CPU-GPU heterogeneous implementations of depth-based foreground detection

Younchang Choi, Jinseong Kim, Jaehak Kim, Yongwha Chung[a)], **Daihee Park, and Sungju Lee**

Dept. of Computer Convergence Software, Korea University, Sejong, KS002, Korea

a) *ychungy@korea.ac.kr*

Abstract: Video sensor data has been widely used in automatic surveillance applications. In this study, we present a method that automatically detects the foreground by using depth information. For real-time implementation, we propose a means of reducing the execution time by applying parallel processing techniques. In general, most parallel processing techniques have been used to parallelize each specific task efficiently. In this study, we consider a practical method to parallelize an entire system consisting of several tasks (i.e., *low-level* and *intermediate-level* computer vision tasks with different computational characteristics) by balancing the total workload between CPU and GPU. Experimental results with a pig monitoring application reveal that the proposed method can automatically detect the foreground using CPU-GPU heterogeneous computing platforms in real time, regardless of the relative performance between the CPU and GPU.

Keywords: computer vision, parallel processing, agriculture IT

Classification: Integrated circuits

References

[1] S. Lee, *et al.*: "Real-time processing for intelligent-surveillance applications," IEICE Electron. Express **14** (2017) 20170227 (DOI: 10.1587/elex.14.20170227).

[2] M. Hirano and S. Ohnuki: "Fast computation for electromagnetic scattering problems using a heterogeneous multi-core processor," IEICE Electron. Express **8** (2011) 1330 (DOI: 10.1587/elex.8.1330).

[3] L. Tangjittaweechai, *et al.*: "Fast bidirectional shortest path on GPU," IEICE Electron. Express **13** (2016) 20160036 (DOI: 10.1587/elex.13.20160036).

[4] J. Stone, *et al.*: "OpenCL: A parallel programming standard for heterogeneous computing systems," Comput. Sci. Eng. **12** (2010) 66 (DOI: 10.1109/MCSE.2010.69).

[5] B. Shao and H. Xin: "A real–time computer vision assessment and control of thermal comfort for group-housed pigs," Comput. Electron. Agric. **62** (2008) 15 (DOI: 10.1016/j.compag.2007.09.006).

[6] M. Kawakita, *et al.*: "Real-time three-dimensional video image composition by depth information," IEICE Electron. Express **1** (2004) 237 (DOI: 10.1587/elex.1.237).

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018

[7] S. Lee, *et al.*: "CPU-GPU hybrid computing for feature extraction from video stream," IEICE Electron. Express **11** (2014) 20140932 (DOI: 10.1587/elex.11.20140932).

[8] M. Poongothai, *et al.*: "A heuristic based real time task assignment algorithm for heterogeneous multiprocessors," IEICE Electron. Express **11** (2014) 20130975 (DOI: 10.1587/elex.11.20130975).

[9] M. Hirano and S. Ohnuki: "Fast computation for electromagnetic scattering problems using a heterogeneous multi-core processor," IEICE Electron. Express **8** (2011) 1330 (DOI: 10.1587/elex.8.1330).

[10] I.-Y. Jung and C.-S. Jeong: "Parallel connected-component labeling algorithm for GPGPU applications," Communications and Information Technologies (ISCIT) (2010) 1149 (DOI: 10.1109/ISCIT.2010.5665161).

[11] J. Lee, *et al.*: "Automatic recognition of aggressive behaviors in pigs using a Kinect depth sensor," Sensors **16** (2016) 631 (DOI: 10.3390/s16050631).

[12] A. M. Reza: "Realization of the contrast limited adaptive histogram equalization (CLAHE) for real-time image enhancement," J. VLSI Signal Process. **38** (2004) 35 (DOI: 10.1023/B:VLSI.0000028532.53893.82).

1 Introduction

Foreground detection is the first step in many image-based applications. In particular, foreground detection should be performed in real-time for video-based surveillance applications [1]. To reduce the execution time of a specific vision task, we can apply parallel processing techniques, such as a *data decomposition* technique, with either a multicore CPU [2] or manycore GPU [3] by assuming a typical PC has one CPU chip and one GPU chip. However, many practical computer vision applications, such as foreground detection, consist of several parallelized tasks whose computational characteristics are different. For example, *low-level* vision tasks are based on regularly-structured computations (i.e., match well with GPU), whereas *intermediate-level* vision tasks are based on irregularly-structured computations (i.e., do not match well with GPU). Additionally, OpenCL [4] has been released to provide a standard parallel programming environment across various computing devices, such as multicore CPU (denoted as deviceCPU) and manycore GPU (denoted as deviceGPU). That is, we need to consider the computational characteristics of each vision task to determine a computing device for each task, and then, parallelize the entire computer vision system with OpenCL to provide the code compatibility across computing devices.

In this study, we consider automatic temperature control of a pig room by employing a video sensor. We focus specifically on foreground (i.e., pig) detection to assess and control the thermal comfort of weaning pigs (21–28 days old) because of their weak immunity [5]. Recently, inexpensive depth sensors, such as Microsoft Kinect, have been released and used by the computer vision community [6]. Therefore, we use this depth sensor to solve the problem of assessing a sleeping pig's thermal comfort at night. First, we apply some noise reduction techniques because Kinect produces considerable noise. Then, we perform background subtraction and binarization to detect the pigs in a pig room. Note that, the target application of this study is not like typical scientific applications such as large-scale

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018

weather prediction based on data decomposition, but real-time video analysis where several tasks having different computational characteristics are involved and parallelizing each task with the data decomposition technique only may not provide acceptable performance. That is, a method to reduce the execution times of regularly-structured computations from 100 hours to 1 hour may be different from a method to reduce the execution times of both regularly- and irregularly-structured computations from 10 msec to 1 msec.

For real-time implementation of the video-based analysis application, we propose a means of reducing execution time by applying parallel processing techniques. However, the goal of this study is not to optimize a specific vision task. Instead, we optimize the entire vision application (i.e., depth-based foreground detection) with *data decomposition*-based parallel solutions for each vision task such as [7, 8, 9]. By measuring the speed of each task in a computing device (i.e., deviceCPU or deviceGPU) through the OpenCL-based implementation, however, we can determine an appropriate computing device for each parallelized task whose computational characteristics are different. We can then reduce the total execution time further by balancing the execution times of the deviceCPU and deviceGPU with the proposed pipeline scheduling algorithm. Note that many data decomposition-based parallel methods for a regularly-structured task have been reported on a CPU-GPU heterogeneous computing platform [7, 8, 9]. To the best of our knowledge, however, this is the first study that detects pigs at night on CPU-GPU heterogeneous computing platforms with a pipelined execution of the foreground detection consisting of several regularly- and irregularly-structured tasks.

2 Proposed method

Ideally, foreground (i.e., pig) detection should be very simple using a depth sensor because depth data are unaffected by illumination changes and color. However, in practical use, many problems remain in differentiating between background and foreground. For example, in the pig room we monitored, the floor is a plastic slat with holes for excreta treatment. With a time-of-flight (ToF) based Kinect Version 2 sensor, this floor structure produces many holes (i.e., noise). Furthermore, the Kinect sensor has maximum distances (i.e., 4.5 m) and fields-of-view (i.e., 70.6° horizontally and 60° vertically) at which it can detect depth. Therefore, the depth value returned from a wall may have unreliable values.

To address the noise and unreliable values, all depth values below a certain threshold are discarded. Moreover, both spatial and temporal interpolations are applied by reducing the resolution size and frame rate (denoted as Task1). Following spatiotemporal interpolation, the background subtraction pixel values, greater than a threshold, are regarded as foreground (denoted as Task2). Morphological operators are then applied to smooth the detected foreground (denoted as Task3). Finally, each area of the Connected Component Analysis (CCA) which is greater than the area of a single pig (i.e., area of touching pigs) is added to assess pig thermal comfort automatically (denoted as Task4).

Note that, the data dependencies of Task1, Task2, and Task3 (i.e., *low-level* computer vision tasks) are "local and regularly-structured", whereas the data

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018

dependencies of Task4 (i.e., *intermediate-level* computer vision task) are “global and irregularly-structured”. Therefore, we should consider the computational characteristics of each task, the architectural characteristics of each device, and the relative performance between the devices, simultaneously. Especially, the data transmitted between deviceCPU and deviceGPU need to go through a host (i.e., CPU) in OpenCL. If Task4 is executed on both deviceCPU and deviceGPU simultaneously, then we need to satisfy the complicated data dependency shown in Fig. 1 with OpenCL (i.e., both the number of communications between deviceCPU and deviceGPU and the data to be transmitted are not determined at compile time). Furthermore, the execution time of Task4 on deviceGPU may be much longer than that of deviceCPU, depending on a given platform (see Fig. 4 in Section 3). In this case, we need to avoid the unnecessary programming burden in order to satisfy the complicated data dependency and to determine the workload distributed to each device, which may not improve the actual performance. In this study, we consider a practical method to parallelize an entire system consisting of several tasks with different computational characteristics by balancing the performance improvement and programming burden.

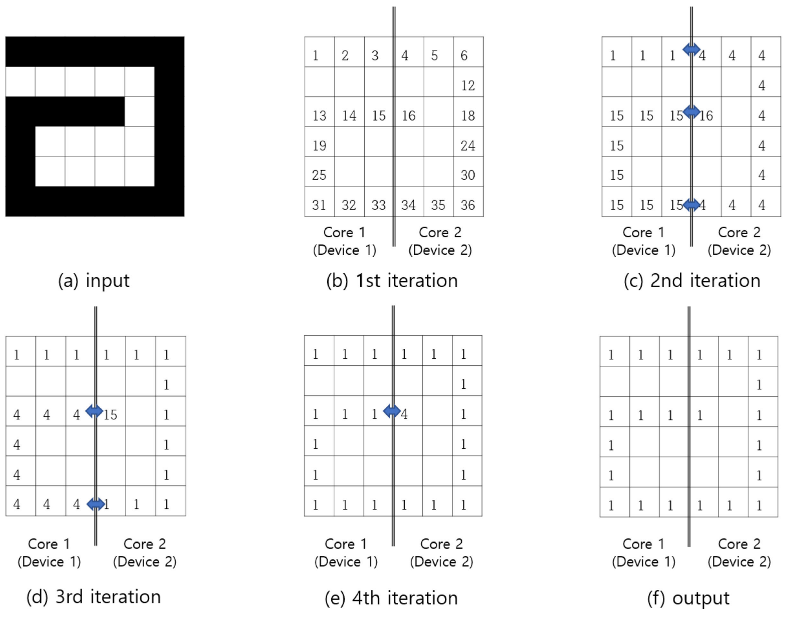

Fig. 1. Data dependency of Task4 in applying a typical data decomposition method

In our OpenCL-based system, each task can be assigned to a device (i.e., deviceCPU or deviceGPU). However, as shown in Fig. 2, we must consider both the communication and computation times to determine the shortest path from the host (start) node to the host (end) node. Specifically, by partitioning the entire tasks into two parts such that the execution time of deviceCPU is balanced with that of deviceGPU, we can maximally utilize the computational resources of deviceCPU and deviceGPU in the thermal comfort assessment system, regardless of the relative performance between deviceCPU and deviceGPU of a given platform.

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018

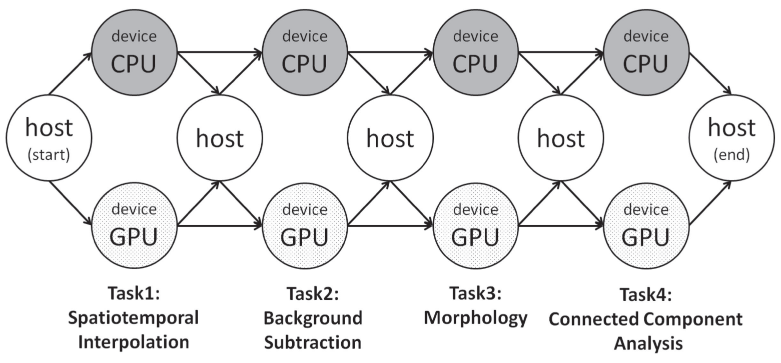

Fig. 2. Possible implementation scenarios in a CPU-GPU heterogeneous computing platform

In particular, we assume that the communication between deviceCPU and deviceGPU goes through the host, as transferring data directly between the devices is impossible. In addition, the greedy rule that determines the task distribution for each platform is "minimizing the total execution time such that the execution times of deviceCPU and deviceGPU are balanced." The details of the pipeline scheduling algorithm are shown in Fig. 3.

```
Input    deviceCPU_time[n tasks], deviceGPU_time[n tasks]
Output   Boundary_index, Task_seq (0 = CPU first, 1= GPU first)

Algorithm
Step 1: Initialize Boundary_index, n, sum1, sum2, sum3, sum4, tmp_time
        Boundary_index = 0; n = number of tasks;
        sum1 = commCPU + sum of deviceCPU_time[n tasks];
        sum2 = commGPU + sum of deviceGPU_time[n tasks];
        sum3 = commCPU; sum4 = commGPU;
        tmp_time = MIN(MAX(sum1, sum4), MAX(sum2, sum3));
Step 2: Determine the Boundary_index by increasing the index of tasks
        for i=1 to n {
            sum1 = sum1 -  deviceCPU_time[i-1];
            sum2 = sum2 -  deviceGPU_time[i-1];
            sum3 = sum3 + deviceCPU_time[i-1];
            sum4 = sum4 + deviceGPU_time[i-1];
            if (tmp_time > MIN(MAX(sum1, sum4), MAX(sum2, sum3))) {
              Boundary_index = i;
              if (MAX(sum1, sum4) < MAX(sum2, sum3))
                  Task_seq = 1;
              else
                  Task_seq = 0;
            tmp_time = MIN(MAX(sum1, sum4), MAX(sum2, sum3));
            }
        }
Step 3: Determine the task distribution
       if (Task_seq = 0)
         Tasks whose index are under the Boundary_index in deviceCPU
         and over the Boundary_index in deviceGPU run simultaneously
       else
         Same as the above except the sequence of workers
```

Fig. 3. Pipeline scheduling algorithm.

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018

3 Experimental results

The camera was located 4 m above the floor to monitor a pig room measuring 4 m × 3 m, and 13 weaning pigs were present in the room. In our experiments, we set the resolution size to 512 × 424 pixels and frame rate to 30 frames per second (fps) to handle the noise and unreliable values. From the 512 × 424 pixel images, we masked out certain regions where pigs could not be detected. A background image was computed as a pixel-by-pixel average of a 10-min video sequence without pigs.

First, we straightforwardly parallelized each task using a data decomposition technique. The OpenCL-based code was developed by three students within two weeks (i.e., programming burden was small). Then, we measured the execution time of each task in each device of a given platform. Based on the execution profile of the given platform, the pipeline scheduling algorithm could determine an appropriate device for each task such that the execution times of deviceCPU and deviceGPU were maximally balanced. That is, we need to determine the workload distributed to deviceCPU and deviceGPU for each "platform", not for each "task". This kind of workload determination becomes very effective when the number of tasks of a given monitoring system is increased.

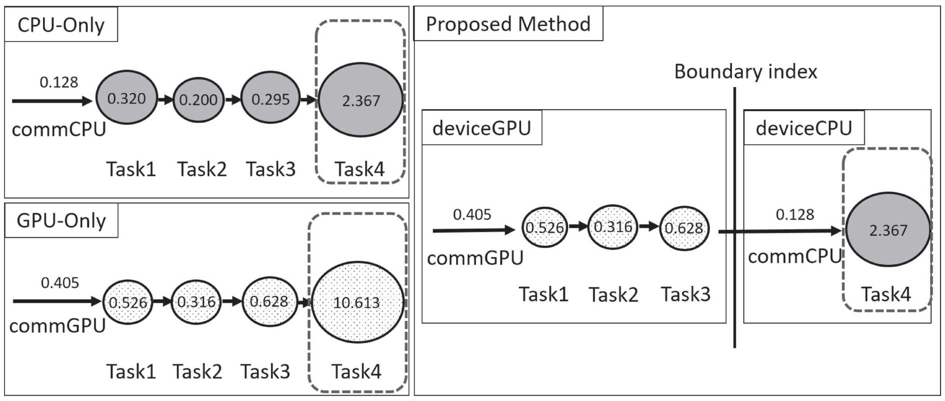

Fig. 4. Illustration of the pipeline scheduling algorithm on platform1.

To evaluate the proposed method, we conducted our experiment using three platforms. The first platform (denoted as platform1) consisted of a 2.66 GHz Intel ® Core™ i5-750 quad-core processor, GeForce GTS 250 with 128 cores, and 8 GB RAM. The second platform (denoted as platform2) consisted of a 3.60 GHz Intel ® Core™ i3-4160 dual-core processor, GeForce GTX 750 Ti with 640 cores, and 4 GB RAM. The third platform (denoted as platform3) consisted of a 3.40 GHz Intel ® Core™ i7-4770 quad-core processor, GeForce GTX 750 with 512 cores, and 8 GB RAM. Based on the relative performance between the deviceCPU and deviceGPU, we considered platform1, platform2, and platform3 as a CPU-higher, GPU-higher, and comparative platform, respectively. For further explanation, the per-video-frame execution times (unit: msec) of deviceCPU and deviceGPU on platform1 are shown in Fig. 4, and speedups of the data decomposition-based CPU-only (i.e., all tasks were executed on deviceCPU), GPU-only (i.e., all tasks were executed on deviceGPU), CPU-GPU (i.e., workload was

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018

distributed into both deviceCPU and deviceGPU based on the workload ratio computed for each task [7, 8, 9]) method, and the proposed method are compared in Fig. 5.

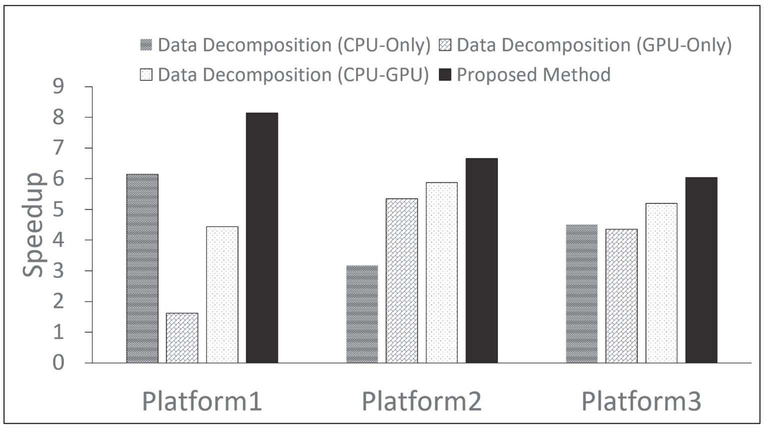

Fig. 5. Speedup comparison between the data decomposition-based method such as [7, 8, 9] and the proposed method.

As explained in Section 2, the data dependencies of Task1, Task2, and Task3 (i.e., *low-level* computer vision tasks) were "local and regularly-structured", and thus Task1, Task2, and Task3 matched well with deviceGPU. However, the data dependencies of the most time-consuming task Task4 (i.e., *intermediate-level* computer vision task) were "global and irregularly-structured". As shown in Fig. 4, it was important to reduce the most time-consuming task. Because the number of merge steps of a typical data decomposition method (see Fig. 1 in Section 2) on a manycore deviceGPU was much greater than that on a multicore deviceCPU, the computational characteristics of Task4 did not match well with deviceGPU. Although we implemented the method of Task4 [10] by reducing the merge steps, Task4 was too slow on deviceGPU of platform1 (i.e., it was unnecessary to execute Task4 on both deviceCPU and deviceGPU if we consider the communication time additionally).

As shown in Fig. 5, the speedup of the proposed method was better than that of either the data decomposition-based CPU-only, GPU-only, or CPU-GPU method, regardless of the platform on which it was used (i.e., a CPU-higher, GPU-higher, and comparative platform). If the number of tasks of a target vision system is increased, then the proposed method can increase the chance to balance the execution times between deviceCPU and deviceGPU with the increased number of tasks. If we distribute each task across both deviceCPU and deviceGPU, however, then we need to determine the workload distributed to each device for each task, with additional programming burden to satisfy the given data dependency. In this sense, the proposed method is practical as well as *scalable*, so that it can be applied to parallelize a complete system consisting of large number of tasks.

With the current target vision system, we could obtain the parallel processing speed of 400 (on platform1), 595 (on platform2), 650 (on platform3) fps, from the sequential processing speed of 49 (on platform1), 89 (on platform2), 107 (on platform3) fps. Note that pig detection is the initial step for intelligent pig

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018

monitoring such as aggressive behavior monitoring [11] after individual pig tracking, and thus we need to reduce the pig detection time as much as possible for the final goal of real-time intelligent pig monitoring.

Finally, Fig. 6 shows the input and output images generated by applying the proposed method. Since the light in the pig room was turned off (i.e., color data was not available) during night-time, we compared the input color and depth images captured at day-time with the corresponding output image. As we can see the depth image captured with a low-cost Kinect camera, the contrast between the background and foreground was not clear. The low contrast problem was effectively solved by the depth-based pig detection method.

In fact, we are currently implementing more numbers of tasks to smooth the outlines of pigs shown in Fig. 6(c). As explained, the more numbers of tasks, the more numbers of workload ratios should be determined in the data decomposition-based heterogeneous method [7, 8, 9]. Furthermore, for example, Task5 in the next version of the target vision system (i.e., CLAHE [12] for contrast enhancement) matches well with deviceCPU, rather than deviceGPU, because Task5 is also an irregularly structured computation such as Task4. Then, the data decomposition-based heterogeneous method [7, 8, 9] divides the data for Task5 based on the optimal workload ratio of Task5 in order to execute Task5 into both deviceCPU and deviceGPU, whereas the proposed method executes Task5 on deviceCPU (matching well with Task5) only. If we consider additionally the communication time, then the speedup of [7, 8, 9] for the next version of the target vision system having more numbers of tasks may be much worse than that of the proposed method.

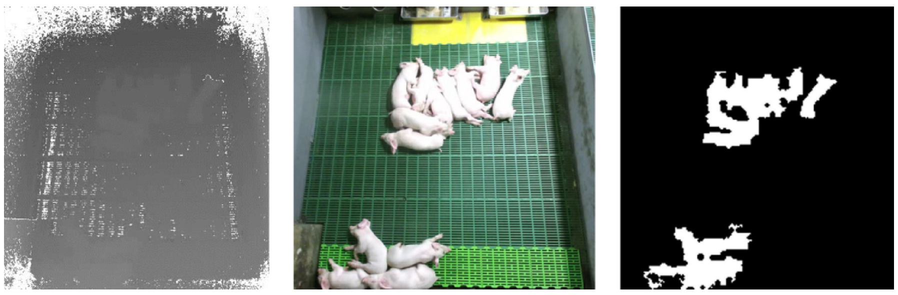
(a) Depth Input (b) Color Input (c) Output

Fig. 6. The input and output images generated by applying the proposed method.

4 Conclusions

By using the depth information obtained from a Kinect sensor, we proposed a method to detect the foreground at night, automatically, on CPU-GPU heterogeneous computing platforms. To satisfy the real-time requirement, we parallelized each task of the entire foreground detection system consisting of *low-level* and *intermediate-level* computer vision tasks with OpenCL. Then, we applied the pipeline scheduling strategy by determining a computing device for each task and balancing the total execution times of CPU and GPU simultaneously.

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018

Experimental results showed that the OpenCL-based pipelining method could provide better speedup than the data decomposition-based CPU-only, GPU-only, or CPU-GPU (i.e., workload was distributed into both deviceCPU and deviceGPU based on the workload ratio computed for each task [7, 8, 9]) method, regardless of platform. That is, based on the actual execution times of each data decomposed task on CPU and GPU, we could reduce the total execution time of several tasks practically, without executing the irregularly-structured *intermediate-level* vision task on both CPU and GPU. We believe the simple, portable, and effective pipelining method can also be applied to other video-based analysis applications consisting of several tasks (i.e., *low-level* and *intermediate-* or *high-level* computer vision tasks with different computational characteristics) to satisfy the real-time requirement.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the MEST (2015R1D1A1A09060594).

© IEICE 2018
DOI: 10.1587/elex.15.20170950
Received September 13, 2017
Accepted January 19, 2018
Publicized February 2, 2018
Copyedited February 25, 2018