**고급소프트웨어실습**

**기말 리포트**

**(CSE 4152)**

**Due: 2020년, 12/17일, 오후 12시 (정오)**

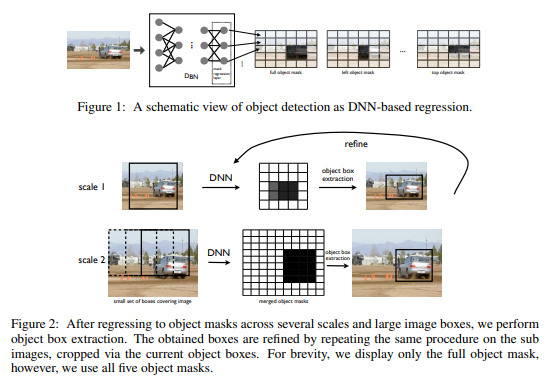
**학번 : 20171678**

**반 번호 : 2반**

**이름 :이찬희**

**1. 영상 처리 분야에서 Deep Neural Network (DNN) 기술을 사용하여 상용화된 SW 또는 제품의 사례가 있는지 조사하고, 있다면 대표적인 한 가지에 사용된 DNN 기술에 대하여 구조도, 기능, 특징을 포함하여 한 페이지 이내로 설명하시오. 만약 상용화 단계의 사례가 없다면, 연구 개발 등에 많이 활용되는 대표적인 영상 처리 DNN 기술에 대하여 같은 방식으로 설명하시오.**

영상 처리 분야에서 DNN을 사용하여 상용화된 제품 중 하나로는 자동차에 탑재되는 자율주행 기능이 있다. 자율 주행이란 사람의 조작 없이 자동차 스스로 운행하는 기능을 말하는데, 이 기능에서 핵심적인 부분 중 하나는 도로 위의 사물을 감지하는 것이다.



위의 구조도에서는 DNN을 이용하여 물체를 인식한 후 이 물체의 bounding box를 여러 스케일에서의 regression을 통해 추출하는 것을 보여주고 있다. Figure 1에서 보이는 것처럼, 전체 오브젝트의 마스트를 Regression 모델을 사용하여 생성할 수 있다. 한 번의 DNN Regression은 이미지 속 여러 물체의 마스크를 반환하고, 각 마스크의 위치를 세밀하게 조장하기 위해 DNN localizer를 sub-window의 작은 집합들에 적용한다.

**2. 아래의 (CODE 1)은 다음과 같은 C/C++ 코드를 어떤 이전 버전의 Visual Studio의 DEBUG 모드에서 컴파일하여 생성한 어셈블러 코드의 일부이다.**

switch (val) {

case 0: tmp = val\*misc; break;

case 1: tmp = val + misc; break;

case 2: tmp = val/misc; break;

**/\* here \*/**

}

**위의 C/C++ 코드의 /\* here \*/ 부분에 다음과 같은 문장을 추가한 후 동일 조건에서 컴파일하였을 때, (CODE 2)와 같은 결과를 얻었다.**

case 3: tmp = val - misc; break;

**수행 속도 차원에서 어떠한 이유로 (CODE 2)의 계산 방식이 (CODE 1)의 방식보다 우월할 수 있는지, 각 방법의 작동/코드 수행 원리를 반드시 설명하면서 상세히 기술하라.**

|  |  |
| --- | --- |
| 그림입니다. 원본 그림의 이름: CLP000008f4731e.bmp 원본 그림의 크기: 가로 473pixel, 세로 733pixel | 그림입니다. 원본 그림의 이름: CLP000008f40001.bmp 원본 그림의 크기: 가로 485pixel, 세로 858pixel |

Code 2는 비교 연산량의 횟수를 줄여 Code 1의 방식보다 우월할 수 있다. Code 1에서는 val 값을 0, 1, 2와 차례대로 비교하여 값이 일치하는 경우 각 케이스에 해당하는 연산을 수행한다. 반면, Code 2에서는 점프테이블을 이용하여 val 값을 3과 비교하여 값이 3보다 작은 경우 필요한 연산을 수행한다.

예를 들어 val 값에 case에 정의되지 않은 큰 값이 들어온다고 가정해보면 Case 1에서는 비교연산을 0, 1, 2에 대해 세 번 수행하지만, Case 2에서는 비교연산을 3에 대해서만 수행하므로 더 적은 횟수의 연산을 수행하여 수행속도가 더 짧다.

**3. CUDA memory hierarchy에 대하여 아래한글 기준 1500글자(공백 제외) 이상의 분량으로 명료하게 기술하라. 강의자료의 내용을 copy-and-paste할 경우 답으로 인정하지 않으며, 반드시 자신의 언어로 CUDA에 대하여 잘 모르는 컴퓨터공학 전공 3학년에게 설명하는 수준에서 기술할 것. (제출 내용에 대하여 copy-check를 진행할 예정임)**

동일한 계산을 서로 다른 데이터에 대하여 반복한다고 할 때, 이 연산을 순차적으로 진행하는 것이 아니라 여러 개의 프로세서에서 병렬적으로 수행시킨다면 계산시간을 크게 줄일 수 있다. GPU는 Streaming Multiprocessor들로 구성되어 있는데, 각 SM은 Processing Block들로, 또 각 Processing Block들은 CUDA 코어들로 나누어져 있고, 이 CUDA 코어가 앞서 말한 연산을 수행하는 하나의 프로세서 역할을 한다.

여러 개의 프로세서에 연산을 병렬적으로 수행시키기 위해 프로그래머는 하나의 쓰레드에서 수행되는 프로그램을 작성해야 하는데, 이를 커널 프로그램이라 한다. 그 다음 태스크를 동일한 크기의 쓰레드 블록으로 나누어 주면, 각 블럭이 GPU의 SM에 매핑되고, 한 블럭 내의 쓰레드들은 32개의 쓰레드가 모인 와프 단위로 나누어져 한 와프 내에서는 커널 프로그램이 동시에 수행된다.

이렇게 작성된 커널 프로그램은 독립적으로 수행되는 것은 아니고, C 프로그램과 혼재하여 수행된다. C 프로그램은 CPU, 커널 프로그램은 GPU에서 수행되고 CUDA에서는 CPU의 프로세서를 host, GPU의 프로세서를 device라고 지칭한다. Host 와 device는 각각 자신만의 host memory와 device memory를 가지고 있고, 프로그램이 수행되기 위해서는 둘 사이에 데이터 전송 작업이 필수적이다. 이는 host application에서 cudaMalloc 등 함수를 이용하여 device memory에 메모리를 할당하고 필요한 값을 복사하고 cudaMemcpy 등 함수를 이용하여 device memory에서 host memory로, 또 host memory에서 device memory로 메모리를 복사해 오는 과정을 통해 이루어진다.

CUDA 커널 프로그램이 수행되는 쓰레드에서는 Register, Local memory, Shared memory, Global memory, Constant memory, Texture memory의 공간에 접근할 수 있다. Register와 Local memory는 범위가 레지스터 공간이 할당되는 스레드로 작고, 수명도 한 스레드로 짧지만 접근 속도가 매우 빠르다. Shared memory는 범위와 수명이 한 블록이고, 지연은 수십 사이클정도 발생한다. Global memory 와 Constant memory, Texture memory의 범위는 해당 호스트 어플리케이션의 모든 쓰레드, 수명은 해당 호스트 어플리케이션으로 동일하다. Global memory의 지연은 수백사이클 정도로 Shared memory나 Register, Local memory와 비교하면 접근속도가 매우 느리다고 볼 수 있다. Constant memory는 캐시를 지원하여 한 와프 안에 있는 스레드들이 그 값을 공유하므로, 최초로 데이터를 가져올 때에는 Global memory와 비슷한 수준의 지연이 발생하지만 이후에 값에 다시 접근할 때에는 빠르게 접근할 수 있다. Texture memory 또한 캐시 읽기를 지원하여 한 와프 안의 쓰레드들이 비슷한 위치에 접근하고자 할 때 빠르게 접근할 수 있다. Global memory와 Constant memory, 그리고 Texture memory는 앞 문단에서 이야기 한 host와의 데이터를 주고받는 데에 사용된다.

앞서 말한 memory들 중 register와 shared memory는 on-chip memory이기 때문에 접근 속도가 빠르지만, 그 외의 off-chip memory들은 접근속도가 느리다. 특히, 캐시를 지원하는 Constant memory나 Texture memory 와는 달리 Global memory는 접근하는데 수백사이클 정도의 지연이 발생하기 때문에 최근의 GPU는 이 문제를 해결하기 위하여 L1캐쉬와 L2캐쉬를 지원한다.