Real-time processing for intelligent-surveillance applications

Sungju Lee, Heegon Kim, Jaewon Sa, Byungkwan Park, and Yongwha Chung

Dept. of Computer and Information Science, Korea University,
2511 Sejong-ro, Sejong City 30019, Korea

Abstract: Although the typical sensor-server model has been widely used in intelligent-surveillance applications, the workload increases for the centralized server as the number of sensors increases; therefore, the provision of a scalable performance requires the distribution of the centralized-server workload into the sensors. Due to the limited resources of the sensor side, however, a resource-efficient real-time processing technique is required. In this paper, a real-time sensor-side surveillance technique for which parallel processing is used on CPU-GPU hybrid-computing devices is proposed. The experiment results reveal that the proposed method can provide a real-time execution of the surveillance system.

Keywords: surveillance, real time, Amdahl’s law, video sensor, CPU-GPU computing, parallel processing

Classifications: Integrated circuits

References

[1] Y. B. Chin, et al.: “Extended fuzzy background modeling for moving vehicle detection using infrared vision,” IEICE Electron. Express 8 (2011) 340 (DOI: 10.1587/elex.8.340).
[2] T. T. Zin, et al.: “Background modeling using special type of Markov chain,” IEICE Electron. Express 8 (2011) 1082 (DOI: 10.1587/elex.8.1082).
[3] Y. Chung, et al.: “Automated detection of cattle mounting using side-view camera,” Trans. Internet Inf. Syst. 9 (2015) 3151 (DOI: 10.3837/tiis.2015.08.024).
[4] S. Lee, et al.: “Energy efficient image/video data transmission on commercial multi-core processors,” Sensors 12 (2012) 14647 (DOI: 10.3390/s121114647).
[5] A. Belbachir: Smart Cameras (Springer, US, 2010) (DOI: 10.1007/978-1-4419-0953-4).
[6] 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).
[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] S. Mittal and J. Vetter: “A survey of CPU-GPU heterogeneous computing techniques,” ACM Comput. Surv. (CSUR) 47 (2014) 69 (DOI: 10.1145/
1 Introduction

In general, video-sensor data have been widely employed in automatic-surveillance applications that monitor vehicles, people, or animals [1, 2, 3]. Although the focus of the previous studies is mostly the surveillance accuracy, large-scale surveillance applications should also consider a real-time analysis of 24-hour video-stream data. For example, many surveillance systems have video sensors that use compression to send 24-hour video stream data to a server. The server then decompresses the received video and interprets the input scene [4]. Although this typical sensor-server model has been widely used in surveillance applications, the workload increases for the centralized server as the number of sensors increases. Therefore, to ensure a scalable performance, the workload of the centralized server needs to be distributed into the sensors [5].

To execute a real-time analysis of 24-hour video-stream data at the sensor side, the parallelism that exists in surveillance applications can be exploited. APUs with both multicore CPUs and GPUs have been recently released, and these multicore processors can be used to reduce the execution time [6, 7, 8]. Therefore, an embedded implementation of the surveillance applications that is based on CPU-GPU hybrid-computing devices is required to achieve a real-time execution.

The parallelization of the surveillance applications requires a consideration of the low-level feature extraction and high-level analysis that constitute the typical surveillance systems. In general, the low-level part has a sufficient parallelism whereas the high-level part does not. For example, many video-based surveillance systems use the Gaussian mixture model (GMM) as a major technique to separate the foreground of an image from its background [1, 2, 3], and the GMM is a time-consuming low-level task with a sufficient parallelism. If the GMM contributes 51% of the total workload in an intelligent-surveillance application and all of the remaining tasks are sequentially portioned, then the ideal speedup with an infinite number of processors is limited by a power of two according to Amdahl’s law [9]. Therefore, the sequential portions should be considered in addition to the parallel ones in a complete surveillance system to further improve the speed.

In this paper, both the parallel and sequential portions in a surveillance system are considered, and both the data and the task parallelism are exploited to overcome the speedup that is limited by Amdahl’s law. It is important to find an effective method that fully utilizes all of the computational resources that are available in both the CPU and the GPU. The execution time and data dependency of each module in an intelligent-surveillance application are analyzed, and an optimal workload is then assigned to both the CPU and the GPU. In addition, the system is designed so that the execution time overlaps with the data-copy time to avoid the
degradation of the parallelization efficiency. Since video-streaming data are continuous within a static workload, the static strategy can be efficiently used to distribute the workload to both the CPU and the GPU without the need for a complicated scheduling overhead.

To the best of the authors’ knowledge, this is the first study of a complete surveillance system that can automatically interpret an input scene at the sensor side in real time. Furthermore, the experiment results reveal that the proposed method can overcome the ideal speedup that is computed by Amdahl’s law, and the proposed methodology can also be applied to other 24-hour surveillance applications with various sensors, such as audio, depth, heat, and infrared.

2 Background

In this paper, the low-weight pig detection is considered as an example of an intelligent-surveillance application. Finding the low-growth pigs in livestock is a very important issue for increasing the productivity. In previous studies [10], a low-weight-pig detection system was developed. Fig. 1 shows a flow chart of a computer-vision system that was developed in a previous study.

![Flow chart of low-weight-pig detection](image)

**Fig. 1.** Flow chart of low-weight-pig detection

Although this typical sensor-server model has been widely used in surveillance applications, the workload increases for the centralized server as the number of sensors increases; that is, since dozens of pigs are raised in a single pig room and the number of pig rooms ranges from dozens to hundreds, the low-weight-pig system should be performed at the sensor side to reduce the server workload. In this paper, a parallel-processing technique is applied for the efficient use of the CPU-GPU hybrid-computing devices.
First, the sequential-execution time of the specific modules is analyzed to parallelize the surveillance system, as shown in Table I. The surveillance system consists of the following modules: image capture, HSV conversion, binarization, the GMM, pixel counting, full-moving-pig detection, decision low-weight pig, and notification generation. As shown in Table I, these modules are categorized into two parts, as follows: sequential, \( t_1 \) and \( t_3 \); and parallel, \( t_2 \). These tasks must be performed in the following sequence due to their data dependencies: \( t_1 \), \( t_2 \), and \( t_3 \).

| Detection of low-weight pigs     | Rate  | Sequential and Parallel portions |
|---------------------------------|-------|---------------------------------|
| Image capture                   | 11.96%| Sequential portion \( (t_1) \) |
| HSV conversion                  | 5.43% |                                   |
| Binarization                    | 6.57% | Parallel portion \( (t_2) \)     |
| GMM                             | 51.39%|                                   |
| Pixel-counting                  | 22.15%|                                   |
| Full-moving-pig detection       | 10.04%| Sequential portion \( (t_3) \)   |
| Decision low-weight pig         | 2%    |                                   |
| Notification generation          | 2%    |                                   |

In the surveillance system, the execution time of the parallel part (74%) is greater than that of the sequential part (26%), so a high speedup is expected during the implementation of the parallel-processing techniques. In general, the speedup is limited to the rate of the parallel part of an application, even when infinite parallel processors are used, according to Amdahl’s law. However, the execution time of the sequential part should not be ignored when reducing the total execution time; moreover, the typical approaches apply parallel-processing techniques to the GMM since the GMM requires the maximum computation time among all of the modules. For example, when parallel processing is performed with \( n \)-cores on a frame, and the parallel portion is 51%, the ideal speedup can be calculated as \( 1/[(0.12 + 0.14) + 0.74/n] \). According to Amdahl’s law, the ideal speedup in the surveillance system is 2.25 when four cores \( (n = 4) \) are present. Even if an infinite number of processors is used, the ideal speedup would be less than 3.8, in accordance with Amdahl’s law. Therefore, a parallel algorithm that uses not only data parallelism but also task parallelism with simultaneously executed multiple sequential workloads should be designed.

3 Proposed approach

3.1 Intelligent-surveillance application

In this paper, parallel-processing techniques for which CPU-GPU hybrid computing devices are used are applied to the surveillance system to reduce the total execution time. As previously stated, the surveillance system consists of the following modules: image capture, HSV conversion, binarization, the GMM, pixel counting, full-moving-pig detection, decision low-weight pig, and notification generation.
generation. In particular, the proposed system can help in the improvement of the quality and productivity of pig farms by notifying the farm administrators of the presence of low-weight pigs in their pig rooms.

It must be noted that each module has data dependencies. For example, the HSV-conversion module cannot be executed without the completion of the image-capture module. However, the intra-frame computations, namely HSV conversion, binarization, and GMM, do not have data dependencies; therefore, the use of the data-level parallelism to parallelize the modules of the surveillance system is more efficient than the use of the task-level parallelism.

![Data dependencies in the low-weight-pig detection system](image)

Fig. 2. Data dependencies in the low-weight-pig detection system

Inter-frame data dependencies exist in the modules of the parallel part. As shown in Fig. 2, when several frames are computed using HSV conversion and binarization, data dependencies are nonexistent between the inter- and intra-frames. In contrast, since the GMM generates a background model by using a probability function that requires inter-frame data, the data dependencies are present. For example, if a current frame is calculated using the GMM, the pixel value of the “key frame”—that is, the previous frame that was used to calculate the background model—is required. Therefore, it is possible to see the data dependencies between the inter-frames of the surveillance-system modules.

3.2 Parallel scenarios in the CPU-GPU hybrid-computing devices

In the proposed method, the frame is divided into two parts that are separately assigned to the CPU and the GPU. The frame is divided based on the height, and therefore a fast execution time can be expected with the locality. In addition, the
size of the workload is determined through the use of the speedup for both the CPU and the GPU. Lastly, the surveillance system can be efficiently parallelized with the use of an optimal workload ratio.

Considering the characteristics of the surveillance system, possible scenarios have been designed for this study to efficiently parallelize the modules on the CPU-GPU hybrid-computing devices, as shown in Fig. 3. Both of the scenarios execute the parallel processing every $n$ frames. If the CPU has four cores, for example, the scenarios execute the parallel processing every four frames, as shown in Fig. 3. Note that $t_1$ and $t_2$ can be sequentially computed while $t_2$ can be computed in parallel. When the modules are executed on a GPU, a data-copy time is also required.

In Scenario 1, when the frames are continuously processed, the idle time can be reduced by the overlapping of each $t_1$ and $t_3$. In this case, the number of times $t_1$ is executed is determined based on the number of CPU cores. Note that $t_1$ and $t_3$ overlap every four frames, as shown in Fig. 3(a). However, if either $t_1$ or $t_3$ is greater than $t_2$ on the GPU, Scenario 1 may negatively impact the execution. In another scenario, the execution on the GPU occurs every four frames. In Scenario 2, the idle time can be reduced by overlapping $t_1$, $t_2$, and $t_3$ with the data-copy

![Fig. 3. Possible scenarios for the efficient parallelization of the modules on CPU-GPU hybrid-computing devices](image)
time. In this case, the $t_1$ computations for the subsequent frames should be executed before $t_2$ for the current frames to overlap $t_1$, $t_2$, and $t_3$ with the data-copy time. As shown in Fig. 3(b), Scenario 2 can reduce the execution time unlike Scenario 1; however, the implementation of the source code for such a case is difficult.

### Algorithm 1. Scenario 1

/* Execution for eight frames */
Step 1: M.FRAME = 4; // # of CPU cores
Step 2: frame_id = init;
Step 3: thread.work ($t_1[i]$);
   SNY(ALL thread);
Step 4: for(i=frame_id; i<= M.FRAME; i++){
   thread.work ($t_2[i]$ for CPU);
   thread.work ($t_2[i]$ for GPU);
   If (i % N.FRAME == 0) thread.work ($t_3[i]$);
   Else thread.work ($t_1[i] + 1$);
}
   SNY(ALL threads);
Step 5: thread.work ($t_1[i]$);
   for(i=frame_id; i<= M.FRAME-1; i++) ($t_3[i]$);
   SNY(ALL threads);

### Algorithm 2. Scenario 2

/* Execution for eight frames */
Step 1: M.FRAME = 4; // # of CPU cores
Step 2: frame_id = init;
Step 3: for(i=frame_id; i<= M.FRAME; i++)
   thread.work ($t_1[i]$);
   SNY(ALL threads);
Step 4: i = frame.id;
   thread.work ($t_2[i]$, $t_2[i+1]$, $t_2[i+2]$, $t_2[i+3]$ for GPU);
   thread.work ($t_2[i]$, $t_2[i+1]$, $t_2[i+2]$, $t_2[i+3]$ for CPU);
   for(i=frame_id; i<= M.FRAME; i++)
      thread.work ($t_1[i+M.FRAME]$);
   SNY(ALL threads);
Step 5: frame_id += M.FRAME;
Step 6: i = frame.id;
   thread.work ($t_2[i]$, $t_2[i+1]$, $t_2[i+2]$, $t_2[i+3]$ for GPU);
   thread.work ($t_2[i]$, $t_2[i+1]$, $t_2[i+2]$, $t_2[i+3]$ for CPU);
   for(i=frame_id; i<= M.FRAME; i++)
      thread.work ($t_1[i+M.FRAME]$);
   SNY(ALL threads);
Step 7: for(i=frame_id; i<= M.FRAME; i++)
      thread.work ($t_3[i]$);
      SNY(ALL threads);
Consequently, the efficiencies of Scenarios 1 and 2 depend on the execution times of \( t_1, t_2, \) and \( t_3 \), as well as the data-copy time. In Scenario 1, \( t_1, t_2, \) and \( t_3 \) on the GPU affect the total execution time; for example, when \( t_2 \) on the GPU time is equal to \( t_1 \) and \( t_3 \), the performance of Scenario 1 is maximized. In contrast, when the data-copy time is equal to \( t_1 \) and \( t_3 \), the performance of Scenario 2 is maximized. In summary, Scenario 1 is affected by the execution times of \( t_1 \) and \( t_3 \) while Scenario 2 is affected by the data-copy time. Therefore, the parallel-processing scenarios have been determined according to the sizes of \( t_1, t_2, \) and \( t_3 \), and the data-copy time. For example, if \( t_1 \) and/or \( t_3 \) are less than \( t_2 \) on the GPU, the efficiency of Scenario 1 is greater than that of Scenario 2.

Scenario 1 and Scenario 2 are implemented using Algorithm 1 and Algorithm 2, respectively. To execute each task on the CPU and GPU, OpenMP and OpenCL are exploited, and Pthread is used for each task distribution [7]. Scenario 1 and Scenario 2 parallelize the eight-frames (i.e., twice the number of CPU cores). In Scenario 1, \( t_1 \) of one frame is first performed, as shown in Step 3 of Algorithm 1, and \( t_2 \) of four frames and \( t_3 \) of three frames are allocated to the CPU and the GPU, respectively (see Algorithm 1, Step 4). Lastly, \( t_3 \) is parallel processed on the CPU. Note that, between each step, the threads are synchronized in consideration of the account-data dependency (i.e., SNY (All thread)). In Scenario 2, \( t_1 \) is performed for four frames, as shown in Algorithm 2, Step 3, and \( t_1 \) and \( t_2 \) of the four frames are allocated to the CPU and GPU, respectively (see Algorithm 2, Step 4). Note that \( t_1 \) and \( t_3 \) are alternately performed while the processing is repeated for the four frames, as shown in Algorithm 2, Step 4 and Step 6. Lastly, \( t_3 \) is processed simultaneously on the CPU, as in Scenario 1.

Also, the performances of Scenario 1 and Scenario 2 vary depending on the sequential portion (i.e., \( t_1 \) and \( t_3 \)) and the parallel portion (i.e., \( t_2 \)). For example, \( t_1 \) has a relatively large effect on Scenario 1 compared with Scenario 2, and the performance of Scenario 1 is consequently lower than that of Scenario 1 when \( t_1 \) is relatively large; therefore, it is necessary to determine Scenario 1 and Scenario 2 according to the task size.

To determine the most efficient scenario, some of the following equations were derived. The total worst execution time of Scenario 1 for the \( m \)-frames and \( n \)-cores of the CPU is represented by the following equation:

\[
T_{\text{Scenario}1}(t_1, t_2, t_3) = t_1 + \frac{m}{n} \times \max(T_{\text{CPU}}(t_1, t_2 \text{ on CPU}, t_3), T_{\text{GPU}}(t_2 \text{ on GPU})) + t_3. \quad (1)
\]

where \( T_{\text{CPU}}(\cdot) \) and \( T_{\text{GPU}}(\cdot) \) are the module-execution times on the multicore-CPU and multicore-GPU for Scenarios 1 and 2, respectively. Also, \( t_1, t_2, \) and \( t_3 \) affect the CPU-execution time; by contrast, \( t_2 \) affects the GPU-execution time.

In Scenario 1, \( T_{\text{CPU}}(\cdot) \) depends on \( t_1, t_2, \) and \( t_3, \) and \( T_{\text{CPU}}(\cdot) \) can be represented using the following equation:

\[
T_{\text{CPU}}(t_1, t_2 \text{ on CPU}, t_3) = (n - 1) \times \left\{ \max\left( \frac{t_2 \text{ on CPU}}{\text{SPEEDUP}_{\text{CPU}}}, (2 \times \text{DataCopyTime} + t_1) \right) \right. \\
+ \max\left( \frac{t_2 \text{ on CPU}}{\text{SPEEDUP}_{\text{CPU}}}, (2 \times \text{DataCopyTime} + t_3) \right) \right\}. \quad (2)
\]
Note that the \( n - 1 \) frame depends on \( t_1 \) and one frame depends on \( t_3 \). Also, \( \text{SPEEDUP}_{\text{CPU}} \) and \( \text{SPEEDUP}_{\text{GPU}} \) can be measured by using the speedup on the CPU and GPU, respectively. In addition, \( t_2 \) can be divided using \( \text{SPEEDUP}_{\text{CPU}} \) and \( \text{SPEEDUP}_{\text{GPU}} \).

In Scenario 1, \( T_{\text{GPU}}(\cdot) \) depends on \( t_2 \), and \( T_{\text{CPU}}(\cdot) \) can be represented using the following equation:

\[
T_{\text{GPU}}(t_2 \text{ on GPU}) = n \times \left( \frac{t_2 \text{ on GPU}}{\text{SPEEDUP}_{\text{GPU}}} + 2 \times \text{DataCopyTime} \right).
\]  \( (3) \)

In Scenario 2, the total execution time can be represented using the following equation:

\[
T_{\text{Scenario}2}(t_1, t_2, t_3) = t_1 + \frac{m}{2n} \times \max(T_{\text{CPU}}(t_1, t_2 \text{ on CPU}, t_3), T_{\text{GPU}}(t_2 \text{ on GPU})) + t_3. \]  \( (4) \)

The \( T_{\text{CPU}}(\cdot) \) and \( T_{\text{GPU}}(\cdot) \) for Scenario 2 can be represented using the following two equations, respectively:

\[
T_{\text{CPU}}(t_1, t_2 \text{ on CPU}, t_3) = 2n + \frac{t_2 \text{ on CPU}}{\text{SPEEDUP}_{\text{GPU}}} \times \max(t_1, t_2). \]  \( (5) \)

\[
T_{\text{GPU}}(t_2 \text{ on GPU}) = 2n \times \left( \frac{t_2 \text{ on GPU}}{\text{SPEEDUP}_{\text{GPU}}} + 2 \times \text{DataCopyTime} \right). \]  \( (6) \)

Note that Scenario 2 reduces the idle time with the overlapping of \( t_1 \) or \( t_2 \) to the data-copy time of the GPU. Since either \( t_1 \) or \( t_2 \) is overlapped by the data-copy time of the GPU, \( T_{\text{CPU}}(\cdot) \) can be represented by Eq. (5).

When both \( t_1 \) and \( t_3 \) are smaller than \( t_2 \) on the GPU, Scenario 1 offers a better performance compared with Scenario 2; however, when both \( t_1 \) and \( t_2 \) are smaller than the data-copy time of the GPU for the \( n \)-frames, Scenario 2 offers the better performance. In the equations, the algorithm that is used to determine the most efficient scenario has been simply designed. In such a case, the \( t_1 \), \( t_3 \), and \( t_2 \) on the GPU are compared. Lastly, the most efficient scenario can be determined through the use of Algorithm 3, as follows:

**Algorithm 3. Selecting the most efficient scenario**

**Step 1:** Measurement of \( t_1 \), \( t_3 \), and \( T_{\text{GPU}}(t_2 \text{ on GPU}) \)

**Step 2:** Determine the most efficient scenario

If : \( \max(t_1, t_3) < T_{\text{GPU}}(t_2 \text{ on GPU}) \)

Then select Scenario 1

Else : Select Scenario 2

### 4 Experiment results

In the experiments of this study, the resolution sizes were set to \( 1200 \times 900 \), and \( 1440 \times 1080 \), and the frame rate is 20 frames per second. The camera was fixed four meters above the floor to monitor a pig room with the dimensions of 4 meters \( \times \) 3 meters and containing 23 weaned pigs. A background subtraction was initially performed to remove some noise from the input-video sequence; additionally, low-weight pigs were detected.
Ten full-moving pigs were first detected in the 10-minute input video. Next, the number of pixels in each full-moving pig was counted. To identify the low-weight pigs, an average was derived through the summation of the size (i.e., the number of pixels) of each full-moving pig in the video sequence. The low-weight-pig threshold was set as 70% of the average.

To evaluate the proposed method, the experiment was conducted using an Intel Core i5-3570 CPU with four cores, a GeForce GTX 660 GPU with 960 cores, and 7200 image frames (5 minutes of video). In addition, OpenMP, OpenCL, and Pthread [7] were used to parallelize the feature extraction on both the CPU and the GPU. Notably, although OpenMP does not require a data-copy time, it does not provide detailed operations for asymmetric task assignments.

First, the optimal workload ratio for the various image frames was determined, as shown in Table II. In the proposed method, three of the CPU cores were used to execute the workload. The optimal workload ratios are 67%:33% and 66%:34% for the 1200 × 900 and 1440 × 1080 image frames, respectively.

The performances of the proposed CPU-GPU hybrid-computing method were compared in terms of the speedup to evaluate the proposed method in further detail. Fig. 4 shows the speed-ups for the CPU-only, GPU-only, Scenario 1, and Scenario 2 relative to the sequential approach. Note that the parallel portion is 74% of the workload (i.e., the sequential portion is 26%) in the proposed surveillance system, and consequently the ideal speedup is less than 3.8, even with the use of infinite processors, in accordance with Amdahl’s law. To avoid the data dependency of the sequential portions (i.e., \( t_1 \) and \( t_3 \)), the proposed method was used to overlap the sequential portion, parallel portion, and data-copy time (i.e., Scenario 1 and Scenario 2), and then the limited speedup was overcome using Amdahl’s law; that is, \( t_1 \), \( t_2 \), \( t_3 \), and the data-copy time were overlapped in both Scenario 1 and Scenario 2, and therefore even the sequential portions can be processed in parallel. Scenario 2 was also determined as the most efficient scenario through the use of Algorithm 3, because \( t_1 \) (i.e., \( \max(t_1, t_3) \)) is larger than \( t_2 \) on the GPU. The results confirmed that, from an ideal speedup of 3.8, the possible scenarios can improve the speedup to 4.1 of Scenario 1 and 4.3 of Scenario 2, respectively.

| Workload ratio | \( W_{\text{CPU}} \) | \( W_{\text{GPU}} \) |
|----------------|------------------|------------------|
| 1200 × 900    | 1200 × 297       | 1200 × 603       |
| 1440 × 1080   | 1440 × 355       | 1440 × 725       |
Lastly, the frames per second (FPS) were measured using different image-frame sizes to verify the real-time operation, and the results are shown in Table III. In the experimental environment, the video consists of 20 FPS, so the FPS would have to be greater than 20 to satisfy the real-time requirement. Both the CPU-only and GPU-only configurations could not satisfy the real-time requirement when the image-frame sizes are $1200 \times 900$ and $1440 \times 1080$. In contrast, it was confirmed that both Scenarios 1 and 2 could be performed in real time using a large image-frame size through the simultaneous use of the CPU and the GPU. In addition, it was confirmed that Scenario 2 could provide better execution times than Scenario 1. Note that the real-time execution is required with high-resolution videos to conduct an accurate high-level analysis, such as an abnormal-behavior analysis or the tracking of individual objects. Accordingly, a consideration of the complete surveillance system should involve the exploitation of the maximum parallelism instead of the parallelization of a specific task, such as the GMM.

### Table III. FPS with various image-frame sizes (unit: FPS)

| Image-frame size | Sequential | Parallel |
|------------------|------------|----------|
|                  | 1200 $\times$ 900 |   1440 $\times$ 1080 |
| Sequential       | 7           | 5        |
| Parallel         |             |          |
| CPU-only         | 14          | 9        |
| GPU-only         | 17          | 11       |
| Scenario 1       | 29          | 20       |
| Scenario 2       | 31          | 21       |

5 Conclusions

In this paper, a real-time surveillance technique is proposed in consideration of not only the parallel parts but also the sequential parts of a complete surveillance system. To overcome the speedup limit that is due to Amdahl’s law, the parallelism for the complete surveillance system was maximally extracted with the use of both...
the data and task parallelisms. Then, the execution time was reduced through a simultaneous use of the CPU and the GPU. In other words, $t_1$, $t_2$, $t_3$, and the data-copy time were overlapped in both Scenario 1 and Scenario 2, and therefore even a sequential portion can be processed in parallel. The experiment results confirmed that the proposed method can execute an intelligent-surveillance application in real time and can improve the speedup to 4.3 from the Amdahl’s ideal speedup of 3.8. Although the proposed parallel-implementation method was applied to a 24-hour video-surveillance application, the proposed methodology is general enough to be applicable to the 24-hour surveillance application with other sensors.

**Acknowledgments**

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Science, ICT & Future Planning (2015R1C1A1A02037688) and the Leading Human Resource Training Program of Regional Neo Industry (2016H1D5A1910730).