Multi-GPU 시스템에서 성능 개선을 위한 작업 스케줄링 기법 연구 동향

엄시우*, 김현준*, 송민호*, 서화정**

*한성대학교 (대학원생)
**한성대학교 (부교수)

Research Trends in Task Scheduling Techniques for Improving Performance on Multi-GPU Systems

Si-Woo Eum*, Hyun-Jun Kim*, Min-Ho Song*, Hwa-Jeong Seo**

*Hansung University(Graduate student)
**Hansung University(Associate Professor)

요 약

본 논문에서는 GPU와 Multi-GPU 기술에 대해서 설명하고, Multi-GPU 시스템에서 발생하는 문제들과 이를 해결하기 위한 스케줄링 기법에 대해서 살펴본다. Multi-GPU 기술을 활용한 대용량 데이터 처리나 머신 러닝 작업 등에서는 스케줄링 기법에 따라서 작업 성능에 차이가 발생할 수 있다. 이러한 작업에서 최적의 성능을 위해 작업을 효율적으로 분배하는 스케줄링 기법에 대해서 연구한 논문을 조사하고 분석하여 스케줄링 기법 연구의 동향을 살펴본다.

I. 서론

GPU는 병렬 처리에 대한 높은 성능을 제공하여 대용량 데이터의 처리 속도를 대폭 향상시킬 수 있다. 특히 여러 개의 GPU를 하나의 시스템에서 사용하는 Multi-GPU 시스템을 활용하면 GPU의 성능을 더욱 확장시켜 대용량 데이터의 처리와 분석을 보다 빠르고 효율적으로 수행할 수 있다. 그러나 Multi-GPU를 활용할 때 고려해야할 여러 문제들이 존재한다. 대표적으로 GPU 간 데이터 전송, 동기화 그리고 각 GPU에 할당되는 작업 부하의 균형을 맞춰주는 것 등이 있다.

이러한 문제를 해결하여 Multi-GPU 시스템에서 최적의 성능을 얻기 위한 스케줄링 기법에 대해 연구가 진행되고 있다. 본 논문에서는 Multi-GPU 시스템에서 작업을 효율적으로 분배하여 성능을 개선하는 스케줄링 기법에 대한 연구 동향을 살펴본다.

II. 관련 연구

2.1 Graphics Processing Unit(GPU)

GPU는 그래픽 처리 장치의 약어로 컴퓨터 그래픽스 및 병렬 처리, 3D 모델링을 위해 설계된 전용 프로세서로 개발되었다.

개발 목적에 맞춰서 3D, 2D 그래픽 연산과 생성의 역할을 담당하였으나 GPU가 가지고 있는 뛰어난 병렬 연산 처리 능력을 활용하기 위해서 CPU가 담당하고 있던 작업을 GPU를 활용하기 시작하였다. 이처럼 기존의 GPU의 역할에서 벗어나 이외의 범용적인 작업을 하는 것을 General Purpose Computing on Graphics Processing Units(GPGPU)라고 하며 머신 러닝 학습, 기상 변화 예측 그리고 암호 크래킹 등의 분야에서 활용되고 있다[1, 2].

2.2 Multi-GPU

Multi-GPU는 하나의 시스템에서 여러 개의

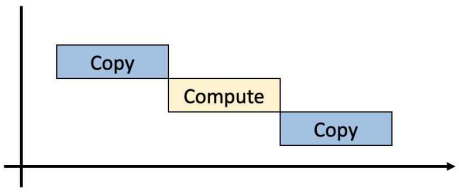
GPU를 사용하는 기술이다. 높은 병렬 연산 처리 능력을 가지고 있는 GPU를 여러 개 사용하여 더 높은 성능과 더 큰 계산 능력을 얻을 수 있다. 이러한 Multi-GPU 기술은 대용량 데이터를 다루는 머신 러닝 작업이나 대규모 시뮬레이션 등과 같은 작업에서 많이 사용되고 있다[3].

2.3 GPU, Multi-GPU의 동작 과정

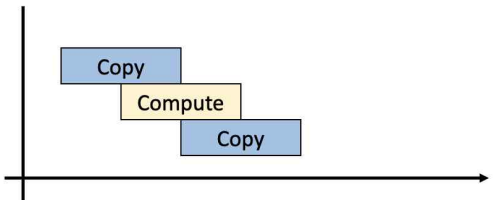
기본적인 GPU의 동작 과정은 (그림 1)과 같다. GPU에서 연산을 처리하기 위해 Host(CPU) 메모리에서 Device(GPU) 메모리로 데이터를 복사를 해야한다. 복사된 데이터를 가지고 작업을 수행한 다음 결과값을 다시 Host로 복사하는 과정으로 동작한다.

이때 Host에서 데이터 복사가 시작되는 부분부터 다시 Host로 데이터가 복사되는 부분까지를 보통 동작 시간으로 측정한다. 이를 최소화하기 위한 방법으로 가장 간단하게 생각할 수 있는 방법은 (그림 2)와 같이 메모리 전송과 연산을 오버랩 하는 방법이다. 이러한 방법은 CUDA Streams을 활용하여 구현 가능하다.

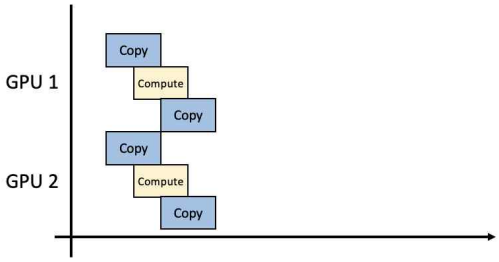
다음으로는 연산을 여러 개의 GPU를 활용하여 연산하는 방법이 있으며 이때 오버랩하는 방법까지 활용하게 되면 훨씬 높은 성능 향상을 얻을 수 있다. (그림 3)은 두 개의 GPU를 활용하는 Multi-GPU 시스템에서 오버랩을 하는 방법을 활용해 작업 시간을 최소화 할 수 있는 방법을 보여준다.



(그림 1) 기본적인 GPU 동작과정



(그림 2) 메모리 전송과 연산을 오버랩하는 방법



(그림 3) GPU 두 개를 활용한 Multi-GPU

다음 장에서는 이처럼 스케줄링을 통해 성능 개선을 하는 방법에 대한 연구를 살펴본다.

III. 성능 개선을 위한 스케줄링 기법 동향

3.1 Design and Analysis of Scheduling Strategies for Multi-GPU architectures

[4]에서는 Multi-CPU와 Multi-GPU 아키텍처를 위한 스케줄링 기법들을 분석하고 비교하는 연구를 진행하였다. 해당 논문에서는 Xkaapi 런타임 위에서 실험을 진행하였다. Xkaapi는 INRIA MOAIS 팀이 개발한 KAAPI 런타임의 새로운 구현으로 병렬 컴퓨팅을 위한 오픈 소스 소프트웨어 프레임워크이다. 동기화, 작업 스케줄링, 데이터 분할 및 메모리 관리와 같은 여러 기능을 제공하여 복잡한 병렬 응용 프로그램 개발의 편의성을 제공한다.

XKaapi 런타임 위에서 Work Stealing, Data-aware work stealing, Locality-aware work stealing 그리고 Heterogeneous Earliest-Finish-Time(HEFT) 이렇게 4가지의 스케줄링 전략을 설계하고 평가를 진행하였다.

Work Stealing은 작업을 분산 시키고, 유휴 스레드가 다른 스레드로부터 작업을 훔쳐오는 방식으로 병렬 처리를 수행한다. Data-aware work stealing은 Work Stealing의 변형으로, 메타 데이터 정보를 고려하여 Host와 Device간의 메모리 전송을 줄이는 것을 목표로 하는 방법이다. Locality-aware Work Stealing은 데이터 지역성을 고려하여 작업을 스케줄링하는 것을 목

표로 한다. 마지막으로 HEFT는 비용 모델에 기반하여 작업 완료 시간을 예측하여 가장 빨리 완료될 수 있는 작업부터 스케줄링하는 방법이다.

이러한 기법들을 12개의 CPU와 8개의 GPU로 구성된 이종 아키텍처에서 여러 벤치마크를 통해 분석하였다, 결과적으로 작업 주석이 데이터 지역성 전략과 함께 주어진 경우 Work Stealing이 효율적일 수 있다는 결론을 내렸으며, 또한 HEFT의 경우 매우 규칙적인 계산과 낮은 데이터 지역성을 가진 응용 프로그램에서 높은 성능을 보여준다는 실험 결과를 보여주었다.

3.2 Priority-Based PCIe Scheduling for Multi-Tenant Multi-GPU Systems

[5]에서는 다중 GPU 시스템이 데이터 센터에서 DNN 훈련과 같은 많은 계산을 요구하는 작업에서 속도 향상을 제공하기 위해서 많이 사용되지만, CPU와 다중 GPU 사이의 PCIe 대역폭이 제한되어 주요 성능 병목 현상이 되는 것을 분석하였으며, 또한 기존의 Round-Robin 기반의 PCIe 스케줄링 방법에 의존하면 많은 대역폭 경쟁이 발생하여 Multi-GPU의 실행이 정지될 수 있다는 것을 분석하였다.

이러한 대역폭 경쟁을 완화하기 위해서 다른 응용 프로그램의 데이터 전송과 GPU 실행을 겹치는 우선 순위 기반 스케줄링 방법을 제안하였다. 또한 Qos(Quality of Service) 요구사항을 충족하고 전체 Multi-GPU 시스템 처리량을 개선할 수 잇는 동적 우선 순위 방법도 제안하였다.

결과적으로 해당 논문에서 제안하는 우선 순위 기반 PCIe 스케줄링 방식을 사용하였을 때, Round-Robin 기반 스케줄링 대비 시스템 처리량이 평균 7.6% 향상되었으며, Semi-QoS 관리를 활용하여 QoS 목표를 충족시키면서 시스템 처리량을 유지할 수 있는 결과를 보여주었다.

3.3 Neon: A Multi-GPU Programming Model for Grid-based Computations

[6]은 그리드 기반 계산을 위한 새로운 프로그래밍 모델을 제안하고 있다. 단일 노드 다중 GPU 시스템의 장점을 쉽게 활용할 수 있도록 도와주는 프로그래밍 모델이다. Neon은 데이터 구조와 계산을 분리하고, 사용자 코드를 다양한 데이터 구조와 장치에 적용할 수 있도록 도움을 주며, 계산과 통신이 겹치는 최적화를 자동으로 수행한다.

Neon은 병렬 스켈레톤 프로그래밍 모델을 따른다. Map, Stencil, Reduce 세 가지의 구성 요소를 제공하며 이를 통해 다양한 계산을 표현할 수 있다. System, Set, Domain, Skeleton 네 가지의 추상화 계층으로 구성되어 있다. System은 아키텍처와 하드웨어 매커니즘을 숨기고, Set은 커널과 데이터 종속성 그래프를 정의하고, Domain은 그리드 데이터 구조를 제공하고 마지막으로 Skeleton은 의존성 그래프를 기반으로 커널과 동기화를 스케줄링한다.

C++ 라이브러리로 구현되었으며, CUDA, OpenMP, OpenCL 등 다양한 백엔드를 지원한다. 여러 그리드 기반 애플리케이션에 대해 8개의 GPU를 사용하는 시스템에서 효율적인 확장성을 보였다. Lattice Boltzmann 유체 해석, 유한 차분 포아송 방정식 해법, 유한 요소 선형 탄성 구조 해석 등의 애플리케이션을 간결하게 구현할 수 있으며, 이들 애플리케이션에서 99% 이상의 이상적 효율성을 달성했다.

IV. 결론

본 논문에서는 Multi-GPU 시스템에서 발생하는 문제점들을 해결하기 위해 여러 연구가 진행되고 있는 것을 확인하였다. 이 중에서 일부는 작업의 크기, 작업의 종류, GPU의 성능 등에 따라 다르게 적용되며, 최적의 성능을 위해서는 이러한 스케줄링 기법들을 조합하여 사용해야 한다. 이러한 연구 결과들은 대용량 데이터 처리나 머신 러닝 작업등 Multi-GPU 기술을 활용하는 연구나 산업 분야에서 유용하게 활용될 수 있을 것으로 기대된다.

V. Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2021-0-00540, Development of Fast Design and Implementation of Cryptographic Algorithms based on GPU/ASIC, 100%).

[참고문헌]

[1] An, SangWoo, et al. "Parallel implementations of ARX-based block ciphers on graphic processing units." Mathematics 8.11 (2020): 1894.

[2] Owens, John D., et al. "GPU computing." Proceedings of the IEEE 96.5 (2008): 879-899.

[3] Chan-Hee Choi, Alchan Kim, Kang-Wook Kim, Chang-Gun Lee. “Multiple GPU Scheduling for Real-time Systems.” 한국정보과학회 학술발표논문집. 1323-1325. 2015.

[4] Lima, Joao VF, et al. “Design and analysis of scheduling strategies for multi-CPU and multi-GPU architectures.” Parallel Computing, 44, 37-52. 2015.

[5] Li, Chen, et al. “Priority-based PCIe scheduling for multi-tenant multi-GPU systems.” IEEE Computer Architecture Letters 18.2. 157-160. 2019.

[6] MENEGHIN, Massimiliano, et al. Neon: A Multi-GPU Programming Model for Grid-based Computations. In: 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2022. p. 817-827.