OpenCL을 사용한 돈사 감시 응용의 효율적인 태스크 분배

김진성*, 최윤창*, 김재학*, 정연우*, 정용화*, 박대희*, 김학재**
*고려대학교
**클래스액트
e-mail: skykeeop@korea.ac.kr

Efficient Task Distribution of Pig Monitoring Application using OpenCL

J. Kim*, Y. Choi*, J. Kim*, Y. Chung*, Y. Chung*, D. Park*, and H. Kim**
*Korea University
**ClassAct

요 약

돈사 감시 응용은 내재된 데이터 병렬성을 활용하고 성능가속기를 사용하여 병렬처리가 가능하다. 본 논문에서는 multicore-CPU와 manycore-GPU로 구성된 이기종 컴퓨팅 환경에서 돈사 감시 응용 수행 시 태스크 분배 방법을 제안한다. 즉, 각 태스크별로 OpenCL로 작성된 병렬 프로그램을 deviceCPU와 deviceGPU 각각에서 수행시켜 측정된 수행시간을 기준으로 가장 적합한 처리기를 결정한다. 제안 방법은 간단하지만 매우 효과적이고, CPU와 GPU로 구성된 이기종 컴퓨팅 플랫폼에서 다른 응용을 병렬화하는데에도 적용될 수 있다. 실험 결과, 상이한 이기종 컴퓨팅 플랫폼에서 최적의 태스크 분배로 수행 한 경우가 전체 태스크들을 deviceGPU에서 수행한 방법에 비교하여 각각 2배, 11배 성능 개선이 되었음을 확인하였다.

1. 서론

최근 고성능 컴퓨팅과 모바일 컴퓨팅 분야에서 성능가속기인 GPU(Graphics Processing Uunit), DSP(Digital Signal Processing), FPGA(Field Programmable Gate Array) 등을 사용하여 성능을 개선하는 연구가 활발히 진행되고 있다. 그러나 GPU의 범용 병렬프로그래밍 프레임워크인 CUDA의 경우 NVIDIA 계열의 제품에서만 동작하는 단점이 있고, CPU와 같은 마이크로프로세서에서는 지원을 하지 않는다. 이러한 이식성(portability)의 문제점을 해결한 것이 최근 발표된 OpenCL 표준[1,2]이다.

많은 OpenCL 응용들은 성능가속기를 대상으로 병렬처리를 하고 있고, 성능가속기 중에서도 GPU를 사용하여 여러 응용들을 병렬처리 한 연구가 발표되었다. 그러나 주어진 태스크의 계산 특성과 수행 플랫폼내 CPU-GPU의 상대적인 성능에 따라서 멀티코어 CPU를 이용한 병렬처리가 더 좋은 성능을 제공할 수도 있다. 또한, 태스크 스케쥴링에 대한 많은 기존 연구가 있지만, 대부분의 경우 분산 시스템하에서 작업부하 분배(workload distribution)에 관한 시스템 레벨의 연구[3-5]이고, 본 논문에서 가정하는 병렬 시스템하에서 태스크 분배에 관한 응용프로그램 레벨에서의 연구는 거의 발표되지 않고 있는 실정이다.

본 논문에서 가정하는 OpenCL은 이기종 컴퓨팅 환경에서 병렬처리를 할 수 있는 프레임워크이기 때문에, 태스크별로 CPU 또는 GPU를 활용하여 병렬처리 할 수 있다. 본 논문에서는 이러한 OpenCL의 특징을 이용하여, 돈사 감시[6,7] 응용의 각 태스크를 CPU 또는 GPU에 할당하여 수행시간을 단축하는 방법을 제안한다. 실험 결과, 어떠한 경우에도 제안 방법을 활용하면 전체 태스크들을 GPU만을 이용하는 경우보다 수행시간을 단축할 수 있음을 확인하였다.

2. 제안 방법

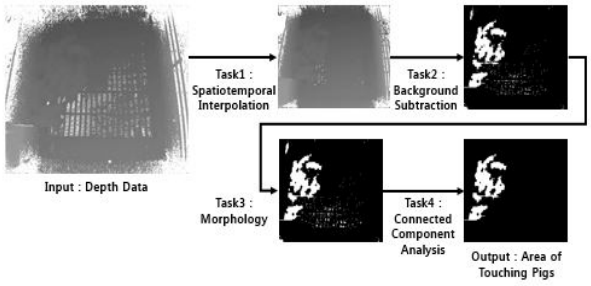
OpenCL 플랫폼은 한 개의 호스트와 한 개 이상의 디바이스들로 구성되고, OpenCL이 동작하기 위해서는 호스트 메모리에서 커널 메모리로 연산에 필요한 데이터가 복사되어야 한다. 마찬가지로 커널 프로그램 수행이 종료가 되면 디바이스에서 호스트로 연산된 데이터가 복사되어야 한다. 본 논문에서는 하나의 플랫폼에 하나의 호스트(host로 명기)와 두개의 디바이스(deviceCPU, deviceGPU로 명기)가 장착된 경우를 가정하고, 돈사 감시 응용은 마이크로소프트의 키넥트 깊이 정보를 이용하여 누워있는 돼지를 탐지하는 경우를 가정한다.

2.1 키넥트 깊이 정보를 이용한 누워있는 돼지 탐지

본 논문에서는 돈방내 돼지들이 실제로 체감하는 온도를 자동으로 파악하고 조절하기 위하여 누워있는 돼지들의 몰려있는 정도[6]를 천장에 설치된 키넥트 카메라를 이용하여 판단하고자 한다. 특히, 야간에는 조명을 소등하기 때문에 키넥트의 컬러 정보 대신 깊이 정보를 이용한다[7].

통상 실내 환경에서 고정된 카메라로 객체를 탐지하는 방법은 객체들이 존재하지 않는 배경 영상을 획득한 후 입력된 영상에서 배경과의 차이를 구하고 Otsu 알고리즘[8]을 적용하여 이진화 시키는 방법을 이용한다. 그러나 저가 센서인 키넥트를 이용하여 획득된 깊이 정보에는 많은 노이즈와 일관되지 않은 깊이값이 존재하는 문제가 있다.

본 논문에서는 이러한 문제를 해결하기 위하여 깊이 영상의 크기를 가로/세로 각각 절반씩 줄인 spatial interpolation을 수행하고, 돼지가 없는 빈 돈방을 10분간 촬영한 후 각 픽셀의 최빈값으로 배경 영상을 생성한다. 입력된 영상에서도 동일한 spatial interpolation을 적용하고, 추가로 30 frames per second(fps) 속도의 입력 영상을 10 fps 속도로 다운샘플하는 temporal interpolation을 적용한다. 이렇게 spatiotemporal interpolation된 입력 영상과 배경 영상의 차이를 구한 후, 모폴로지 연산을 통하여 경계 부분 조정을 수행한다. 최종적으로 Connected Component Labeling후 레이블링된 면적이 두 마리 이상의 돼지, 즉 Touching-Pigs의 면적을 계산함으로써 근접하여 누워있는 돼지들의 몰려있는 정도를 자동으로 판단한다. 이러한 태스크들의 흐름도를 정리하면 그림 1과 같다.

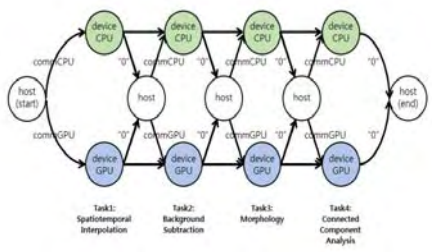


(그림 1) 누워있는 돼지 탐지를 위한 태스크 흐름도

2.2 돼지 탐지의 태스크 분배

본 논문에서는 태스크를 스케쥴링의 최소 단위로 가정하며, 2.1에서 언급한 4가지 태스크들은 내부적으로 모든 계산이 병렬로 처리될 수 있는 것과 데이터 종속성에 의해 일부 계산은 순차로 처리되어야 하는 상이한 계산 특성을 갖는다. 즉, 태스크의 계산 특성과 이기종 플랫폼내 CPU-GPU의 상대적인 성능에 따라 각 태스크별로 적합한 처리기가 있고, 각 태스크의 계산에 필요한 데이터의 통신 오버헤드도 고려해야 한다.

본 논문에서는 돼지 탐지의 4가지 태스크가 하나의 deviceCPU와 하나의 deviceGPU로 구성된 이기종 플랫폼에서 수행될 수 있는 시나리오를 DAG 형태의 구현 그래프(Implementation Graph)로 표현(그림 2 참조)하고, 이를 활용하여 전체 비용이 최소가 되는 태스크 분배 알고리즘을 제안한다. 특히, deviceCPU와 deviceGPU간에는 직접 데이터 통신이 불가하여 호스트를 경유하는 통신 오버헤드를 가정한다. 또한, 각 태스크별로 적절한 디바이스 결정을 편리하게 하기 위하여 통신 오버헤드를 호스트에서 디바이스로 갈 때("host-to-device + device-to-host")와 디바이스에서 호스트로 올 때("0")를 차별화한다. 즉, 그림 2에서 "commCPU"란 "host-to-deviceCPU + deviceCPU-to-host"를 의미하고, "commGPU"란 "host-to-deviceGPU + deviceGPU-to-host"를 의미한다.



(그림 2) 돼지 탐지를 위한 구현 그래프

플랫폼별로 태스크 분배를 결정하는 그리디 알고리즘은 다음과 같다. "Large-Task with Large-Parallelism First" 원칙에서 Large-Task란 순차 처리 시간이 오래 걸리는 것을 의미하고 Large-Parallelism이란 병렬 처리시 speedup(통신 오버헤드 고려)이 큰 것을 의미한다. 즉, 각 태스크별로 deviceCPU와 deviceGPU에서의 순차 처리 시간가 병렬 speedup을 곱하여 하나의 척도로 표현하고, 이 척도를 기준으로 가장 파급효과가 큰 것을 먼저 선택하는 방법으로 4가지 태스크에 대한 적절한 처리기를 결정한다. 그림 3에 제안 알고리즘을 정의하였고, n개의 태스크에 대한 스케쥴링 알고리즘의 시간 복잡도는 $O(n^2)$이다. 또한, 서론에서 언급한 기존의 작업부하 분배 알고리즘은 매순간 수행되지만, 본 논문에서 제안한 스케쥴링 알고리즘은 돼지 탐지 응용의 최초 한번만 수행되므로 스케쥴링 알고리즘에 대한 부담은 없다고 해도 무방하다.

Input : Implementation Graph DAG [n-tasks, 2-workers(deviceCPU, deviceGPU)]
Output : Final Distribution Matrix [n-tasks]

Step1. Construct Efficiency Matrix [n-tasks, 2-workers]
For $i = 1 \sim n$

$$Efficiency[i, worker] = \frac{Comp[i, host]^2}{Comp[i, worker] + Comm[i, worker]}$$

Step2. Construct Initial_Distribution Matrix [n-tasks]
For $i = 1 \sim n$
$Initial\_Distribution[i] = \max(Efficiency[i, deviceCPU], Efficiency[i, deviceGPU])$

Step3. Determine Final_Distribution Matrix [n-tasks]
For $i = 1 \sim n$
Find a max from Initial_Distribution Matrix
Suppose the max is the case where task j is executed on worker k
$Final\_Distribution[j] = k$;
$Initial\_Distribution[j] = -1$;
$Initial\_Distribution[j+1]$ and $Initial\_Distribution[j+1]$ are updated such that $Comm[j-1, k] = Comm[j+1, k] = 0$;

(그림 3) 태스크 분배 알고리즘

3. 실험 결과

본 실험을 위해 두 개의 플랫폼(플랫폼1: 3.50GHz

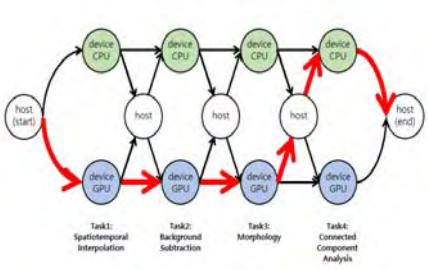
Intel ® CoreTM i5-4690 CPU, GeForce GTX 760, 8GB RAM, 플랫폼2: 2.67GHz Intel ® CoreTM i5 CPU 7504, GeForce GTS 250, 8GB RAM)을 준비하였고, 각각의 플랫폼에서 측정한 태스크별 수행시간을 정리하면 표 1과 같다. 예를 들어, 플랫폼1에서 Task1의 CPU 병렬처리 시간(0.311ms)이 GPU 병렬처리 시간(0.083ms) 보다 빠르지 않음을 확인할 수 있다. 반면, 플랫폼2에서는 캐쉬 등의 효과로 Task1의 4-코어 CPU 병렬처리 시간(0.290ms)이 1-코어 순차처리 시간(8.518ms)의 1/4 보다 적으며 GPU 병렬처리 시간(0.430ms) 보다도 적다. 즉, 각 태스크별로 상이한 계산 특성 때문에 병렬 처리 시간이 다르고 CPU-GPU의 상대적인 성능에서도 플랫폼별로 상이하여, 최적의 태스크 분배에 대한 필요성이 있음을 확인할 수 있다. 특히, Task4에서 중간값들을 merge하는 과정의 시간을 단축시키기 위한 방법[9]을 적용하였음에도 불구하고, 플랫폼에 상관없이 GPU의 처리시간이 많이 느린 특성이 있다. 이는 Connected Component Analysis의 계산 특성상 multicore-CPU보다 manycore-GPU에서 중간값 merge하는 단계수가 훨씬 많기 때문이다.

<표 1> 태스크별 수행시간 (단위: msec)

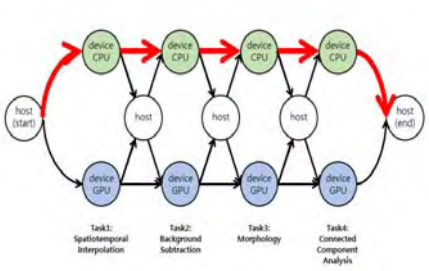
플랫폼1				
		병렬처리		순차처리
		deviceGPU	deviceCPU	host(CPU)
통신 오버헤드	host-to-device	0.063	0.050	-
	device-to-host	0.113	0.036	-
계산 시간	Task1	0.083	0.311	4.342
	Task2	0.086	0.150	0.188
	Task3	0.123	0.326	0.417
	Task4	10.591	4.282	21.545
	Total	0.113	5.155	26.492

플랫폼2				
		병렬처리		순차처리
		deviceGPU	deviceCPU	host(CPU)
통신 오버헤드	host-to-device	0.195	0.025	-
	device-to-host	0.188	0.074	-
계산 시간	Task1	0.430	0.290	8.518
	Task2	0.463	0.198	0.426
	Task3	0.505	0.235	0.772
	Task4	51.473	3.851	46.728
	Total	53.254	4.673	56.444

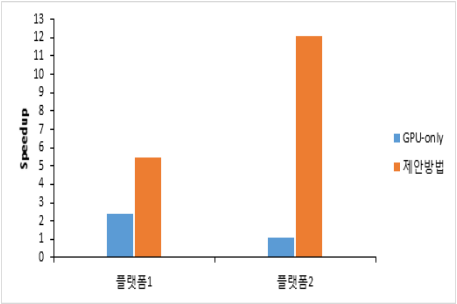
또한, 제안된 알고리즘으로 각 플랫폼별 태스크를 분배한 결과를 나타내면 그림 4, 5와 같고, 이렇게 분배된 경우와 전체 태스크들을 deviceGPU에서 수행한 경우(GPU-only로 명기)의 성능을 비교하면 그림 6과 같다. 즉, 전체 태스크들을 deviceGPU에서만 수행한 GPU-only 방법과 비교하여 제안 방법은 플랫폼1에서 2배, 플랫폼2에서는 11배 향상된 성능을 확인하였다. 이렇게 성능 차이가 나는 이유는 앞서 설명한 Task4의 계산 특성이 GPU와는 잘 매치되지 않는데, 제안 방법에서는 이를 반영하여 Task4를 CPU에서 수행하지만 GPU-only 방법에서는 그렇지 못하기 때문이다.



(그림 4) 플랫폼1에서의 태스크 분배 결과



(그림 5) 플랫폼2에서의 태스크 분배 결과



(그림 6) 각 태스크 분배의 성능 비교

4. 결론

본 논문에서는 이기종 컴퓨팅 환경에서 돈사 감시 응용의 병렬처리를 위하여 OpenCL을 이용하여 CPU-GPU 간 태스크를 분배하는 방법을 제안하였다. 실험 결과, 태스크의 계산 특성에 따라 적합한 처리기가 있고, 플랫폼내 CPU-GPU의 상대적인 성능에 따라서도 태스크 분배가 달려져야 함을 확인하였다. 즉, 전체 태스크들을 deviceGPU에서 수행한 방법과 비교하여 제안 방법은 플랫폼1에서 2배, 플랫폼2에서는 11배 향상된 성능을 확인하였다.

감사의 글

본 연구는 2016년도 미래창조과학부의 재원으로 한국연구재단의 지원을 받아 수행된 지역신산업선도인력양성사업 (2016H1D5A1910730)으로 수행된 연구결과임.

참고문헌

[1] J. Stone, D. Gohara, G. Shi, “OpenCL: A Parallel Programming Standard for Heterogeneous Computing

Systems," *Computing in Science and Engineering*, 12(3), 66-73, 2010.

[2] D. Kaeli, P. Mistry, D. Schaa, D. Zhang, *Heterogeneous Computing with OpenCL 2.0*, Morgan Kaufmann, 2015.

[3] W. Wang, Y. Chang, W. Lo, Y. Lee, "Adaptive Scheduling for Parallel Tasks with QoS Satisfaction for Hybrid Cloud Environments," *J. of Supercomputing*, 66(2), 783-811, 2013.

[4] Y. Hao, M. Xia, N. Wen, R. Hou, H. Deng, L. Wang, Q. Wang, "Parallel Task Scheduling under Multi-Clouds," *TIIS*, 11(1), 39-60, 2017.

[5] F. Ramezani, J. Lu, J. Taheri, F. Hussain, "Evolutionary Algorithm-based Multi-Objective Task Scheduling Optimization Model in Cloud Environments," *WWW-Internet & Web Info. Sys.*, 18(6), 1737-1757, 2015.

[6] B. Shao, H. Xin, "A Real-Time Computer Vision Assessment and Control of Thermal Comfort for Group-Housed Pigs," *Computers and Electronics in Agriculture*, 62(1), 15-21, 2008.

[7] 김진성, 최윤창, 사재원, 주미소, 정용화, 박대희, 김학재, "질감 정보를 이용한 돈방의 배경 제거," *스마트미디어학회 추계학술대회*, 2016.

[8] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Tr. Systems, Man, and Cybernetics*, Vol. 9, No. 1, pp. 62-66, 1979.

[9] I. Jung, C. Jeong, "Parallel Connected-Component Labeling Algorithm for GPGPU Applications," Proc. of ISCIT, 1149-1153, 2010.