붙임2

연구논문/작품 중간보고서

2018 학년도 제 2 학기

제목	가상화 GPU 성능 분석과 개선 방안 탐색	○ 논문(O) 작품( ) *해당란 체크
GitHub URL	https://github.com/sonmj426/graduation_thesis	
평가등급	지도교수 수정보완 사항	팀원 명단
A, B, F중 택1 (지도교수가 부여)	○ ○ ○ A	손민지 (인) (학번:2015318684)

2018 년 9 월 21 일

지도교수 : 한 환 수 서명

■ 요약

Docker는 컨테이너 기반의 오픈소스 가상화 플랫폼으로서 다양한 프로그램과 실행환경을 컨테이너로 추상화하고 동일한 인터페이스를 제공하여 프로그램의 배포 및 관리의 단순화를 가능하게 한다. 하나의 머신에서 여러 가상화 환경을 제공하는 기술의 수요가 증가함에 따라 운영체제 전체를 가상화해야 하는 가상머신에 비하여 운영체제의 일부만을 가상화하는 컨테이너는 가볍다는 장점이 있어 널리 쓰이고 있다. 최근 딥 러닝의 중요성이 높아지면서 GPU의 중요성 또한 갈수록 높아지고 있다. 따라서 본 논문에서는 시간 및 메모리 측면에서 NVIDIA Docker의 성능을 분석하여 이를 바탕으로 성능을 개선할 방안을 밤색한다. 앞선 실험을 통해 여러 개의 컨테이너를 실행할 시 성능 저하의 원인과 GPU 사용 패턴을 파악한 후 여러 컨테이너들이 GPU 자원을 효율적으로 사용할 수 있는 해결안을 제시해보려고 한다.

■ 서론

1. 제안배경 및 필요성

소프트웨어는 다양한 환경들이 존재하고 요구되기 때문에 그만큼 다양한 변수를 고려하여 개발되어야 한다. 특히 서버를 관리하는 것은 매우 복잡하고 섬세한 작업을 요구한다. 어떠한 서비스를 제공하기 위해서는 그에 적합한 서버가 필요한데 이 때, 데이터베이스, 서버, 웹 서버, 캐시 서버 등을 하나의 서버에서 관리한다면 설정이 복잡해지거나 애플리케이션이 거대해지거나 필요할 때 횡적인 확장을 하는 작업이 어려워진다. 이처럼 하나의 머신에서 여러 가상화 환경을 제공하는 기술의 수요가 점점 증가하고 있다. Docker는 컨테이너라는 격리된 공간에서 프로세스를 동작함으로써 서버 환경의 변화에 따른 설정과 관리 등에서의 어려움을 크게 줄여준다. 이는 뛰어난 이식성을 제공하며 환경에 구애받지 않고 애플리케이션을 신속하게 배포하고 서버를 효율적으로 운영할 수 있도록 한다. 원래 컨테이너란 많은 화물들을 수송하는 데 쓰는 규격화된 금속 용기로 옷, 전자제품, 음식 등 다양한 물품들을 담을 수 있지만 사실 이 안을 채울 때 이 컨테이너가 어느 배에 쌓일지 뭐와 같이 쌓일지 고려할 필요가 없는 것처럼 docker 컨테이너도 다양한 프로그램이나 실행환경을 추상화하고 동일한 인터페이스를 제공하여 여러 프로그램의 배포와 관리를 단순하게 해준다. “build, ship, and run

any app anywhere" 컨테이너를 잘 활용한다면 어떠한 앱도 어디서나 쉽게 빌드하고 실행할 수 있다.

컨테이너와 가상머신은 모두 애플리케이션과 그에 대한 의존성을 격리시켜 어디서든 실행 가능하게 한다는 목적을 갖고 있는데 가상머신은 하이퍼바이저를 이용하여 호스트 컴퓨터에서 여러 운영체제를 동시에 실행하고 관리하며 각 가상머신들에게 자원들을 분배한다. 가상머신은 가상화된 하드웨어 위에 운영체제가 올라가기 때문에 호스트와 거의 완벽하게 분리된다는 장점이 있지만 그 때문에 너무 무겁다는 단점 또한 있다. 그에 비해 docker는 게스트 운영체제를 두지 않고 호스트 운영체제 커널을 바로 사용한다. 운영체제의 커널 위의 사용자 공간 자원 제한을 통해 공간을 분리하고 필요한 자원만을 쓰도록 하기 때문에 하이퍼바이저와 달리 게스트 운영체제가 필요하지 않고 cpu allocation, storage, RAM 등이 적게 요구된다. 따라서 가상머신보다 훨씬 가볍고 가볍기 때문에 빠르고 하나의 머신에서 보다 더 많이 만들어질 수 있다. 또한 docker는 물리 환경과 가상화 환경에서 모두 호환이 가능하므로 이식성이 매우 우수하다. Docker 컨테이너로 개발되는 애플리케이션은 인프라 정보까지 포함되기 때문에 개발자들 입장에서 수많은 환경 변수들을 크게 신경쓰지 않아도 되기 때문에 개발과 배포 과정이 매우 간단하여 주목받고 있다. 기존의 컨테이너를 활용한 기술들에 비해 Docker는 다음과 같은 장점들을 지닌다. 우선 사용법이 간편하며 Docker hub를 통한 다양한 이미지 검색 및 사용이 용이하다. 적은 자원을 효율적으로 사용하여 가볍고 빠르며 애플리케이션의 기능들을 각각의 컨테이너로 쉽게 분리할 수 있기 때문에 뛰어난 모듈성과 확장성을 지닌다.

GPU는 그래픽 카드의 핵심 칩으로서 게임이나 영화 등의 그래픽 처리 과정에서 여러 픽셀들에 대한 단순한 연산들을 동시에 빠르게 수행하기 위해 즉, 병령 연산을 효율적으로 처리하기 위해 수천 개의 코어를 갖도록 설계된 보조-프로세서이다. 기존에는 단순히 CPU의 연산 결과를 화면으로 출력하는 부품 역할을 수행해왔다. 이후 NVIDIA가 1999년에 GeForce라는 이름의 새로운 그래픽 컨트롤러를 내놓으며 GPU라는 용어를 제창하였으며 GPU는 그의 주된 역할인 2D 및 3D 그래픽 연산 및 생성 이외의 범용 작업을 수행하게 되었다. 즉, 본래 빠른 그래픽 처리를 위해 설계된 보조-프로세서 GPU는 강력한 병렬 연산 능력을 갖추고 있어 CPU만으로도 대부분의 작업을 처리할 수 있었던 예전과 달리 다양한 산업과 연구 분야의 general-purpose high-preformance computing에 사용되고 있다. 또한 최근 딥 러닝 분야의 부상으로 그 중요성이 강조되고 있다. 패턴 인식, 자연어

처리 등의 작업들을 수행하기 위하여 딥 러닝의 중요성이 높아지고 있다. 딥 러닝 모델은 많은 트레이닝 데이터를 통해 빠르게 모델을 트레이닝할 수 있어야 하는데 GPU를 통해 수많은 데이터 훈련의 반복 처리에 걸리는 시간을 몇 주에서 몇 일로 대폭 줄일 수 있다. 일련의 명령어를 순차적으로 하나씩 처리하는 CPU에 비해 GPU는 여러 명령어들을 동시에 처리하는 병렬 처리 방식에 특화된 구조를 지닌다. 따라서 수많은 연산을 한꺼번에 수행할 수 있는 GPU가 인공지능, 자율주행, 빅데이터 등의 분야에서 주목받고 있다. 자율 주행이 가능하려면 여러 센서, 카메라, 레이더 등으로부터 전달받은 수많은 정보들을 바탕으로 앞의 장애물이 무엇인지 판단하거나 주행 방법을 결정해야 하는데 수많은 변수들을 빠르게 파악하고 처리하는데 GPU가 필수적인 역할을 한다.

2. 연구논문/작품의 목표

본 논문의 최종 목표는 컨테이너 간 GPU 자원의 공평한 사용을 위한 해결안을 제시하는 것이다. NVIDIA Docker는 단순하게 GPU 전체를 하나의 컨테이너에게 할당한다. GPU를 가상화하여 공유하려는 기존의 다양한 시도들은 모두 성공하지 못하고 성능 저하를 보였다. 그에 비해 NVIDIA Docker가 제시한 방식은 성능 저하가 거의 없으며 할당된 GPU를 자유롭게 사용할 수 있다. 하지만 다른 컨테이너에서 같은 GPU에 동시에 접근할 시 치명적인 문제가 발생한다. NVIDIA Docker는 GPU를 할당만 할 뿐 그 후에 각 컨테이너가 GPU를 어떻게 사용하고 있는지에 대해서는 관리하지 않는다. 따라서 여러 컨테이너가 같은 GPU에 동시에 접근하거나 메모리 할당이 동시에 발생하는 경우가 생길 수 있다. 본 논문을 통해 여러 컨테이너들이 GPU 자원을 효율적으로 사용할 수 있는 해결안을 제시해보려고 한다.

그에 앞서 NVIDIA Docker의 성능을 분석해보는 실험들을 진행하였다. Docker 컨테이너 내에서 NVIDIA GPU를 이용할 수 있도록 해주는 NVIDIA Docker의 성능을 시간 및 메모리 측면에서 분석한다. 결과 분석을 통해 NVIDIA Docker의 성능 개선안으로 통합 메모리 (unified memory)와 연산 선점 (compute preemption) 기능이 제공되는 최신 GPU 사용을 제안한다. 이를 통해 메모리 접근의 오버헤드를 줄이고 폭넓은 메모리 사용이 가능하기를 기대한다.

3. 연구논문/작품 전체 Overview

일반적으로 서버를 관리하는 것은 매우 복잡하고 섬세한 작업을 요구한다. 어떠한 서비스를 제공하기 위해서는 그에 적합한 서버가 필요한데 이 때, 데이터베이스 서버, 웹 서버, 캐시 서버 등을 하나의 서버에서 관리한다면 설정이 복잡해지거나 애플리케이션이 거대해지거나 필요할 때 횡적인 확장을 하는 작업이 어려워진다. 이처럼 하나의 머신에서 여러 가상화 환경을 제공하는 기술의 수요가 점점 증가하고 있다. Docker는 컨테이너 기반의 오픈소스 가상화 플랫폼으로서 서버 환경의 변화에 따른 설정과 관리 등에서의 어려움을 줄여준다.

본래 빠른 그래픽 처리를 위해 고안된 GPU는 강력한 병렬 연산 능력으로 인해 최근에는 다양한 산업과 연구 분야의 general-purpose high-performance computing에 사용된다. 최근 패턴 인식과 자연어 처리 등을 수행하기 위하여 딥러닝의 중요성이 높아지고 있다. 딥 러닝 모델은 많은 트레이닝 데이터를 통해 빠르게 모델을 트레이닝할 수 있어야 하는데 GPU를 통해 수많은 데이터 훈련의 반복 처리에 걸리는 시간을 몇 주에서 몇 일로 대폭 줄일 수 있다.

Docker는 컨테이너 기반의 오픈소스 가상화 플랫폼이다. 컨테이너는 격리된 공간에서 프로세스를 동작하는 기술로서 docker는 다양한 프로그램과 실행환경을 컨테이너로 추상화하고 동일한 인터페이스를 제공하여 프로그램의 배포 및 관리의 단순화를 가능하게 한다. OS(Operating System) 전체를 가상화하는 VM(Virtual Machine)에 비하여 OS의 일부만을 가상화하는 컨테이너는 가볍다는 장점을 갖기에 널리 쓰이고 있다.

GPU는 병렬 연산를 효율적으로 처리하기 위해 수천 개의 코어를 갖는다. 따라서 애플리케이션의 연산집약적인 부분을 GPU로 넘기고 나머지 코드를 CPU에서 처리하는 GPU 가속 컴퓨팅은 해당 애플리케이션을 CPU만으로 처리할 때와 비교하여 엄청난 성능 향상을 보인다. 그에 따라 GPU의 중요성이 갈수록 높아지고 있다. 이처럼 GPU를 그래픽처리가 아닌 일반 연산에 사용하기 위한 프로그래밍 모델과 API를 제공하는 병렬 연산 플랫폼 중 하나가 CUDA이다.

NVIDIA Docker에서의 GPU 사용을 효율적인 방향으로 개선하는 것이 주요 목표이다. 하나의 컨테이너에서 하나의 애플리케이션을 실행할 시에는 실제 디바이스에서 실행할 때의 성능에서 크게 저하되지 않는다. 하지만 여러 개의 컨테이너에서 각각 하나 또는 그 이상의 애플리케이션을 실행할 시 성능 저하의 원인과 GPU 사용 패턴을 파악하여 최종적으

로 여러 컨테이너들이 GPU 자원을 효율적으로 사용할 수 있도록 하는 해결안을 제시하고자 한다.

4. 섹션 소개

이후 내용은 다음과 같이 구성된다. 우선 관련 연구 섹션에서 본 연구와 관련된 논문들을 소개한다. 제안 작품 소개 섹션에서 Docker, NVIDIA Docker와 GPU CUDA 실행모델 및 메모리 모델에 대한 이론적 배경을 제공하고 실험이 진행된 시스템 구성에 대하여 설명한다. 또한 2개의 실험이 실제로 구현된 방식에 대하여 구체적으로 서술한다. 다음으로 구현 및 결과 분석 섹션에서 각 실험 결과, 그에 대한 분석과 개선 방안을 소개한다. 결론 및 소감 섹션에서 연구를 진행하며 경험하고 느낄 수 있었던 점들을 서술하고 참고 문헌 섹션으로 보고서를 마무리한다.

■ 관련연구

1. D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal,* vol. 2014, no. 239, p. 2, 2014.

여러 논문에서 가상 머신 환경과 컨테이너 환경에 대하여 다양한 항목으로 성능평가를 진행 하였으며, 대부분의 논문들이 Docker의 우수성을 평가 하였다. 하지만 대부분의 비교 방식이 호스트 기반 가상화 기술이거나 Bare-Metal 방식 기술이긴 하지만 상대적으로 시장 점유율이 낮은 KVM (Kernel based Virtual Machine)정도와의 비교였다. 따라서 해당 논문에서는 기존 논문들과는 차별화 되게 현재 전 세계적으로 많이 사용하고 있는 상용제품인 VMware vSphere(Bare-Metal 기반 가상화 기술)과 비교 분석을 진행하였다. 평가 항목은 Object(가상 머신, Container) 생성 속도, 부팅 속도, 재부팅 속도와 같이 운영에 있어서 중요한 작업 속도에 대한 평가, 그리고 CPU, Memory, Disk 점유율, 그리고 가장 중요한 Disk I/O성능 순이다. [1]

2. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H., Skadron, K., Rodinia: A benchmark suite for heterogeneous computing. In proceedings of the IEEE International Symposium on Workload Characterization, 2009.

해당 논문은 heterogeneous computing (이기종 컴퓨팅)을 위한 벤치마크

인 Rodinia를 소개하고 그 특징을 설명한다. Rodinia는 GPU와 같이 새로 출현하는 플랫폼들을 연구하는 데 도움이 되도록 멀티 코어 CPU 및 GPU 플랫폼을 대상으로 하는 애플리케이션과 커널들을 포함한다. Rodinia 벤치마크는 병렬 통신 패턴, 동기화 기술 및 전력 소비에 대하여 폭넓게 다루고 있다.

3. D. Kang, T. J. Jun, D. Kim, J. Kim, and D. Kim, "ConVGPU: GPU Management Middleware in Container Based Virtualized Environment", 2017 IEEE International Conference on Cluster Computing, 2017.

CPU에 비해 병렬 연산에서 압도적인 성능을 보이는 GPU가 최근 범용 고성능 컴퓨팅 분야에서는 필수적이다. 그러한 GPU를 가상화된 환경에서 효율적으로 사용하려는 많은 시도들이 있어왔다. 특히 NVIDIA Docker는 GPU를 컨테이너 기반 가상화 환경에서 사용할 수 있는 가장 실용적인 방법을 제안하였다. 그러나 대부분의 시도들은 GPU를 여러 컨테이너 간에서 공유하는 경우는 고려하지 않았다. 이를 고려하지 않는다면 최악의 경우에는 프로그램이 실패하거나 deadlock 상태가 발생할 수 있다. 해당 논문은 여러 컨테이너 간에서 GPU를 공유하는 해결안인 ConVGPU를 제안한다. ConVGPU를 사용한다면 시스템은 각 컨테이너가 실행되는데 요구되는 GPU 메모리를 보장할 수 있다.[3] 이를 달성하기 위해 컨테이너들에 의해 쓰일 GPU 메모리를 관리하는 네 가지 스케줄링 알고리즘을 소개한다. 이러한 알고리즘들은 시스템이 실행 도중 deadlock 상태에 빠지는 상황을 방지할 수 있다.[3]

■ 제안 작품 소개

1. 배경 지식

1.1. Docker

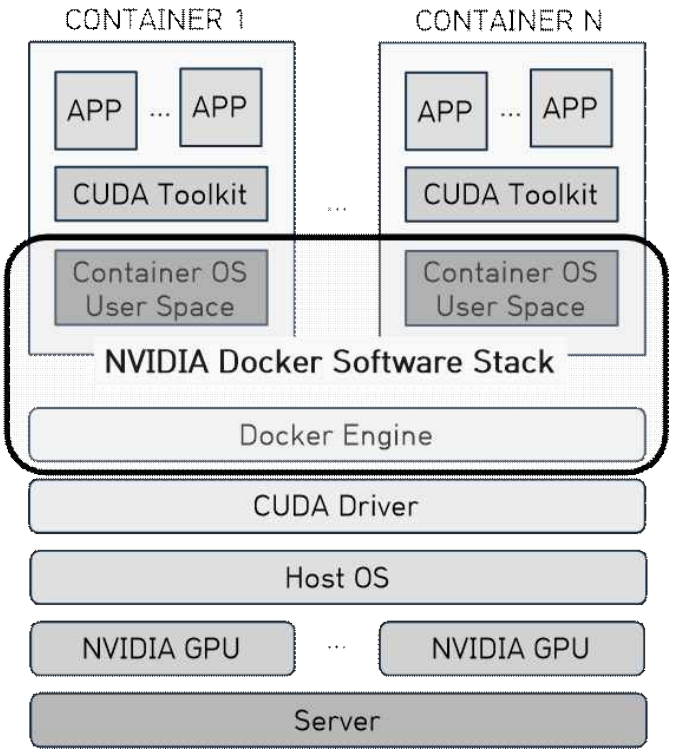
Docker는 namespaces, control groups와 같은 리눅스 커널 기능을 이용한 리눅스 컨테이너를 기반으로 시작한 오픈소스 가상화 플랫폼이다. 컨테이너라는 격리된 공간을 제공하고 이들이 서로 충돌하지 않도록 하기 위해 namespaces 기능을 사용하며 이는 pid, mnt, net 등의 네이스페이스를 지원한다. Control groups는 CPU, 메모리, 네트워크 등의 자원에 대한 제어를 가능하게 해준다. 즉, 각각의 컨테이너가 필요한 자원만을 사용하도록 자원을 제어하고 제한한다.

컨테이너는 code, runtime, system tools, system libraries 등 어떠

한 애플리케이션이 실행 환경과 관계없이 동일하게 실행되는데 필요한 모든 요소를 포함한다. 가상 머신과는 다르게 컨테이너는 유저 공간을 추상화하고 호스트 시스템의 커널을 다른 컨테이너들과 공유하기 때문에 매우 가볍고 빠르며 이식성이 높다는 장점이 있다.

Docker는 이미지로부터 생성되는데 이미지란 컨테이너의 실행에 필요한 프로그램, 라이브러리, 소스 코드, 설정값 등을 모두 포함하고 있는 read-only, immutable한 파일이다. 즉, 컨테이너는 이미지를 실행한 상태라고 볼 수 있고 추가되거나 변하는 값은 이미지가 아닌 컨테이너에 저장된다. 같은 이미지에서 여러 개의 컨테이너를 생성할 수 있고 컨테이너의 상태가 바뀌거나 컨테이너가 삭제되더라도 이미지는 변하지 않고 그대로 남아있다. Docker 이미지는 docker hub에 등록하거나 docker registry를 직접 만들어 관리할 수 있다.

1.2. NVIDIA Docker



[그림] NVIDIA Docker 구조

Docker 컨테이너는 어떠한 운영 체제나 프로세서의 조합인지에 대한 아무런 지식이 없더라도 상관없이 기능을 수행할 수 있음

(platform-agnostic)은 물론 어떠한 하드웨어에도 종속적이지 않다(hardware-agnostic). 따라서 Docker는 NVIDIA GPU와 같이 특수한 하드웨어를 기본적으로 지원하지 않는다.

NVIDIA Docker는 NVIDIA GPU를 활용하는 Docker 이미지 이식을 가능하게 하기 위해 개발되었으며 GPU 기반 컨테이너 이식을 위해 필요한 중요한 요소 다음의 2가지를 제공한다. 첫 번째는 드라이버 종류에 상관없이 실행되는 CUDA 이미지이다. CUDA는 GPU를 그래픽처리가 아닌 범용 목적의 연산에 사용하기 위한 프로그래밍 모델과 API를 제공하는 병렬 연산 플랫폼이다. 두 번째는 Docker 상위에서 일부 명령들을 포착하여 기존의 Docker 명령이 아닌 nvidia-docker 명령이 실행되도록 옵션을 추가해주는 래퍼(wrapper)이다.

1.3. GPU CUDA 실행모델 및 메모리 모델

GPU는 긴 메모리 지연시간을 숨기기 위해 수천에서 수만개의 쓰레드를 동시에 동일한 명령어를 수행하며, 이를 SIMT (Single Instruction Multiple Thread) 모델이라고 한다. CUDA에서는 일반적으로 32개의 쓰레드 그룹을 워프(warp)라고 부르고, 다수개의 워프 그룹을 쓰레드 블록(thread block)이라고 한다. 각 쓰레드 블록은 GPU 내부의 멀티프로세서에 할당되며, 각 멀티프로세서 내부에서는 할당 받은 쓰레드 블록을 다시 워프 단위로 구분하여 명령어를 수행한다.

GPU는 온/오프-칩 메모리로 구분된다. 온-칩 메모리에는 레지스터 파일, 다중-레벨 캐시, 그리고 공유 메모리가 있다. 오프-칩 메모리에는 전역 메모리, 상수 메모리, 지역 메모리, 그리고 텍스처 메모리가 있다. 상수 메모리와 텍스처 메모리는 읽기 전용이며, 전역 메모리에 대해서는 읽기/쓰기가 가능하다. 지역 메모리는 레지스터 스필(register spill)이 발생하였을 때 사용하는 메모리 공간이다. 오프-칩 메모리는 일반적으로 온-칩 메모리 대비 최대 100배 정도의 긴 접근 시간을 필요로 한다. CUDA 프로그래밍에서는 기본적으로 커널 실행 전에 CPU 메모리의 데이터를 GPU의 전역 메모리로 복사하고 사용하며, 필요에 따라 통합 메모리(unified memory) 기능을 이용하여 CPU와 GPU 간에 메모리 복사 없이 커널 수행중에 필요한 데이터를 CPU 메모리에 직접 접근하여 사용하는 방법도 제공하고 있다.

2. 실험 환경 및 벤치마크

본 실험이 진행된 시스템은 Intel Core i7-6700K 프로세서와 NVIDIA GTX980Ti (Maxwell 아키텍처) GPU 1대로 구성되어 있으며 Ubuntu 16.04.1에서 수행하였다. GPU는 드라이버 버전 387.26과 CUDA 버전 8.0.61을 사용하였고, 메모리는 총 6GB가 탑재되어 있다.

본 실험은 Rodinia 3.1 벤치마크 애플리케이션을 대상으로 NVIDIA Docker version 2.0.2 성능을 측정하였다. Rodinia는 GPU 성능 측정을 위해 많이 쓰이는 실제 애플리케이션들로 구성되어있다. 벤치마크로서 병렬 통신 형태, 동기화 기법과 동력 소비량에 대하여 광범위하게 다룬다.

3. 실제 구현

3.1. NVIDIA Docker의 소프트웨어 스택 오버헤드 분석

본 실험에서는 Rodinia 벤치마크 애플리케이션 19개를 대상으로 하여 Docker 사용 유무에 따른 실행 시간 차이를 분석한다. 19개의 컨테이너를 동시에 생성하여 애플리케이션을 실행하기 위하여 스크립트를 작성하였다. Docker를 사용한 경우에는 외부에서 컨테이너 안의 명령을 실행하는 docker exec을 사용하여 애플리케이션을 실행하였으며 실험 결과 값은 각 애플리케이션을 Docker를 사용할 경우와 사용하지 않을 경우 각각 5회 실행하고 그 평균치를 내었다.

3.2. GPU의 제한된 디바이스 메모리 크기에 따른 한계

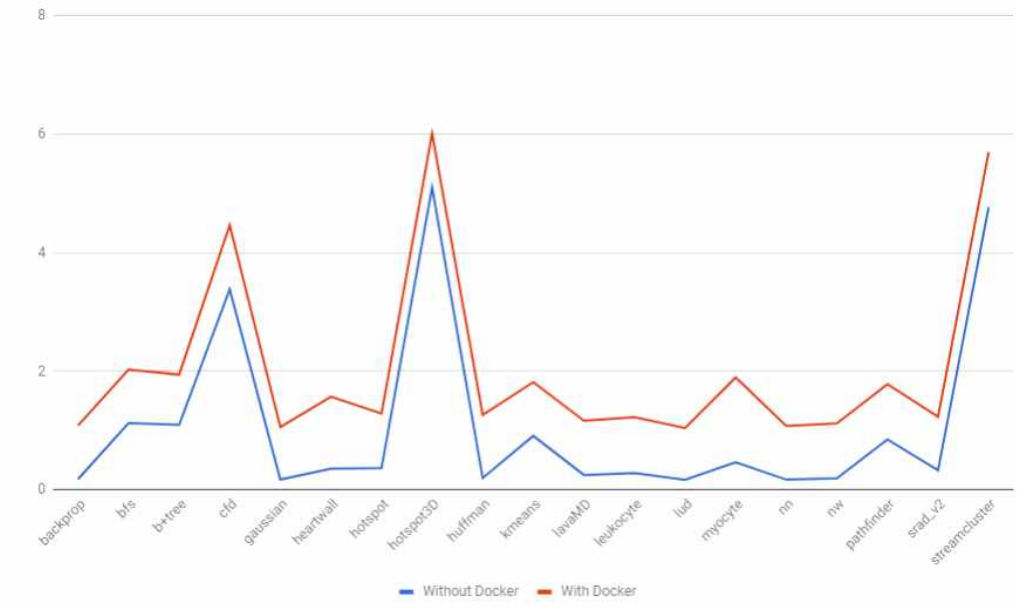
본 실험에서는 Docker 컨테이너 개수를 점차 늘려가며 실행하여 메모리 크기에 따른 한계를 알아본다. 실험은 Rodinia 벤치마크 중 커널 시간이 비교적 긴 hotspot3D 애플리케이션을 중점으로 진행하였다. 하나의 hotspot3D 애플리케이션은 98MB만큼의 메모리를 사용한다.

■ 구현 및 결과분석

1. NVIDIA Docker의 소프트웨어 스택 오버헤드 분석

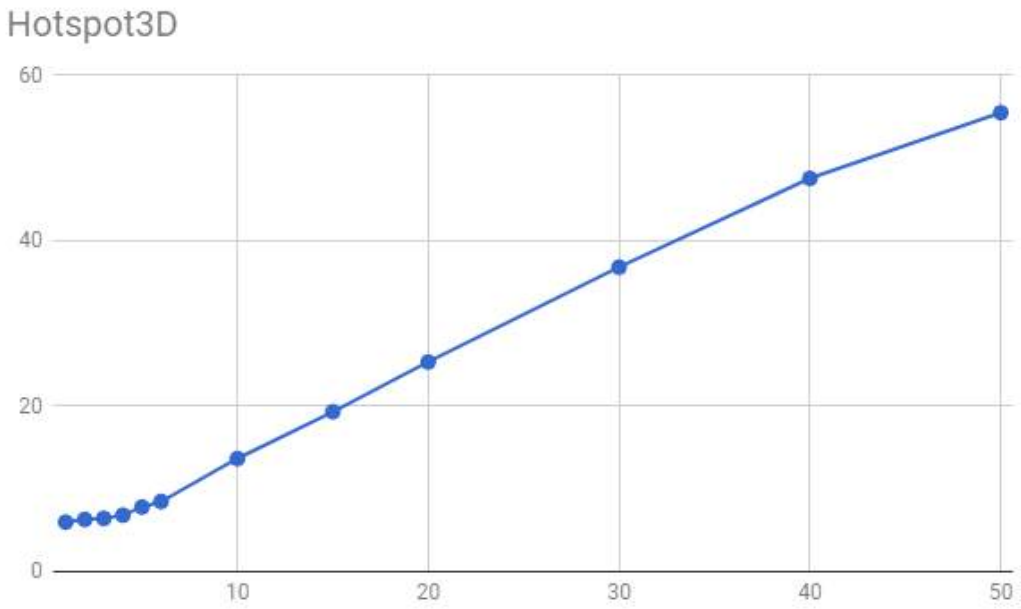
	Baremetal	Docker
backprop	0.177	1.083
bfs	1.1222	2.027
b+tree	1.0942	1.9426
cfd	3.3854	4.4678
gaussian	0.1694	1.0568
heartwall	0.3552	1.5696
hotspot	0.3626	1.2836
hotspot3D	5.1054	6.0152
huffman	0.1974	1.2638
kmeans	0.909	1.8178
lavaMD	0.248	1.1644
leukocyte	0.2786	1.2226
lud	0.1666	1.0412
myocyte	0.4648	1.8956
nn	0.1696	1.0762
nw	0.19	1.119
pathfinder	0.849	1.783
srad_v2	0.3272	1.234
streamcluster	4.769	5.7008

Rodinia 벤치마크 애플리케이션 19개를 대상으로 하여 Docker 사용 유무에 따른 실행 시간 차이를 분석한 결과, Docker를 사용할 경우에 Docker를 사용하지 않은 경우보다 실행 시간이 평균 0.96초 정도의 증가를 보였다. 이러한 차이는 애플리케이션의 종류나 실행 시간에 영향을 받는 것이 아니라 NVIDIA Docker 소프트웨어 스택의 추가로 인해 발생하는 오버헤드이다.



2. GPU의 제한된 디바이스 메모리 크기에 따른 한계

Docker 컨테이너 개수를 점차 늘려가며 실행하여 메모리 크기에 따른 한계를 알아본다. 실험은 Rodinia 벤치마크 중 커널 시간이 비교적 긴 hotspot3D 애플리케이션을 중점으로 진행하였다. 하나의 hotspot3D 애플리케이션은 98MB만큼의 메모리를 사용한다. 해당 애플리케이션의 경우, 52개의 컨테이너를 동시 운용하여 총 GPU 메모리의 약 93%를 사용할 때가지는 모든 애플리케이션이 원활하게 실행되었으나, 이를 초과할 시 오류가 나고 대부분의 애플리케이션이 실행에 실패하였다.



■ 결론 및 소감

신속한 배포와 효율적인 서버 운영을 위해서는 Docker가 필요하며 딥 러닝의 부상으로 GPU의 중요성이 부각되고 있다. 본 논문에서는 Docker와 GPU를 모두 활용하는 NVIDIA Docker에 대한 성능 분석을 수행하였다. 총 2가지의 실험을 통해 1) NVIDIA Docker의 소프트웨어 스택 오버헤드 수준을 확인하였고 2) 하나의 GPU 디바이스를 대상으로 다중 컨테이너가 공유하여 수행될 때 발생하는 GPU 디바이스 메모리 오류를 분석 및 확인하였다.

본 연구를 진행하면서 논문에 삽입할 그림을 만드는 법, 참고 문헌을 작성하는 법 등 기본적으로 논문을 어떤 식으로 작성하는지에 대하여 알 수 있었다. 처음에는 본 주제에 대하여 단독으로 연구를 진행하게 된 상황이고 거의 아무것도 모르는 상태라서 당황스럽고 진행하는 과정에서도 어려움이 많았지만 그만큼 열의를 갖고 연구를 하니 배운 점도 얻은 점도 보다 많은 것 같다.

Pascal GPU 아키텍처는 기존 Maxwell과 Kepler GPU 아키텍처에서 제공하는 쓰레드 블록 단위에서의 연산 선점 대신 명령어 단위에서의 연산 선점이 가능하다. 이러한 연산 선점은 장시간 실행되는 애플리케이션이 시스템을 독점하거나 제한 시간을 초과하는 상황을 방지하며 효율적인 스케줄링을 가능하게 한다. 향후 이러한 이점을 활용하여 여러 컨테이너에서 여러 GPU를 공유하는 환경에서 발생할 수 있는 공유 자원 경합 문제를 분석하고 개선할 계획이다.

■ 참고문헌

[1] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal,* vol. 2014, no. 239, p. 2, 2014.

[2] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Lee, S.H., Skadron, K., Rodinia: A benchmark suite for heterogeneous computing. In proceedings of the IEEE International Symposium on Workload Characterization, 2009.

[3] D. Kang, T. J. Jun, D. Kim, J. Kim, and D. Kim, "ConVGPU: GPU Management Middleware in Container Based Virtualized Environment", 2017 IEEE International Conference on Cluster Computing, 2017.