KIPS Trans. Comp. and Comm. Sys.
Vol.11, No.5 pp.133~138
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2022.11.5.133

A Performance Study on CPU-GPU Data Transfers of Unified Memory Device

Oh-Kyoung Kwon† · Gibeom Gu††

ABSTRACT

Recently, as GPU performance has improved in HPC and artificial intelligence, its use is becoming more common, but GPU programming is still a big obstacle in terms of productivity. In particular, due to the difficulty of managing host memory and GPU memory separately, research is being actively conducted in terms of convenience and performance, and various CPU-GPU memory transfer programming methods are suggested. Meanwhile, recently many SoC (System on a Chip) products such as Apple M1 and NVIDIA Tegra that bundle CPU, GPU, and integrated memory into one large silicon package are emerging. In this study, data between CPU and GPU devices are used in such an integrated memory device and performance-related research is conducted during transmission. It shows different characteristics from the existing environment in which the host memory and GPU memory in the CPU are separated. Here, we want to compare performance by CPU-GPU data transmission method in NVIDIA SoC chips, which are integrated memory devices, and NVIDIA SMX-based V100 GPU devices. For the experimental workload for performance comparison, a two-dimensional matrix transposition example frequently used in HPC applications was used. We analyzed the following performance factors: the difference in GPU kernel performance according to the CPU-GPU memory transfer method for each GPU device, the transfer performance difference between page-locked memory and pageable memory, overall performance comparison, and performance comparison by workload size. Through this experiment, it was confirmed that the NVIDIA Xavier can maximize the benefits of integrated memory in the SoC chip by supporting I/O cache consistency.

Keywords : HPC, GPU, Unified Memory, Data Transfer

통합메모리 장치에서 CPU-GPU 데이터 전송성능 연구

권 오 경† · 구 기 범††

요 약

최근 고성능컴퓨팅, 인공지능 분야에서 GPU 장치 사용이 일반화되고 있지만, GPU 프로그래밍은 여전히 어렵게 여겨진다. 특히 호스트(host) 메모리와 GPU 메모리를 별도로 관리하기 때문에 성능과 편의성 방면에서 연구가 활발히 진행되고 있다. 이에 따라 여려가지 CPU-GPU 메모리 전송 방법들이 연구되고 있다. 한편 CPU와 GPU 및 통합메모리(Unified memory) 등 하나의 실리콘 패키지로 묶는 SoC(System on a Chip) 제품들이 최근에 많이 출시되고 있다. 본 연구는 이러한 통합메모리 장치에서 CPU, GPU 장치간 데이터를 사용하고 전송시 성능관련 비교를 하고자 한다. 기존 CPU내 호스트 메모리와 GPU 메모리가 분리된 환경과는 다른 특징을 보여준다. 여기서는 통합메모리 장치인 NVIDIA SoC칩들과 NVIDIA SMX 기반 V100 GPU 카드에서 CPU-GPU 간 데이터 전송 프로그래밍 기법별로 성능비교를 한다. 성능비교를 위해 워크로드는 HPC 분야의 수치계산에서 자주 사용하는 2차원 행렬 전치 커널이다. 실험을 통해 CPU-GPU 메모리 전송 프로그래밍 방법별 GPU 커널 성능차이, 페이지 잠긴 메모리와 페이지 가능 메모리를 사용했을 경우 전송 성능차이, 전체(Overall) 성능비교, 마지막으로 워크로드 크기별 성능비교를 하였다. 이를 통해 통합메모리 칩인 NVIDIA Xavier에서 I/O 캐시일관성 지원을 통해 SoC 칩내 통합메모리에 대한 이점을 극대화 할 수 있음을 확인할 수 있었다.

키워드 : 고성능컴퓨팅, GPU, 통합메모리, 데이터전송

1. 서 론

PCI 익스프레스(PCIe) 인터페이스로 연결된 GPU 카드내 메모리는 호스트(host)에 있는 메모리와 분리되어 있다. 그래서 GPU에서 계산수행을 위해서는 호스트 메모리에서 GPU 메모리로 데이터를 복사하고, 수행결과를 호스트에서 확인하기 위해서 다시 호스트 메모리로 결과를 복사해야 한다. 이 과정에서 호스트와 GPU 메모리를 별도로 관리해야 하는 불편함과 CPU-GPU 사이 데이터 이동으로 인한 성능 저하로 인해 호스트 메모리와 호스트 메모리를 효율적으로 관리하는 연구가 많아지고 있다[1]. 대표적인 예로 하드웨어와 상관없이 소프트웨어 API로 지원하는 호스트 메모리와 GPU 메모리간 제로카피(zero copy)와 통합메모리접근(unified memory access) 기법이 알려져 있다. 두 기법은 호스트 메모리와 GPU 메모리를 소프트웨적으로 연결(mapping)하여 명시적으로 메모리 복사를 호출하지 않아도 GPU 커널에서 호스트 메모리를 바로

※ 이 논문은 대한민국 정부(과학기술정보통신부)의 재원으로 한국연구재단 슈퍼컴퓨터개발선도사업의 지원을 받아 수행된 연구임(과제번호: 2020M3H6A1084857).
※ 이 논문은 2021년 한국정보처리학회 ACK 2021의 우수논문으로 "NVIDIA Tegra와 Tesla GPU에서의 CPU-GPU 데이터 전송성능 연구"의 제목으로 발표된 논문을 확장한 것임.
† 정 회 원 : 한국과학기술정보연구원 슈퍼컴퓨팅본부 책임연구원
†† 비 회 원 : 한국과학기술정보연구원 슈퍼컴퓨팅본부 책임연구원
Manuscript Received : December 23, 2021
Accepted : February 24, 2022
* Corresponding Author : Oh-Kyoung Kwon(okkwon@kisti.re.kr)

접근할 수 있게 한다. 하지만 사용성 관점에서는 편리한 방법일 수 있어도 PCIe를 통한 데이터 이동에 대한 부하가 여전히 발생하므로 성능 측에서 장점은 크지 않다.

최근에는 NVIDIA Tegra, Apple M1 등과 같이 CPU, GPU, 통합메모리 등을 하나의 실리콘 패키지에 통합하는 SoC(System on a Chip) 제품들이 많이 출시되고 있다. 해당 제품들은 계산 시 GPU와 CPU가 물리적으로 동일한 칩내 메모리를 사용하기 때문에 PCIe 인터페이스를 사용하는 GPU 카드보다 메모리 전송 성능이 향상된다. 또한 위에서 언급한 제로카피나 통합메모리 기술을 적용하기에 적합한 구조이다.

해당 연구는 통합메모리를 사용하는 NVIDIA GPU와 PCIe 인터페이스 기반 NVIDIA GPU 환경에서 CPU-GPU 데이터 이동을 위한 프로그래밍 기법들에 대한 성능을 비교한다. 이때 고성능컴퓨팅 분야에서 빈번하게 사용하는 행렬 전치 계산을 대상으로 한다. 다음 세 가지 점이 해당 연구의 주요 기여부분이다.

1. 통합메모리 칩에서 CPU-GPU 데이터 전송 프로그래밍 기법에 대한 연구이다. 특히 하드웨어 I/O 캐시일관성 지원 및 워크로드 크기가 통합메모리 칩내 성능에 미치는 영향을 확인할 수 있다.
2. 페이지 가능 메모리와 잠긴 메모리 사이의 CPU-GPU 데이터 전송 성능비교를 하였다.
3. 다양한 GPU 장치들의 CPU-GPU 데이터 전송 모델의 전체(Overall) 성능비교를 하였다.

2. CPU-GPU 데이터 이동 프로그래밍 비교

이 장에서는 NVIDIA GPU 장치에서 사용할 수 있는 CPU-GPU 데이터 이동을 위한 프로그래밍 기술을 설명한다.

2.1 페이지 가능 메모리 전송

호스트와 GPU 메모리를 각각 독립적으로 메모리로 할당 후 데이터를 서로 전송하는 가장 일반적인 방법이다. 호스트의 메모리는 malloc 함수를 이용해 페이지 가능(pageable) 영역을 할당하고, GPU 메모리는 cudaMalloc 함수를 사용해 공간을 할당한다. 호스트 메모리에서 GPU 메모리로 또는 GPU 메모리에서 호스트 메모리로 데이터를 전송시 cudaMemcpy 함수 등을 이용한다.

2.2 페이지 잠긴 메모리 전송

호스트와 GPU 메모리를 각각 독립적으로 할당하되, 호스트 메모리를 페이지 잠긴(page locked 혹은 pinned) 메모리로 할당하여 전송한다. 호스트는 cudaMallocHost 함수를 사용하여 메모리 공간을 할당하는데, 페이지 가능(pageable) 메모리 영역을 할당하는 malloc 함수보다 더 많은 대역폭을 제공받을 수 있다. 이를 통해 데이터 전송 시 성능향상이 기대된다. 하지만 페이지 잠긴 메모리 영역을 과도하게 많이 할당하면 페이지 영역이 급격히 줄어들어서 데이터 관련 성능에 영향을 미친다.

2.3 제로카피 기술을 활용한 메모리 전송

호스트와 GPU가 별도로 메모리 할당 함수 및 이동함수를 호출할 필요가 없는 방법이다. 즉, cudaHostAlloc 함수를 한번만 호출해서 페이지 잠긴 메모리의 영역을 할당한다. 할당 후 호스트와 GPU 장치 모두 명시적으로 데이터 전송 함수 호출 없이 메모리를 접근할 수 있다.

2.4 통합메모리접근(Unified Memory Access) 기술을 활용한 메모리 전송

소프트웨어적으로 구현된 통합메모리접근 기술을 이용해서 호스트와 GPU 메모리 사이 데이터 전송을 구현하는 방법이다. 메모리 통합 관리 기능을 제공하므로 제로카피 기술처럼 호스트와 GPU가 별도로 메모리 할당 함수 및 이동함수를 호출할 필요가 없다. 그러나 제로카피는 cudaHostAlloc 함수로 페이지 잠긴 영역를 할당하는 반면, 통합메모리접근 기술은 cudaMallocManaged 함수를 사용하며 메모리를 할당한다. 즉, 제로카피는 코드의 성능이 페이지 잠긴 영역에 어느 정도 남아 있는지에 따라 달라질 수 있는 반면, 통합메모리접근기술은 데이터 접근에 대해서 메모리와 실행 영역을 분리하는데 초점이 맞춰있다[2].

3. 통합메모리 장치

3.1 통합메모리 장치

본 연구에서 사용할 NVIDIA Tegra Xavier(Fig. 1)는 CPU, GPU, 통합메모리 등을 하나의 실리콘 패키지에 묶은 SoC (System on a Chip)이다. 이 칩은 원래 스마트폰과 같은 IoT 장치용으로 개발되었지만, 범용적으로도 사용할 수 있다. GPU

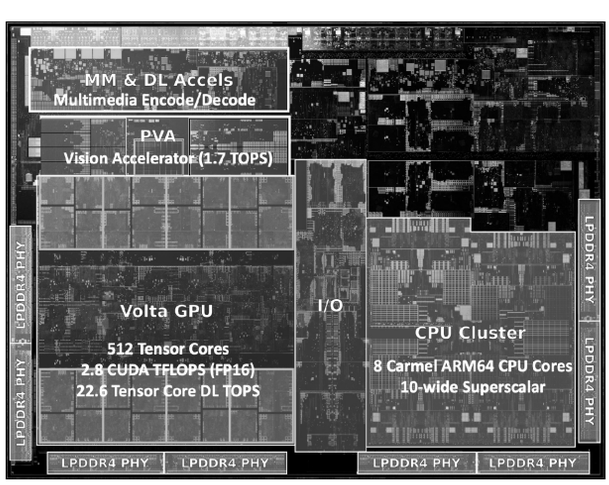

Fig. 1. Architecture of NVIDIA Jetson Xavier

와 CPU 사이의 데이터 전송성능 관점에서 보면 GPU와 CPU가 물리적으로 동일한 곳에 위치한 메모리를 사용하기 때문에 PCIe(SMX) 장치 기반 GPU 카드보다 지연속도(latency)가 줄어들어 전송 성능이 개선된다. 반면 CPU- GPU 데이터 전송 프로그래밍 기법에 따라 성능차이가 크게 난다.

3.2 하드웨어 I/O 캐시일관성

NVIDIA Tegra는 CPU와 GPU가 물리적으로 메모리가 칩내에 같이 있지만, 모델에 따라서 CPU와 GPU에서 메모리 접근시 캐시(cache) 일관성(coherency) 방법이 상이해서 데이터 전송 성능이 다르다. 예를 들어 NVIDIA TX2는 CPU와 GPU에서 메모리 접근시 캐시를 하드웨어 적으로 보장하지 않아서 GPU 계산시 메모리 접근시 캐시 미스(miss)가 발생하면 호스트 메모리에서 데이터를 복사해야 한다. 한편 NVIDIA Xavier 모델은 Fig. 2처럼 하드웨어 캐시 I/O 일관성이 보장되어[4,5], 호스트 메모리 접근 회수를 최소화할 수 있다.이를 통해 GPU 커널 실행시 제로카피 기법을 사용하면 메모리 접근시 TX2보다 성능향상이 가능하다.

3.3 소프트웨어 통합메모리접근에 대한 하드웨어 지원

NVIDIA GPU 장치는 아키텍처, CUDA 버전에 따라 소프트웨어 통합메모리 접근을 지원하는 수준이 다르고, 이에 따라 성능 차이가 발생한다. Kepler 아키텍처(CUDA 6.x 이후)부터 통합메모리에 대한 프로그래밍 모델이 소개되었지만, 페이지 폴트에 대한 하드웨어 지원은 없었다. Pascal 아키텍처(CUDA 8.x)는 페이지 폴트를 하드웨어 레벨에서 지원하고, 가상주소체계(virtual address space)가 48bit까지 제공했다. 그리고 Volta 아키텍처(CUDA 9.x)부터 메모리 접근에 대한 하드웨어 카운터가 추가되면서 핫 페이지(hot pages)에 접근할 때만 데이터 이동이 일어나도록 구현되었다. Pascal 아키텍처 이후부터는 통합메모리접근 기법 사용 시 성능이 향상될 수 있다.

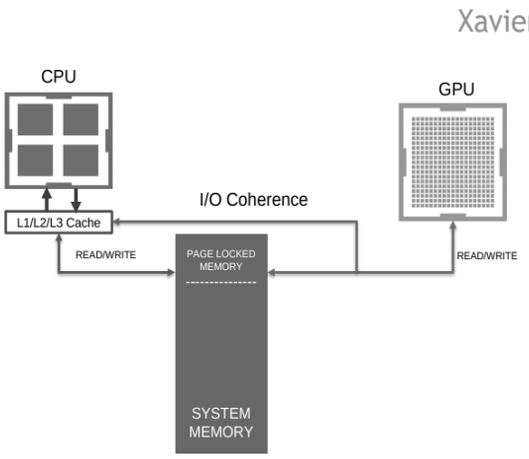

Fig 2. Schematic of I/O Coherence [4]

4. 워크로드

실험은 수치연산에서 자주 사용하는 행렬 전치(matrix transpose) 연산을 사용한다. 이때 2차원 행렬 연산을 사용하며, 각 데이터는 실수(float) 타입이다. 행렬 전치를 위해 Fig. 3처럼 CUDA C로 두 가지 형태의 GPU 커널을 구현하였다. 첫 번째 커널은 GPU 전역 메모리를 직접 접근(access)하는 Naive(matrixTransposeNaive)커널이다. 두 번째 커널은 GPU 전역 메모리 접근을 줄이기 위해 공유 메모리를 사용하여 최적화한 SM(matrixTransposeSharedMemory) 커널이다. 두 가지 커널 성능 비교를 통해 전역 메모리 횟수 증가가 각 프로그래밍 모델 및 통합메모리장치에 미치는 영향을 파악할 수 있다.

두 가지 GPU 커널과 4가지 CPU-GPU 데이터 전송기법의 총 8가지 조합으로 Fig. 4 실행 순서로 수행한다.

1. GPU 메모리 내 데이터를 할당한다. 호스트 메모리에서 GPU 메모리로 이동할 입력 데이터와 계산 후 다시 호스트 메모리로 이동해을 출력 데이터가 대상이다. 할당된 메모

```
__global__ void matrixTransposeNaive(float* x, float* y) {
   int a = blockIdx.x * blockDim.x + threadIdx.x;
   int b = blockIdx.y * blockDim.y + threadIdx.y;
   int in  = b * sizeX + a;
   int out = a * sizeY + b;
   y[out] = x[in];
}
__global__ void matrixTransposeSharedMemory(float* x, float* y)
{
   __shared__ float mat[BLOCK_SIZE_Y][BLOCK_SIZE_X];
   int block_x = blockIdx.x * BLOCK_SIZE_X;
   int block_y = blockIdx.y * BLOCK_SIZE_Y;
   int i  = block_x + threadIdx.x;
   int j  = block_y + threadIdx.y;
   int ii = block_y + threadIdx.x;
   int jj = block_x + threadIdx.y;
   if(i < sizeX && j < sizeY) {
     mat[threadIdx.y][threadIdx.x] = x[j * sizeX + i];
   }
   __syncthreads();
   if(ii < sizeY && jj < sizeX){
     y[jj * sizeY + ii] = mat[threadIdx.x][threadIdx.y];
   }
}
```

Fig. 3. GPU Kernel Code of Matrix Transpose

```
1. GPU memory (Global)allocation(pre-process)
2. Copy data from host(CPU) memory to device(GPU) memory
   (HtoD copy)
3. GPU kernel execution (kernel)
4. Copy data from device(GPU) memory to host(CPU) memory
   (DtoH copy)
5. Copy data from page-locked memory to pageable memory
   (memcpy)
6. Execution validation (post-process)
```

Fig. 4. Workload Execution Order

리에 데이터를 초기화하기 위해 페이지 가능 메모리를 이용한 전송과 페이지 잠긴 메모리를 이용한 전송기법은 cudaMemset 함수를 이용하고 나머지 데이터 전송기법은 할당된 변수에 데이터를 직접 접근해서 쓴다(write).

2. 호스트 메모리에 할당되어 있는 행렬 데이터를 GPU 메모리로 전송한다. 해당 단계는 페이지 가능 메모리와 페이지 잠긴 메모리를 사용할 때만 사용한다.
3. 행렬 전치에 대한 GPU 커널을 수행한다.
4. GPU 메모리에 있는 커널 실행 결과를 호스트 메모리로 전송한다. 2번처럼 페이지 가능 메모리와 페이지 잠긴 메모리를 사용할 때만 해당 단계가 필요하다.
5. 결과 데이터를 페이지 잠긴 메모리에서 페이지 가능 메모리로 복사한다. 이 단계는 NVIDIA TX2에서만 필요하다. NVIDIA TX2의 경우 CPU에서 결과 검증시 페이지 잠긴 메모리에서 데이터를 읽을(read) 때 성능 저하가 많이 발생한다. 그래서 페이지 가능 메모리로 복사하는 단계가 필요하다.
6. 마지막 단계로 CPU에서 결과가 제대로 계산되었는지 검증한다.

5. 실험 결과

앞선 4장에서 언급한 총 8가지 조합을 Table 1에 나열한 3가지 GPU장치에서 실험하였다. 여기서 V100-SMX2는 PCIe가 아닌 SMX2를 사용하는데, 본 실험은 단일 GPU만 사용하므로 PCIe 대비 CPU-GPU 전송성능 차이는 없다. SMX2는 PCIe 대비 GPU 간 데이터 전송에서 큰 대역폭을 제공한다.

해당 장에서는 Fig. 5에 나타난 결과를 바탕으로 다음 네 가지 관점에서 성능을 비교·분석하고자 한다

1. 호스트(CPU) 메모리와 GPU 메모리 사이 데이터 전송 방법에 따라 GPU 커널 성능 차이.
2. 페이지 잠긴 메모리와 페이지 가능 메모리 기법 간 데이터 전송성능 차이.
3. 계산 및 전송 시간을 포함한 전체(overall) 성능 비교.
4. NVIDIA Xavier에서 행렬 크기에 따른 수행 성능 비교.

첫 번째, 세 가지 GPU 장치별로 데이터 전송 기법에 따라 GPU 커널 성능 차이가 발생하는지 알아보자(Fig. 5). Xavier는 기법별로 커널 간 성능 차이가 거의 없는 반면, TX2와 V100-SMX2는 확연한 성능 차이를 보여준다. Xavier 경우

Table 1. GPU Device and CUDA Version used in the Experiment

GPU Device	Memory Specification (Size, Bandwidth)	CUDA	Architecture
NVIDIA Tesla V100 SMX2	HBM2 (16GB, 900.1GB/s)	11.0	Volta
NVIDIA TX2	LPDDR4 (8GB, 59.7GB/s)	10.2	Pascal
NVIDIA Xavier	LPDDR4x (32GB, 136.5GB/s)	10.2	Volta

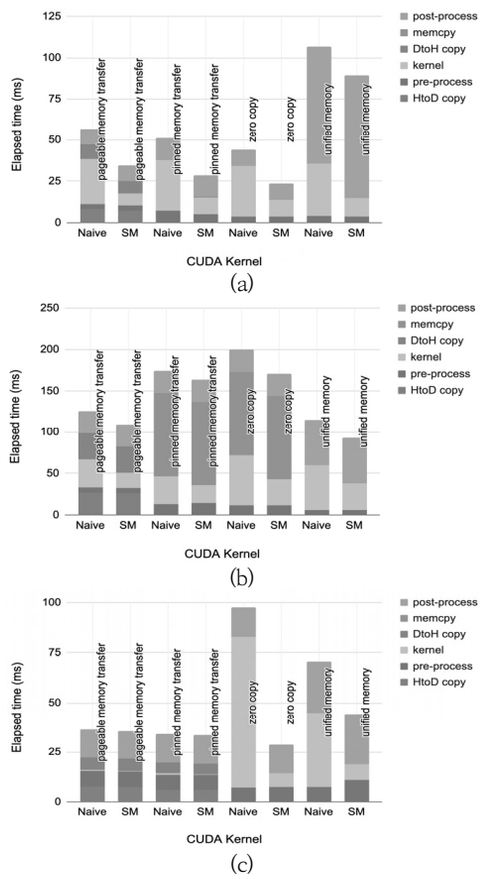

Fig. 5. Performance Comparison of Three GPU Devices According to CPU-GPU Data Transfer Method (Matrix Size: 4096 x 4096): (a) NVIDIA Xavier (b) NVIDA TX2 (c) NVIDIA V100-SMX2

제로카피와 통합메모리접근 기법은 하드웨어 I/O 캐시 일관성 지원으로 인해 GPU에서 커널을 실행할 때 메모리 접근에 대한 성능 저하가 거의 발생하지 않고 8가지 조합들이 일정하게 성능이 유지된다. 반면 TX2와 V100-SMX2는 GPU에서 커널 실행 시 캐시에 데이터 접근시 페이지 폴트가 계속 발생하여 메모리에 있는 데이터를 GPU 캐시로 가져오기 때문에 성능이 나빠진다. 특히 TX2가 V100-SMX2 보다 GPU 커널 성능 차이가 발생하는 이유는 TX2는 물리적으로 통합된 메모리 내부에서 데이터 이동인 반면 V100-SMX2는 물리적으로 분리된 메모리 간 전송이기 때문이다. 그리고 V100-SMX2 경우 Naive 커널 실행시 제로카피 기법 사용이 통합메모리접근 기법보다 성능이 좋지 않다. 그 이유는 GPU 전역 메모리 접근시 제로카피의 페이지 폴트 관련 기능이 최적화 되어 있지 않기 때문이다. 3.3절에서 언급하였듯이 NVIDIA는 Pascal 이후 아키텍처는 통합메모리접근 기법 사용 시 페이지 폴트에 대해 하드웨어 지원을 하고 있다. 반면 SM 커널은 전역 메모리 접근 회수가 이미 줄어서, 통합메모리접근 방법 으로 인한 성능 개선이 많이 없다. 한편 TX2 경우에는 V100-SMX2와 달리 GPU 커널 실행 시 제로카피와 통합메모리접근 기법 관련 성능 차이가 크지 않았다.

두 번째, 페이지 가능 메모리와 페이지 잠긴 메모리의 데

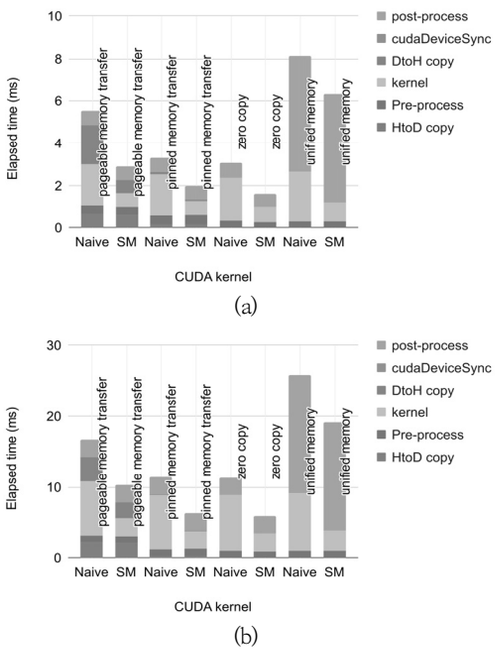

Fig. 6. Performance Evaluation with Matrix Size According to CPU-GPU Data Transfer Method at NVIDIA Xavier Device: (a) 1024 x 1024 (b) 2048 x 2048

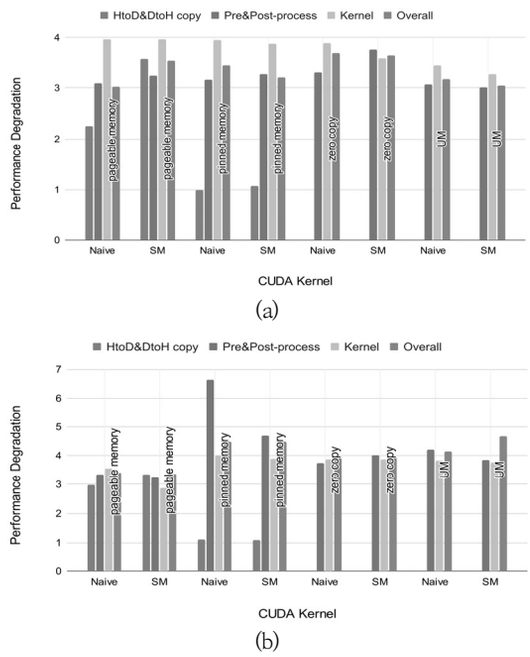

Fig. 7. Performance Degradation with Matrix Size According to CPU-GPU Data Transfer Method at NVIDIA Xavier Device: (a) 2048 x 2048 over 1024 x 1024 (b) 4096 x 4096 over 2048 x 2048

이터 전송성능을 비교해보자. 호스트 메모리에서 페이지 잠긴 메모리로 할당되었을 경우 호스트 메모리에서 GPU 메모리로 이동은 Xavier, TX2, V100-SMX2는 페이지 가능 메모리 대비 각각 약 48배, 164배, 1.26배, GPU 메모리에서 호스트 메모리로 이동은 각각 72배, 242배, 1.14배 향상되었다. 호스트 메모리를 페이지 잠긴 메모리로 할당하면, Xavier와 TX2에서 명시적으로 데이터 전송 함수를 호출할 때 속도는 빠르지만, CPU에서 코드(pre-process와 post-process)을 실행하거나 GPU에서 커널 실행 시 메모리 접근 성능이 더 나빠지는 것을 확인할 수 있다.

세 번째, 데이터 전송과 GPU 커널 실행 시간 등을 모두 포함한 전체(overall) 성능을 비교해보자. Xavier는 데이터 전송 기법 중 제로카피가 가장 빠른 성능을, 통합메모리접근이 가장 성능이 좋지 않음을 보여준다. 제로카피를 사용한 Naive와 SM 커널은 페이지 가능 메모리 전송 방법을 사용한 Naive와 SM 커널보다 각각 1.28배, 1.47배 빠르다. 통합메모리접근 방법은 CPU에서 결과를 검증(post-processing, Fig. 5의 청록색)할 때 성능이 많이 느려지는데, 이것은 소프트웨어 최적화 문제로 추측된다. TX2 경우에 페이지 가능 메모리와 통합메모리접근 방법을 사용한 데이터 전송이 가장 빠르며, 제로카피 방법은 페이지 폴트를 처리에서 최적화 문제로 여겨진다. 그리고 페이지 잠긴 메모리와 제로카피를 이용한 데이터 전송 기법은 앞서 언급한 것처럼 페이지 가능 메모리 영역으로 복사(memcpy, Fig. 5의 주황색)하는 과정으로 성능 저하가 추가적으로 발생한다. V100-SMX2 장치에서 Naive 커널 경우에 페이지 잠긴 메모리를 이용한 데이터 전송이, SM 커널은 제로카피 기법을 이용한 전송 성능이 가장 좋다. Naive 커널은 페이지 잠긴 메모리가 페이지 가능 메모리를 이용한 경우보다 1.07배, SM 커널은 제로카피가 페이지 가능 메모리 보다 1.24배 속도가 빠르다. 또한 SM 커널은 Naive 커널 보다 전역 메모리 접근 횟수가 적기 때문에 커널 실행 시 성능부하가 줄었다.

마지막으로, NVIDIA Xavier에서 행렬 크기에 따른 성능을 비교한다. Fig. 5(a)와 Fig. 6(a)(b)는 NVIDIA Xavier에서 4096x4096, 1024x1024, 2048x2048의 크기를 갖는 행렬을 전치할 때 측정한 요소별 성능을 보여주고, Fig. 7은 커널 크기, 전송 속도, 전체 성능에 대해서 행렬 크기에 따른 성능 감소 비율을 나타내었다. Fig. 6을 통해 커널 크기와 상관없이 제로 카피를 이용한 경우가 전체 성능이 가장 빠름을 알 수 있고, 전체 성능(Fig. 7의 초록색)은 Fig. 5(a), Fig. 6(a)(b)에서 보듯이 커널 크기가 커짐에 따라 성능감소 정도가 커짐을 알 수 있다. 그리고 GPU 커널 속도(Fig. 7의 오렌지색)는 행렬 크기에 비례해서 약 4배 정도 일정하게 느려지지만, 호스트와 GPU 메모리 간 전송성능(Fig. 7의 파란색)은 행렬 크기에 비례해서 페이지 가능 메모리가 페이지 잠긴 메모리보다 크게 느려짐을 알 수 있다. 그리고 전처리(Pre-process)와 후처리(Post-process)에서 행렬 크기가 커지면 성능감소 정도가 많이 커지는데(Fig. 7의 빨간색), Xavier 메모리 대역폭이 원인으로 보인다.

6. 관련 연구

Oh-Kyoung Kwon 연구는 NVIDIA Tegra와 같은 통합메모리 칩에서 유사한 성능비교를 수행하였지만, 행렬 크기에 따른 성능 차이를 비교하지 않았다[6]. 한편, 다른 연구에서는

통합메모리 칩에서 연구한 경우보다는 PCIe 기반 GPU 카드에 대해 CPU-GPU 데이터 전송 프로그래밍 기법의 성능비교 연구가 대부분이다[1,7,8]. R. S. Santos 연구[1]는 CPU- GPU 데이터 전송 프로그래밍 4가지 기법을 Kepler 아키텍처에서 성능비교를 수행하였다. 이때, 제로카피 모델이 페이지 잠긴 메모리 전송 모델 대비 평균 19% 이상 성능향상이 있었고, 통합메모리접근 모델은 성능이 가장 좋지 않았다. 하지만, 통합메모리접근 사용시 페이지 폴트에 대한 하드웨어 지원이 없는 Kepler 아키텍처에서 수행되어 본 연구에서 수행한 결과와 비교하기가 어렵다. 반면, Steven Chien[7]과 Pengyu Wang [8] 연구는 NVIDIA Volta 아키텍처 이후 모델에서 통합메모리접근 기법의 성능 비교를 수행하였다. 해당 연구에서는 하드웨어 지원을 통해 특정 응용프로그램에서 통합메모리접근 성능이 다른 기법과 유사하게 성능이 나옴을 확인할 수 있다.

7. 결론 및 향후 연구

NVIDIA V100-SMX2 GPU 카드, NVIDIA Tegra Xavier, NVIDIA TX2 장치에서 4가지 CPU와 GPU 사이 데이터 전송 기법별로 행렬 전치 코드를 사용해서 성능 비교를 하였다. 실험을 통해, Xavier는 하드웨어적으로 I/O 캐시일관성 지원을 통해 데이터 전송 기법과 상관없이 GPU 커널 성능 차이가 거의 없었고, TX2와 V100-SMX2는 제로카피와 통합메모리접근 기법에서 커널 간 성능 차이가 발생함을 알 수 있었다. 즉, Xavier에서는 하드웨어적으로 I/O 캐시일관성 지원을 통해 SoC 칩내 통합메모리에 대한 이점을 극대화 할 수 있음을 확인할 수 있었다. 그리고 페이지 잠긴 메모리를 사용하면 페이지 가능 메모리 대비 CPU-GPU 사이 데이터 전송 속도가 향상됨을 확인할 수 있었다. 이를 통해 페이지 가능 메모리를 사용할 수 있는 워크로드 즉, 메모리를 아주 많이 사용하지 않는다면, 가능하면 페이지 가능 메모리를 사용하는 것이 바람직함을 알 수 있었다. 전체 성능면에서 Xavier, TX2, V100-SMX2는 각각 제로카피, 페이지 가능 메모리 전송, 페이지 잠긴 메모리 전송 방법이 가장 빨랐다. 마지막으로 NVIDIA Xavier에서 각 행렬 크기별로 성능 비교를 하였는데, 전체 성능은 커널 크기가 커짐에 따라 성능감소 정도가 커짐을 알 수 있었다.

반도체 공정기술이 발전됨에 따라 향후 통합메모리를 사용한 장치가 많이 출시될 것으로 예상된다. 본 연구에서는 통합메모리가 아닌 NVIDIA GPU 환경에서 사용된 프로그래밍 4가지 방법을 사용하여 테스트하였지만, OpenCL의 SVM(Shared Virtual Memory) 등 통합메모리를 지원하는 다른 프로그래밍 모델에 대해서도 검토할 필요가 있다. 또한 해당 연구에서는 행렬 전치와 같은 벤치마크용 커널 코드를 이용해서 테스트를 수행하였는데, 실제 응용프로그램은 I/O 등 워크로드가 복잡하므로 이에 대해서 다양하게 테스트할 필요가 있다. 특히 CPU와 GPU간 데이터 전송이 빈번한 워크로드라면, Xavier와 같은 통합메모리 칩에서 제로카피 기법이 다른 프로그래밍 모델 대비 성능가속이 많이 될 것으로 예상된다.

References

[1] R. S. Santos, D. M. Eler, and R. E. Garcia, “Performance evaluation of data migration methods between the host and the device in CUDA-based programming,” *Information Technology: New Generations*, pp.689-700, 2016.

[2] CUDA C++ Programming Guide [Internet], https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.

[3] CUDA for Tegra [Internet], https://docs.nvidia.com/cuda/cuda-for-tegra-appnote/index.html.

[4] Anshuman Bhat, CUDA on Xavier, GTC 2018 [Internet], http://on-demand.gputechconf.com/gtc/2018/presentation/s8868-cuda-on-xavier-what-is-new.pdf.

[5] Nikolay Sakharnykh, Everything You Need Toknow about Unified Memory, GTC 2018 [Internet], https://on-demand. gputechconf.com/gtc/2018/presentation/s8430-everything-you-need-to-know-about-unified-memory.pdf.

[6] O. K. Kwon and G. Gu, “A performance study on CPU-GPU data transfers of NVIDIA tegra and tesla GPUs,” *Proceedings of Annual Conference of KIPS 2021*, pp.39-42, 2021.

[7] S. Chien, I. Peng, and S. Markidis, “Performance evaluation of advanced features in CUDA unified memory,” *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, pp.8-18, Nov. 2019.

[8] P. Wang, J. Wang, C. Li, J. Wang, H. Zhu, and Guo, M. “Grus: Toward unified-memory-efficient high-performance graph processing on GPU,” *ACM Transactions on Architecture and Code Optimization (TACO)*, Vol.18, No.2, pp.1-25, 2021.

권 오 경

https://orcid.org/0000-0001-9734-9257
e-mail : okkwon@kisti.re.kr
1996년 ~ 2000년 고려대학교 컴퓨터학과 (학사)
2000년 ~ 2002년 KAIST 전산학과(석사)
2002년 ~ 현 재 한국과학기술정보연구원 슈퍼컴퓨팅본부 책임연구원
관심분야 : 고성능컴퓨팅, 병렬컴퓨팅, 이기종컴퓨팅

구 기 범

https://orcid.org/0000-0001-5313-123X
e-mail : gibeom.gu@kisti.re.kr
1993년 ~ 1997년 서강대학교 컴퓨터공학과 (학사)
1997년 ~ 1999년 서강대학교 컴퓨터공학과 (석사)
2002년 ~ 현 재 한국과학기술정보연구원 슈퍼컴퓨팅본부 책임연구원
관심분야 : 고성능컴퓨팅, 컴퓨터 그래픽스