KIPS Tr. Comp. and Comm. Sys.
Vol.6, No.10 pp.407~414 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2017.6.10.407

Efficient Task Distribution for Pig Monitoring Applications Using OpenCL

Jinseong Kim† · Younchang Choi†† · Jaehak Kim††† · Yeonwoo Chung†††† · Yongwha Chung††††† · Daihee Park†††††† · Hakjae Kim†††††††

ABSTRACT

Pig monitoring applications consisting of many tasks can take advantage of inherent data parallelism and enable parallel processing using performance accelerators. In this paper, we propose a task distribution method for pig monitoring applications into a heterogenous computing platform consisting of a multicore-CPU and a manycore-GPU. That is, a parallel program written in OpenCL is developed, and then the most suitable processor is determined based on the measured execution time of each task. The proposed method is simple but very effective, and can be applied to parallelize other applications consisting of many tasks on a heterogeneous computing platform consisting of a CPU and a GPU. Experimental results show that the performance of the proposed task distribution method on three different heterogeneous computing platforms can improve the performance of the typical GPU-only method where every tasks are executed on a deviceGPU by a factor of 1.5, 8.7 and 2.7, respectively.

Keywords : Parallel Programming, Heterogeneous-Computing, Computer Vision

OpenCL을 이용한 돈사 감시 응용의 효율적인 태스크 분배

김 진 성† · 최 윤 창†† · 김 재 학††† · 정 연 우†††† · 정 용 화††††† · 박 대 희†††††† · 김 학 재†††††††

요 약

다수의 태스크로 구성된 돈사 감시 응용은 내재된 데이터 병렬성을 활용하고 성능가속기를 사용하여 병렬 처리가 가능하다. 본 논문에서는 멀티코어 CPU와 매니코어 GPU로 구성된 이기종 컴퓨팅 플랫폼에서 돈사 감시 응용 수행 시 태스크 분배 방법을 제안한다. 즉, 각 태스크별로 OpenCL을 이용한 병렬 프로그램을 작성한 뒤, deviceCPU와 deviceGPU 각각에서 수행시켜 측정된 수행시간을 기준으로 가장 적합한 처리기를 결정한다. 제안 방법은 간단하지만 매우 효과적이고, CPU와 GPU로 구성된 이기종 컴퓨팅 플랫폼에서 다수의 태스크로 구성된 다른 응용을 병렬화하는 경우에도 적용될 수 있다. 실험 결과, 상이한 이기종 컴퓨팅 플랫폼에서 최적의 태스크 분배로 수행한 경우가 전체 태스크들을 deviceGPU에서 수행한 GPU-only 방법에 비교하여 각각 2.7배, 8.7배, 2.7배 성능 개선이 되었음을 확인하였다.

키워드 : 병렬 프로그래밍, 이기종 컴퓨팅, 컴퓨터 비전

1. 서 론

최근 고성능 컴퓨팅과 모바일 컴퓨팅 분야에서 성능가속기인 GPU(Graphics Processing Unit), DSP(Digital Signal Processing), FPGA(Field Programmable Gate Array) 등을 사용하여 성능을 개선하는 연구가 활발히 진행되고 있다. 그러나 널리 사용하는 GPU의 범용 병렬프로그래밍 프레임워크인 CUDA(Compute Unified Device Architecture)[1]의 경우 NVIDIA 계열의 제품에서만 동작하고, 다른 GPU나 CPU와 같은 마이크로프로세서에서는 지원을 하지 않는 단점이 있다. 이러한 이식성(portability)의 문제점을 해결한 것이 최근 발표된 OpenCL 표준[2, 3]이다.

많은 OpenCL 응용들은 성능가속기를 대상으로 병렬 처리를 하고 있고, 성능가속기 중에서도 GPU를 사용하여 여러 응용들을 병렬 처리한 연구가 발표되었다[4]. 그러나 주어진 태스크의 계산 특성과 수행 플랫폼 내 CPU-GPU의 상대적인 성능에 따라서 멀티코어 CPU를 이용한 병렬 처리

※ This research was supported by Leading Human Resource Training Program of Regional Neo Industry through NRF funded by MSIP (2016H1D5A1910730).
† 준 회 원 : 고려대학교 컴퓨터정보학과 학사과정
†† 준 회 원 : 고려대학교 컴퓨터정보학과 석사과정
††† 비 회 원 : 고려대학교 컴퓨터정보학과 학사과정
†††† 비 회 원 : 고려대학교 응용통계학과 학사과정
††††† 종신회원 : 고려대학교 컴퓨터정보학과 교수
†††††† 정 회 원 : 고려대학교 컴퓨터정보학과 교수
††††††† 비 회 원 : 클래스액트(주) 대표이사
Manuscript Received : July 11, 2017
Accepted : September 2, 2017
* Corresponding Author : Yongwha Chung(ychungy@korea.ac.kr)

가 매니코어 GPU를 이용한 병렬 처리보다 더 좋은 성능을 제공할 수도 있다. 또한, 태스크 스케쥴링에 대한 많은 기존 연구가 있지만, 대부분의 경우 분산 시스템 하에서 작업부하 분배(workload distribution)에 관한 시스템 레벨의 연구[5-7]이고, 본 논문에서 가정하는 병렬 시스템 하에서 태스크 분배에 관한 응용프로그램 레벨에서의 연구는 거의 발표가 되고 있지 않다.

예를 들어, 데이터 병렬성(data parallelim)을 이용하여 영상처리 응용의 수행시간을 개선하기 위한 연구로, 응용프로그램 레벨에서 OpenCL을 이용하여 CPU와 GPU의 자원을 동시에 활용하는 결과가 발표되었다[8, 9]. 그러나, 주어진 태스크의 계산 특성에 따라 CPU와 GPU간 작업부하 배분량은 병렬처리시 파라미터가 되고, 만약 응용이 수십개의 태스크로 구성된다면 각 태스크 별 실험 환경(즉, 플랫폼)의 성능에 따라 수십개의 파라미터를 설정해야하는 문제가 존재하여 많은 시간과 노력이 소요된다. 따라서 복잡한 응용을 병렬화하기 위해서는 각 태스크별로 파라미터 설정을 하지 않아도 되는 보다 현실적인 병렬처리 방법의 개발이 필요하다.

본 논문은 최종 목적인 24시간 돈사 자동 감시 응용을 위한 전처리 단계로써, 최종 응용은 “complete” computer vision system으로써 객체(돼지)를 탐지하는 low-level 단계에서부터 근접 객체(돼지)를 분리[10] 및 추적[11]하는 intermediate-level 단계와 객체(돼지)들의 행동 패턴을 분석[12]하는 high-level 단계까지의 과정으로 구성되어 있다. 24시간 자동 돈사 감시 응용을 위해서는 24시간 데이터를 24시간내에 처리해야 하며, 그러기 위해서는 앞서 설명한 모든 단계들(수십개의 태스크로 구성)은 통합되어 실시간 처리가 되어야 한다.

본 논문에서는 이러한 돈사 감시 응용중 영상처리에 해당하는 low-level 단계(8개의 태스크로 구성)를 OpenCL을 사용하여 병렬 처리하고자 한다. 통상의 영상처리 응용 병렬화는 앞서 설명한대로 데이터 병렬성을 이용하는 GPU의 구조와 잘 매치되어, GPU를 이용한 수많은 영상처리 연구 결과가 발표되었다[4]. 그러나 영상처리 응용에 속하는 유사한 태스크라도 데이터 종속성에 따라 상이한 계산 특성을 가지며, 매니코어 GPU보다 멀티코어 CPU에서 더 좋은 성능을 제공할 수 있다.

OpenCL은 이기종 컴퓨팅 환경에서 병렬 처리할 수 있는 프레임워크로, 태스크별로 deviceCPU 또는 deviceGPU를 활용하여 병렬 처리할 수 있다. 이러한 OpenCL의 특징을 이용하여, 본 논문에서는 상이한 계산 특성을 가지는 돈사 감시 응용[13, 14]의 각 태스크를 deviceCPU 또는 deviceGPU에 할당하여 수행시간을 단축하는 방법을 제안한다. 실험 결과, 제안 방법을 활용하면 전체 태스크들을 deviceGPU에서 수행한 경우(GPU-only)보다 수행시간을 단축할 수 있음을 확인하였다. 또한, 제안 방법은 각 태스크를 deviceCPU 또는 deviceGPU 중 하나에 분배하기 위하여 각 플랫폼 별로 한 번의 수행시간만 측정하면 되므로, 복잡한 응용을 병렬처리하는 경우 앞서 설명한 다수의 파라미터 설정 문제를 피할 수 있는 매우 현실적인 방법이라는 장점이 있다.

본 논문의 구성은 다음과 같다. 2장에서는 키넥트를 이용한 객체탐지 방법과 관련된 연구와 논문에서 사용된 병렬처리 프레임워크 중 하나인 OpenCL에 대하여 소개한다. 3장에서는 키넥트 깊이정보를 이용하여 누워있는 돼지탐지와 돼지 탐지의 태스크 분배 방법을 소개한다. 4장에서는 제안 방법을 이용한 실험 결과에 대해 분석하고, 마지막 5장에서는 결론을 맺는다.

2. 관련 연구

2.1 키넥트를 이용한 객체 탐지

통상 실내 환경에서 고정된 카메라로 획득한 영상에서의 전·배경 분리를 위해 배경 제거 기법을 사용한다. 객체들이 존재하지 않는 배경 영상과 입력된 영상에 배경 제거 기법을 적용한 후 Otsu 알고리즘[15]을 적용하여 전경과 배경이 분리된 이진 영상을 획득하는 방법을 이용한다. 그러나 저가 센서인 키넥트를 이용하여 획득한 깊이 정보에는 많은 노이즈가 존재한다. 또한, 같은 위치의 픽셀 깊이 정보 값이 시간에 따라 일관되지 않은 깊이 정보 값이 존재하는 문제가 있다. 예를 들어, 돼지가 없는 빈 돈방을 10분간 촬영한 후 각 픽셀의 최솟값/평균값/최댓값을 이용하여 생성된 배경 영상과 입력 영상에 배경 제거 기법을 사용한 후 Otsu 알고리즘을 적용하면 돼지를 정확히 탐지하지 못함을 확인할 수 있다(Fig. 1 참조).

(a) Minimum (b) Average (c) Maximum

Fig. 1. Results of Background Subtraction with Minimum, Average and Maximum values

또 다른 어려움으로 키넥트의 깊이 센서는 획득된 깊이 정보가 보장되는 최대 거리(키넥트2의 경우 4.5m)와 최대 화각(키넥트2의 경우 수평 70.6°/수직 60°)에 제약이 있고, 이 범위를 벗어나면 급격히 정확도가 떨어지는 문제가 있다. 규격상 최대 거리가 4.5m인 키넥트2[16]를 이용하여 천장으로부터 4m 이상 떨어진 바닥에 있는 돼지들을 깊이 정보 영상을 이용하여 탐지하는 경우, 깊이 정보 영상에 노이즈와 부정확한 깊이 정보 값의 한계가 존재한다.

2.2 OpenCL(Open Computing Language)

OpenCL은 개방형 범용 병렬 컴퓨팅 프레임워크이다. 많은 제조사와 소프트웨어 전문가의 견해를 반영하여 Khronos 그룹에서 제정된 OpenCL은 여러 제조사들의 협력을 통해서 등장하였다[2]. OpenCL 구현은 CPU, GPU를 포함하여 DSP

와 FPGA에 기반 한 플랫폼에서도 수행이 되기 때문에, 하나의 프로그램을 작성하여 여러 종류의 플랫폼에서 사용할 수 있는 장점이 있다. 이것은 응용프로그램 수행 중에 OpenCL 커널 프로그램이 생성되면서 디바이스에 맞는 이진 코드를 생성하기 때문이다. 또한, C99를 기반으로 한 언어인 OpenCL C로 커널 프로그램을 작성하기 때문에 프로그램을 작성하는데 있어서 C 언어와 유사한 특징도 가지고 있다.

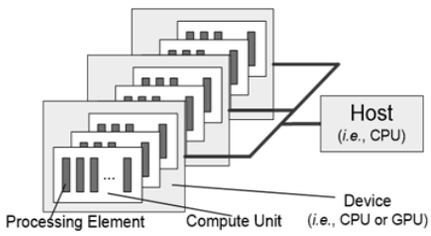


Fig. 2. Platform Model of OpenCL[2]

OpenCL 플랫폼(Fig. 2 참조)은 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. 만약 중간에 다른 디바이스에서 연산을 하려면, 커널에서 다른 커널로 직접적인 통신이 불가능하기 때문에 호스트를 경유하여 통신을 해야 한다.

3. 제안 방법

본 논문에서는 하나의 플랫폼에 하나의 호스트(host로 명기)와 두개의 디바이스(deviceCPU, deviceGPU로 명기)가 장착된 경우를 가정하고, 돈사 감시 응용은 마이크로소프트의 키넥트 카메라에서 획득한 깊이 정보를 이용하여 누워있는 돼지를 탐지하는 경우를 가정한다.

3.1 키넥트 깊이 정보를 이용한 누워있는 돼지 탐지

본 논문에서는 돈방 내 돼지들이 실제로 체감하는 온도를 자동으로 파악하고 조절하기 위하여 누워있는 돼지들의 몰려있는 정도[13]를 천장에 설치된 키넥트 카메라를 이용하여 판단하고자 한다. 특히, 야간에는 전등을 소등하기 때문에, 빛이 없는 환경에서도 탐지가 가능한 키넥트의 깊이 정보 영상을 이용한다[14].

1) 전·배경 분리

먼저 키넥트 카메라로부터 획득한 깊이 정보 영상을 보정하기 위해 돼지가 없는 빈 돈방을 10분간 촬영한 후 각 픽셀의 최빈값으로 배경 영상을 생성한 후, 깊이 영상의 크기를 가로/세로 각각 절반씩 줄인 Spatial Interpolation을 수행한다. 입력된 영상에서도 동일한 Spatial Interpolation을 적용하고, 추가로 30 frames per second(fps) 속도의 입력 영상을 10 fps 속도로 다운샘플하는 Temporal Interpolation을 적용한다(Task1). 이렇게 Spatiotemporal Interpolation된 배경 영상과 입력 영상에 배경 제거 기법을 적용하여 객체를 검출하고(Task2), 모폴로지 연산을 통하여 검출된 객체의 경계선 부분을 다듬는다(Task3). 다음으로 Connected Component Analysis를 통하여 각 객체의 면적을 계산한 후 계산된 면적이 한 마리 돼지의 면적보다 작으면 노이즈로 판단하여 제거하고, 두 마리 돼지의 면적보다 크면 붙어 있는 돼지로 판단한다(Task4).

2) 붙어있는 돼지의 윤곽선 보정

Task4의 결과로 획득한 붙어 있는 돼지는 Task2에서 실험적으로 측정된 파라미터를 사용하기 때문에 윤곽선이 부정확한 문제가 있다. 이를 해결하기 위하여 Spatiotemporal Interpolation(Task1) 영상 중 붙어 있는 돼지 영역에 대하여 윤곽선을 보정한다. 먼저 명암 분포를 균일하게 만들어 명암 대비를 증가시키는 CLAHE(Contrast Limited Adaptive Histogram Equalization)[17]을 사용하여 객체와 배경의 대비를 뚜렷하게 만든 후(Task5), Median Blur를 적용하여 윤곽선을 보정한다(Task6). 이후 Otsu 알고리즘[15]을 적용하여 이진화 시킨 후(Task7), Median Blur를 사용하여 다시 윤곽선을 보정한다. 마지막으로 Task4의 결과와 Task7의 결과를 Logical AND 연산을 수행함으로써 돼지의 윤곽선을 보정한다(Task8). 이 결과를 바탕으로 돼지들의 몰려있는 정도를 측정함으로써 돈방의 온도를 자동적으로 조절한다. 키넥트 깊이 정보를 이용한 누워있는 돼지 탐지 태스크들의 흐름도를 정리하면 Fig. 3과 같다.

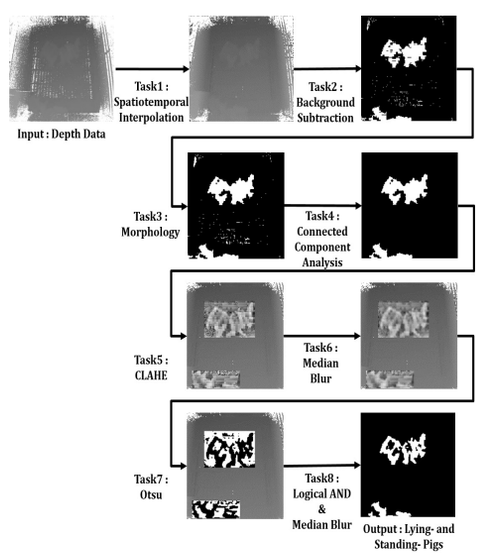


Fig. 3. Task Flow Chart for Lying-Pigs Detection

3.2 돼지 탐지 태스크들의 병렬 처리

3.1의 태스크들은 최종 목적인 24시간 돼지 모니터링 시스템의 low-level 단계로, 이후에 intermediate-level과 high-level 태스크들이 존재한다. 따라서 시스템의 실시간 처리를 위해

각 단계들의 수행시간이 최소가 되어야하기 때문에 3.1의 태스크들을 OpenCL을 사용하여 병렬 처리함으로써 수행시간을 단축한다.

먼저, Spatiotemporal Interpolation(Task1), Background Subtraction(Task2), Morphology(Task3), Median Blur(Task6)은 병렬 처리함에 있어 데이터간 종속성이 존재하지 않으므로 각각 픽셀에 대하여 독립적으로 계산이 가능하다. 그러므로 Task1은 mask가 2, Task2는 mask가 1, Task3 및 Task6은 mask가 3을 가지는 stencil computation으로 병렬처리를 할 수 있다(Fig. 4 참고).

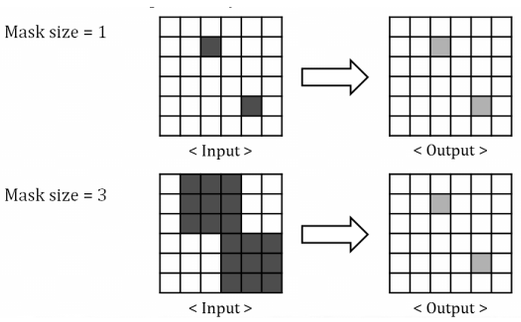


Fig. 4. Stencil Computation

하지만 Connected Component Analysis(Task4), CLAHE(Task5), Otsu(Task7)는 데이터간 종속성이 존재하기 때문에 Fig. 4와 같은 stencil computation 형식이 아니다. 본 논문에서는 Connected Component Analysis를 병렬 처리하기 위하여, 우선 라벨링[18]과 동시에 각 라벨의 면적을 계산한 후, 일정 면적이하인 라벨을 동시에 제거하는 방법을 사용한다. 또한 CLAHE는 영상을 8×8로 나누어 각각 히스토그램을 계산하여 히스토그램 평활화를 한다. 마지막 Otsu 알고리즘은 히스토그램을 계산하고, 순차처리로 분산과 임계값을 계산한 후, 그 임계값으로 이진화 시키는 과정으로 병렬 처리한다.

3.3 돼지 탐지의 태스크 분배

본 논문에서는 태스크를 스케쥴링의 최소 단위로 가정하며, 3.1에서 언급한 8가지 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 플랫폼 내 CPU-GPU의 상대적인 성능에 따라 각 태스크별로 적합한 처리기가 있고, 각 태스크의 계산에 필요한 데이터의 통신 오버헤드도 고려해야 한다.

본 논문에서는 돼지 탐지의 8가지 태스크가 하나의 deviceCPU와 하나의 deviceGPU로 구성된 이기종 플랫폼에서 수행될 수 있는 시나리오를 DAG 형태의 구현 그래프(implementation graph)로 표현(Fig. 5 참조)하고, 이를 활용하여 전체 비용이 최소가 되는 태스크 분배 알고리즘을 제안한다. 특히, deviceCPU와 deviceGPU간에는 직접 데이터 통신이 불가능하기 때문에 호스트를 경유하여 통신한다. 또한, 각 태스크별로 적절한 디바이스 결정을 편리하게 하기 위하여 통신 오버헤드를 호스트에서 디바이스로 갈 때(“host-to-device + device-to-host”) 디바이스에서 호스트로 올 때의 통신 오버헤드를 포함시킨다. 즉, Fig. 5에서 host에서 CPU_device로 통신하는 통신 오버헤드는 “commCPU”라고 명기하고 “host-to-deviceCPU + deviceCPU-to-host”를 의미한다. 또한 host에서 GPU_device로 통신하는 통신 오버헤드는 “commGPU”라고 명기하고 “host-to-deviceGPU + deviceGPU-to-host”를 의미한다.

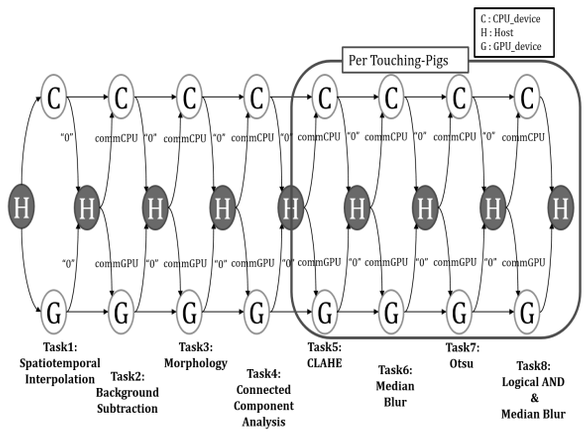


Fig. 5. Implementation Graph for Lying-Pigs Detection

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 통신 오버헤드(Comm)를 포함한 speedup이 큰 것을 의미한다. 즉, 각 태스크별로 deviceCPU와 deviceGPU에서의 순차 처리 시간(Comp)과 병렬 speedup을 곱하여 Efficiency라는 척도로 표현하고, 이 척도를 기준으로 가장 파급효과가 큰 것을 먼저 선택하는 방법으로 8가지 태스크에 대한 적절한 처리기를 결정한다. Fig. 6에 제안 알고리즘을 정리하였고, n개의 태스크에 대한

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 ~ 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 ~ n
$Initial_Distribution[i]$ = max($Efficiency[i,$ deviceCPU], $Efficiency[i,$ deviceGPU])

Step3. Determine Final_Distribution Matrix [n-tasks]
For i = 1 ~ 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 $Comm[j-1, k] = Comm[j+1, k] = \Phi;$

Fig. 6. Algorithm of Task Distribution

제안 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.

4. 실험 결과

본 실험을 위해 세 개의 플랫폼에서 실험을 진행하였다. 플랫폼1의 실험 환경은 3.50GHz Intel ® CoreTM i5-4690 CPU, GeForce GTX 750, 8GB RAM으로 구성되어있으며 플랫폼2의 실험 환경은 2.67GHz Intel ® CoreTM i5 CPU 750, GeForce GTS 250, 8GB RAM으로 구성되어있다. 마지막으로, 플랫폼3의 실험 환경은 3.40GHz Intel ® CoreTM i7 CPU 4770, GeForce GTX 750, 8GB RAM으로 구성되어있다. 각 태스크를 OpenCL 코드로 병렬화하였고, 각 플랫폼에서 1000장의 영상에 대하여 측정한 태스크별 평균수행시간을 정리하면 Table 1~3과 같다.

예를 들어, 플랫폼1에서 Task1의 CPU 병렬 처리 시간(0.311ms)이 GPU 병렬 처리 시간(0.083ms) 보다 빠르지 않음을 확인할 수 있다. 반면, 플랫폼2에서는 캐쉬 등의 효과로 Task1의 4-코어 CPU 병렬 처리 시간(0.290ms)이 1-코어 순차처리 시간(8.518ms)의 1/4 보다 적으며 GPU 병렬 처리 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 중간 값들을 병합하는 과정의 시간을 단축시키기 위한 방법[17]을 적용하였음에도 불구하고, 플랫폼에 상관없이 Task1~3과 비교하면 GPU의 처리시간이 느린 특성이 있다. 이는 Connected Component Analysis의 계산 특성상 멀티코어 CPU보다 매니코어 GPU에서 중간값을 병합하는 단계수가 훨씬 많기 때문이다. 또한 Task5와 Task7에서도 GPU의 처리시간이 CPU의 처리시간에 비하여 많이 느린 특성이 있다. 이는 모든 픽셀에 대하여 연산을 수행하는 것이 아니라 붙어 있는 돼지 부분에 대해서만 연산을 수행하여, 매니코어 GPU에서는 일부 코어만을 사용하기 때문이다.

또한, 제안된 알고리즘으로 각 플랫폼별 태스크를 분배한 결과를 나타내면 Fig. 7~9와 같고, 이렇게 분배된 경우와 전체 태스크들을 deviceGPU에서 수행한 경우(GPU-only로 명기)의 Speed up을 비교하면 Fig. 10과 같다. 즉, 전체 태스크들을 deviceGPU에서만 수행한 GPU-only 방법과 비교하여 제안 방법은 플랫폼1에서 2.7배, 플랫폼2에서 8.7배, 플랫폼3에서는 2.7배 향상된 성능을 확인하였다. 이렇게 성능 차이가 나는 이유는 앞서 설명한 Task4, Task5, Task7의 계산 특성이 CPU에 비하여 GPU와는 잘 매치되지 않는데, 제안 방법에서는 이를 반영하여 Task4~7을 CPU에서 수행하지만 GPU-only 방법에서는 그렇지 못하기 때문이다.

Table 1. Execution Time of each Task on Platform1 (msec)

Platform1				
		Parallel		Sequencial
		deviceGPU	deviceCPU	host(CPU)
Communication Overhead	host-to-device	0.063	0.005	
	device-to-host	0.113	0.036	
Execution Time	Task1	0.083	0.311	4.342
	Task2	0.086	0.150	0.188
	Task3	0.123	0.326	0.417
	Task4	1.421	1.394	5.345
	Task5	2.994	0.451	0.637
	Task6	0.144	0.162	0.281
	Task7	3.660	0.389	0.613
	Task8	0.091	0.156	0.319
	Total	8.778	3.425	12.142

Table 2. Execution Time of each Task on Platform2 (msec)

Platform2				
		Parallel		Sequencial
		deviceGPU	deviceCPU	host(CPU)
Communication Overhead	host-to-device	0.195	0.025	
	device-to-host	0.188	0.064	
Execution Time	Task1	0.430	0.290	8.518
	Task2	0.463	0.198	0.426
	Task3	0.505	0.235	0.772
	Task4	9.505	2.476	9.779
	Task5	16.999	0.730	1.500
	Task6	0.698	0.277	0.601
	Task7	15.506	0.618	0.987
	Task8	0.620	0.246	0.676
	Total	45.109	5.169	23.259

Table 3. Execution Time of each Task on Platform3 (msec)

Platform3				
		Parallel		Sequencial
		deviceGPU	deviceCPU	host(CPU)
Communication Overhead	host-to-device	0.075	0.013	
	device-to-host	0.087	0.025	
Execution Time	Task1	0.088	0.289	4.251
	Task2	0.088	0.090	0.183
	Task3	0.139	0.198	0.414
	Task4	1.329	1.525	4.586
	Task5	2.410	0.339	0.711
	Task6	0.109	0.124	0.287
	Task7	3.647	0.289	0.662
	Task8	0.093	0.115	0.330
	Total	8.065	3.007	11.424

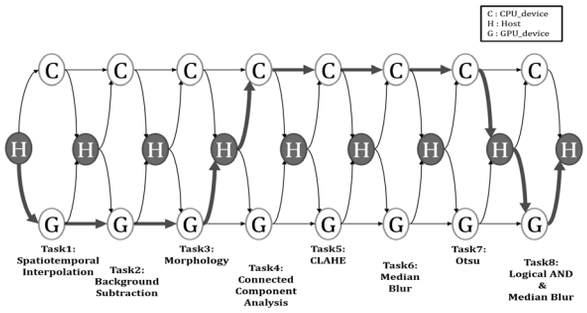


Fig. 7. Result of Task Distribution on Platform1

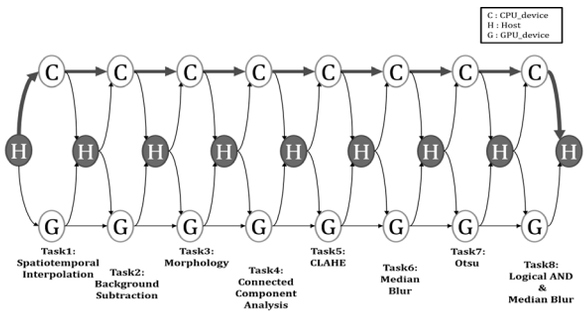


Fig. 8. Result of Task Distribution on Platform2

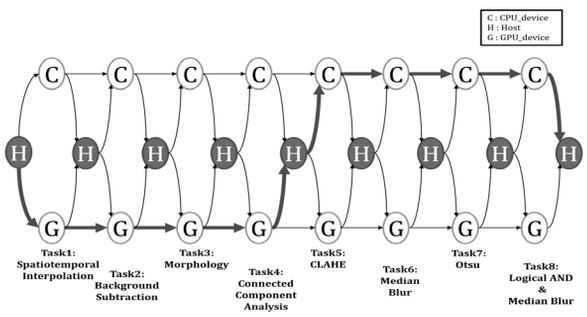


Fig. 9. Result of Task Distribution on Platform3

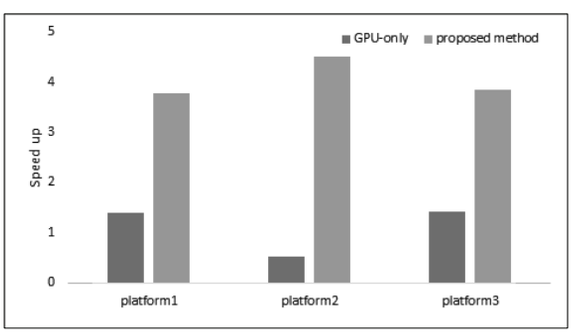
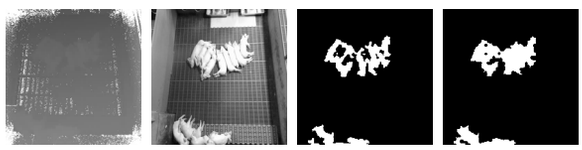


Fig. 10. Performance Comparison of each Task Distribution

마지막으로, 병렬 프로그래밍과 순차처리 프로그래밍의 돼지 탐지 결과를 비교하면 Fig. 11과 같다. Fig. 11(a)에 나타낸바와 같이 키넥트의 깊이 영상은 육안으로 돼지를 확인하기 어려워 Fig. 11(b)에 상응하는 컬러 영상을 참고용으로 보여주었고, 순차처리 탐지 결과(Fig. 11(c) 참조)와 병렬처리 탐지 결과(Fig. 11(d) 참조)가 유사함을 확인할 수 있다. 보다 정확한 비교를 위하여 모든 돼지들이 움직이는 영상, 절반의 돼지들이 움직이는 영상, 거의 모든 돼지들이 움직이지 않는 영상을 사용을 각 1시간씩 총 3시간의 영상을 사용하였고, ground truth와 비교하였다. TP(True Positive), TN(True Negative), FP(False Positive), FN(False Negative)로 표현되는 정확도((TP+TN)/(TP+TN+FP+FN))는 픽셀 단위 정확도와 마리 수 기준 정확도로 구분하여 측정하였다. 측정 결과, 순차처리 프로그래밍의 픽셀 단위 정확도는 88.8%(마리 수 기준 정확도 99.8%)이고, 병렬 프로그래밍의 픽셀 단위 정확도는 78.8%(마리 수 기준 정확도 99.8%)이었다. 즉, 픽셀 단위 정확도에서는 일부 차이가 있으나, 마리 수 기준 정확도에서는 순차 프로그래밍과 병렬 프로그래밍의 결과가 차이가 없음(즉, 두 경우 모두 있는 돼지를 누락하거나 없는 돼지를 반대로 탐지하는 경우가 거의 없음)을 확인할 수 있다.



(a) Depth (b) Color (c) Sequential (d) Parallel

Fig. 11. Input Images and Detection Outputs with Sequential and Parallel Executions

5. 결 론

본 논문에서는 이기종 컴퓨팅 환경에서 8개의 태스크로 구성된 돈사 감시 응용의 병렬처리를 위하여 OpenCL을 이용하여 CPU-GPU간 태스크를 분배하는 방법을 제안하였다. 먼저 돈사 감시 응용에서 OpenCL로 작성된 각 태스크 별로 deviceCPU와 deviceGPU에서 수행하여 시간을 측정하였다. 측정된 결과를 바탕으로 그리디 알고리즘을 사용하여 가장 적합한 device에 태스크를 분배하였다.

실험 결과, 태스크의 계산 특성에 따라 적합한 처리기가 있고, 플랫폼 내 CPU-GPU의 상대적인 성능에 따라서도 태스크 분배가 달라져야 함을 확인하였다. 즉, 전체 태스크들을 deviceGPU에서 수행한 방법과 비교하여 제안 방법은 플랫폼1에서 2.7배, 플랫폼2에서 8.7배, 플랫폼3에서는 2.7배 향상된 성능을 확인하였다.

또한 이기종 컴퓨팅 환경에서 개별 태스크를 효과적으로 병렬처리하는 연구들은 존재하지만, 본 논문에서 가정하는 수십개의 태스크들로 구성된 복잡한 시스템 전체를 이기종 컴퓨팅 환경에서 병렬처리하는 효과적인 방법은 보고된바 없다. 따라서, 다음 단계로 개발할 intermediate-level vision의 태스크[10,11]들과 high-level vision의 태스크[12]들을 병렬처리할때도 그대로 적용될 수 있으며, pipeline 기법을 적용하여 CPU와 GPU를 동시에 활용하는 것도 향후 연구로 고려하고 있다.

References

[1] J. Sanders and E. Kandrot, *CUDA by Example*, Addison Wesley, 2011.

[2] J. Stone, D. Gohara, and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science and Engineering*, Vol.12, No.3, pp.66-73, 2010.

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

[4] R. Couturier, *Designing Scientific Applications on GPUs*, CRC Press., 2013.

[5] W. Wang, Y. Chang, W. Lo, and Y. Lee, "Adaptive Scheduling for Parallel Tasks with QoS Satisfaction for Hybrid Cloud Environments," *J. of Supercomputing*, Vol.66, No.2, pp. 783-811, 2013.

[6] Y. Hao, M. Xia, N. Wen, R. Hou, H. Deng, L. Wang, and Q. Wang, "Parallel Task Scheduling under Multi-Clouds," *TIIS*, Vol.11, No.1, pp.39-60, 2017.

[7] F. Ramezani, J. Lu, J. Taheri, and F. Hussain, "Evolutionary Algorithm-based Multi-Objective Task Scheduling Optimization Model in Cloud Environments," *WWW-Internet & Web Info. Sys.*, Vol.18, No.6, pp.1737-1757, 2015.

[8] H. Kim, J. Sa, D. Choi, H. Kim, S. Lee, Y. Chung, and D. Park, "Parallel & Distributed Computing : Efficient Workload Distribution of Photomosaic Using OpenCL into a Heterogeneous Computing Environment," *KIPS Tr. Transaction on Computer and Communication Systems*, Vol.4, No.8, pp. 245-252. 2015.

[9] S. Lee, H. Kim, D. Park, Y. Chung, and T. Jeong, "CPU-GPU Hybrid Computing for Feature Extraction from Video Frame," *IEICE Electronics Express*, Vol.11, No.22, pp.1-8, 2014.

[10] J. Sa, S. Han, S. Lee, H. Kim, S. Lee, Y. Chung, and D. Park, "Image Segmentation of Adjoining Pigs Using Spatio-Temporal Information," *KIPS Tr. Software and Data Eng.*, Vol.4, No.10, pp.473-478, 2015.

[11] S. Zuo, L. Jin, Y, Chung, and D. Park, "An Index Algorithm for Tracking Pigs in Pigsty," in *Proc. of International Conference on Information Technology and Management Science*, pp.797-803, 2014.

[12] J. Lee, L. Jin, D. Park, and Y. Chung, "Automatic Recognition of Aggressive Behavior in Pigs by using a Kinect Depth Sensor," *Sensors*, Vol.16, No.5, 2016.

[13] B. Shao and H. Xin, "A Real-Time Computer Vision Assessment and Control of Thermal Comfort for Group-Housed Pigs," *Computers and Electronics in Agriculture*, Vol.62, No.1, pp.15-21, 2008.

[14] J. Kim, Y. Choi, J. Sa, M. Ju, Y. Chung, D. Park, and H. Kim, "Pig's Room Background Removal using Texture Information," in *Proc. of Smart Media Fall Conference*, 2016.

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

[16] J. Han, L. Shao, D. Xu, and J. Shotton, "Enhanced computer vision with microsoft kinect sensor: A review," *IEEE Transactions on Cybernetics*, Vol.43, No.5, pp.1318-1334, 2013.

[17] A. M. Reza, "Realization of the contrast limited adaptive histogram equalization(CLAHE) for real-time image enhancement," *The Journal of VLSI Signal Processing*, Vol.38, Issue 1, pp.35-44, 2004.

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

김 진 성

e-mail : skykeeop@korea.ac.kr

2012년~현 재 고려대학교 컴퓨터정보학과 학사과정

관심분야 : 병렬처리, 영상처리

최 윤 창

e-mail : ycc4477@korea.ac.kr

2017년 고려대학교 컴퓨터정보학과(학사)

2017년~현 재 고려대학교 컴퓨터정보학과 석사과정

관심분야 : 병렬처리, 영상처리

김 재 학

e-mail : bbq3438@korea.ac.kr

2012년~현 재 고려대학교 컴퓨터정보학과 학사과정

관심분야 : 영상처리

정 연 우

e-mail : william0516@korea.ac.kr

2013년~현 재 고려대학교 응용통계학과 학사과정

관심분야 : 병렬처리, 영상처리

정 용 화

e-mail : ychungy@korea.ac.kr
1984년 한양대학교 전자통신공학과(학사)
1986년 한양대학교 전자통신공학과(석사)
1997년 U. of Southern California(박사)
1986년~2003년 한국전자통신연구원 생체인식기술연구팀 팀장
2003년~현 재 고려대학교 컴퓨터정보학과 교수
관심분야 : 병렬처리, 영상처리, 축산 IT

박 대 희

e-mail : dhpark@korea.ac.kr
1982년 고려대학교 수학과(학사)
1984년 고려대학교 수학과(석사)
1989년 플로리다 주립대학 전산학과(석사)
1992년 플로리다 주립대학 전산학과(박사)
1993년~현 재 고려대학교 컴퓨터정보학과 교수
관심분야 : 데이터마이닝, 인공지능, 축산 IT

김 학 재

e-mail : krunivs@gmail.com
2007년 고려대학교 전산학과(학사)
2009년 고려대학교 전산학과(석사)
2009년~2012년 한국전자통신연구원 서버플랫폼연구팀 연구원
2013년~2013년 ㈜엔텍 기업부설연구소 연구소장
2014년~현 재 ㈜클래스액트 기업부설연구소 연구소장
2014년~현 재 ㈜클래스액트 대표이사
관심분야 : 병렬처리, 영상처리, 고성능컴퓨팅, 축산 IT