezSYCL의 주요 파이썬 클래스들이다.

• Allocator: USM 메모리를 관리하는 클래스이며, 지정한 데이터 타입과 크기에 맞는 Array 객체를 반환한다 (get_array 함수).

• Kernel: 디바이스에서 실행될 커널을 관리하는 클래스로, 커널 코드를 작성하고 (set_function 함수), build 함수를 사용하여 실행 가능한 바이너리 파일을 만든다.

• Queue: sycl::queue를 상속받은 클래스로, Kernel 객체를 입력받아 작업을 제출 (submit 함수)하고, 필요에 따라 작업이 완료될 때까지 동기화 (wait 함수)한다.

• Array: 지정된 데이터 타입과 크기에 맞춰 USM 메모리를 할당한 후, 이렇게 할당받은 메모리를 그대로 활용하는 numpy.ndarray 객체를 제공한다 (to_numpy 함수).



(그림 1) ezSYCL의 주요 클래스다이어그램

3.4 ezSYCL 프로그래밍의 예

```
#include <iostream>
#include <sycl/sycl.hpp>
using namespace sycl;

const std::string secret{
  "Ifmmp-!xpsme\"\012J(n!tpssz-!Ebwf/!"
  "J(n!bgsbje!J!dbo(u!ep!uibu/!.!IBM\01"};

const auto sz = secret.size();

int main() {
  queue q;

  char* result = malloc_shared<char>(sz, q);
  std::memcpy(result, secret.data(), sz);

  q.parallel_for(sz, [=](auto& i) {
     result[i] -= 1;
    }).wait();

  std::cout << result << "\n";
  free(result, q);
  return 0;
}
```

(그림 2) SYCL의 “Hello, World”코드

그림 2는 Data Parallel C++[11]에 포함된 예제로, 주어진 문자열의 각 문자에 대해 커널 함수에서 해당 문자 코드보다 1 작은 값으로 변환한 뒤, 그 결과를 출력하는 내용을 담고 있다. 이 코드를 ezSYCL을 사용해서 작성한 코드는 그림 3과 같다. 이 예제의 주요 흐름은 다음과 같다. 먼저 ezsycl 패키지를 임포트한 후, 커널 작업 제출에 필요한 Queue 객체를 생성하고 처리할 문자열을 USM 메모리를 사용하는 Array 객체에 저장한다. 이후 Kernel 객체를 생성하여 커널 함수를 작성하고, 커널 함수가 필요로 하는 데이터를 정의한 다음, 실행 가능한 바이너리로 빌드한다. 이렇게 생성된 커널은 Queue 객체의 submit 함수를 사용해 작업을 제출된다. 마지막으로, 커널 작업 수행 완료를 확인한 후 결과를 출력한다.

```
import ezsycl

Queue = ezsycl.Queue()
allocator = ezsycl.Allocator(queue)

string = allocator.get_array( \
            "Ifmmp-!xpsme\"\012 J(n!tpssz-!Ebwf/!" \
           +"J(n!bgsbje!J!dbo(u!ep!uibu/!.!IBM\01")

kernel = ezsycl.Kernel()
kernel.set_data(chars=string)
kernel.set_function( """
           chars[id] = chars[id] -1; """)

queue.submit(len(string), kernel)
queue.wait()

print("".join(string))
```

(그림 3) ezSYCL의 “Hello, World” 코드

4. 결론

본 논문에서는 보다 쉽게 SYCL 프로그래밍을 작성할 수 있도록 돕는 파이썬 래퍼, ezSYCL를 소개하였다. ezSYCL를 통해 SYCL 프로그래밍의 진입 장벽이 낮아질 것으로 기대한다. 향후에는 태스크 그래프 기반 스케줄링 등 SYCL에서 제공하고 있는 다양한 기능을 더욱 간결하게 프로그래밍할 수 있도록 지속적으로 보완해 나갈 계획이다.

이 논문은 대한민국 정부 (과학기술정보통신부)의 재원으로 한국과학기술정보연구원 초고성능컴퓨팅 공동활용을 위한 통합 환경 개발 및 구축 사업의 지원을 받아 수행된 연구임 (과제번호: K24L2M1C6)

참고문헌

[1]https://www.openacc.org
[2]https://www.openmp.org/wp-content/uploads/2021-10-20-Webinar-OpenMP-Offload-Programming-I

ntroduction.pdf
[3]https://documen.tician.de/pycuda/
[4]https://documen.tician.de/pyopencl/
[5]https://www.khronos.org/sycl/
[6]https://github.com/IntelPython/dpctl
[7]https://numba.pydata.org/
[8]https://learn.microsoft.com/ko-kr/cpp/cpp/object-lifetime-and-resource-management-modern-cpp?view=msvc-170
[9]https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html
[10]https://github.com/pybind/pybind11
[11]https://link.springer.com/book/10.1007/978-1-4842-9691-2