J Supercomput (2015) 71:2275–2308
DOI 10.1007/s11227-015-1382-3

An optimal many-core model-based supercomputing for accelerating video-equipped fire detection

Junsang Seo · Myeongsu Kang · Cheol Hong Kim · Jong-Myon Kim

Published online: 24 January 2015

Abstract Automatic fire detection has become more and more appealing because of the increasing use of video capabilities in surveillance systems used for early detection of fire. However, its high computational complexities limit its use in real-time applications. To meet the real-time processing of today's fire detection techniques, this study proposes a single instruction, multiple data many-core model. To design an efficient many-core model for image processing applications such as fire detection, a key design parameter is the image data-per-processing-element (IDPE) variation of the many-core system, which is the amount of image data directly mapped to each processing element PE. This study quantitatively evaluates the impact of the IDPE variation on system performance and energy efficiency for the multi-stage fire detection approach that consists of movement-containing region detection, color segmentation, fire feature extraction of fires, and decision making if there is a fire or non-fire in a processing video frame. In this study, we use six IDPE ratios to determine an optimal many-core model that provides the most efficient operation for fire detection using architectural and workload simulation. Experimental results indicate that the most efficient many-core model is achieved at the 64 IDPE value in terms of the worst-case execution time and energy

J. Seo · J.-M. Kim (✉)
School of Electrical Engineering, University of Ulsan,
Bldg. #7, Room #308, 93 Daehak-ro Nam-gu, Ulsan 680-749, South Korea
e-mail: jongmyon.kim@gmail.com; jmkim07@ulsan.ac.kr

J. Seo
e-mail: jsseo2006@gmail.com

M. Kang · C. H. Kim
School of Electronics and Computer Engineering, Chonnam National University,
Bldg. #7, Room#506, 77 Yongbong-ro, Buk-gu, Gwangju 500-757, South Korea
e-mail: ilmareboy@gmail.com

C. H. Kim
e-mail: cheolhong@gmail.com

efficiency. In addition, this study compares the performance of the most efficient many-core configuration with that of a commercial graphics processing unit (Nvidia GeForce GTX 480) to show the improved performance of the proposed many-core model for the fire detection algorithm. This many-core configuration outperforms the commercial graphic processing unit in the worst-case execution time and energy efficiency.

Keywords Design space exploration · Fire detection · General-purpose graphic processing units · Many-core model

1 Introduction

Early fire detection has been an increasingly important issue since it is closely related to personal security and property. In spite of the fact that sensor-based fire detection systems first came into the spotlight by detecting either heat or smoke for early identification of whether or not a fire is occurring, these systems have the drawback that sensors should be densely distributed in a wide area for a highly precise fire detection system [1]. Likewise, sensor-based fire detection systems may be inappropriate for early fire detection since the spread of heat and smoke requires some time after the onset of fire.

Recent advances in video processing technologies have led to a wave of research on computer vision-based fire detection systems whose advantages are summarized as follows [2]:

- As the speed of light transmission is much faster than that of the heat and smoke, computer vision-based fire detection is appropriate for early detection of fire.
- In general, images have more scene information such as color and texture, which enables diverse approaches to fire detection.

In this study, we quantitatively investigate recent computer vision-based fire/flame detection technologies. Most are based on the multi-stage pattern recognition, which basically consists of the following four stages [3–20]: movement-containing region detection (MCRD), color segmentation (CS), feature extraction (FE), and classification (CLASSIFY). MCRD is a fundamental task in computer vision-based fire detection and a number of methods have been proposed to detect movement-containing regions from static cameras, including optical flow [12,22], temporal differencing [21], Gaussian mixture modeling [10,17], and background subtraction [11,13,16,20,23–25]. Background subtraction has been widely used for MCRD in fire detection due to its high efficiency. Although MCRD selects candidate regions of fire, the candidate regions may still include moving objects with colors that are similar to the color of fire such as red vehicles, vehicle brake lights, and a person wearing red clothes. To deal with this issue, many researchers then detect more refined regions of fire using color information [2–7,9–11,14,16,17,20,25,26], and extract features of fire including characteristics of fire such as flicker [10,11,23,26,27], color variations under spatial wavelet analysis [11,23,26], and dynamic textures [19]. Finally, a classifier is employed to determine if there is a fire or a non-fire in a processing movie frame. To do this, the following classifiers are commonly considered: support vector machines [5,9,17,28–30] and neural networks [12,20]. As a classifier, neural network techniques have

inherent drawbacks such as local optimization problems, lack of generalization, and uncontrolled convergence. In addition, they require a large amount of training data to achieve high classification accuracy. Unlike neural networks, support vector machines have the following advantages: (1) the computational complexity of SVMs does not depend on the dimensionality of the input feature vector, (2) SVMs are less prone to overfitting, which can be a major reason that SVMs often outperform neural networks in practice, and (3) the solution to SVMs is global and unique while neural networks suffer from the existence of multiple local minima solutions as aforementioned. Thus, we use a support vector machine as a classifier for early detection of fire in this study.

To precisely detect fire in movies, these complicated fire detection approaches are commonly utilized. However, they demand tremendous computational cost, which limits their use in real-time applications [31]. Among the many computational models available for complex image processing, parallel computers including single instruction multiple data (SIMD) many-core architectures and graphics processing units (GPUs) are promising candidates [32–39]. Typically, thousands of processing elements (PEs) in the many-core architecture are utilized to accelerate target applications including fire detection. Currently, GPUs have been adopted in fire detection due to their high computational throughput capability [40]. However, inter-streaming multiprocessor (SMs) communication on GPUs is inefficiently achieved by implementing barrier synchronization via the host, which occurs by terminating the current GPU-offloaded computation and then re-launching a new GPU-offloaded computation. The lack of such inter-SM communication largely limits data-parallel or task-parallel applications [41,42]. Moreover, GPUs consume a large amount of power by integrating more hardware resources and operating them at higher frequencies [43].

While it is evident that the overall performance improves by increasing the number of PEs [44], no general consensus has been reached regarding which grain sizes of processors and memories on the many-core system provide the most efficient operation of the aforementioned fire detection algorithms with regard to performance and energy efficiency. This study introduces the effects of various image data-per-processing-element (IDPE) ratios which indicate the variations in the amount of image data directly mapped to each PE on the performance and energy efficiency, and then identifies the most efficient many-core model that delivers the required processing performance with the longest battery life for the fire detection algorithm. In this study, we quantitatively evaluate the effects of different IDPE ratios using architectural and workload simulation. To determine the most efficient many-core model, six different IDPE ratios are simulated using the fire detection algorithm with five different fire and non-fire movie clips. Experimental results indicate that the most efficient operation is achieved at IDPE $= 64$ in terms of execution time and energy efficiency. In addition, the most efficient many-core configuration outperforms a commercial GPU (i.e., Nvidia GeForce GTX 480) in both execution time and energy efficiency.

The rest of this paper is organized as follows. Section 2 presents a multi-stage fire detection scheme and validates its accuracy; Sect. 3 introduces the reference many-core model along with parallel implementation of the multi-stage fire detection algorithm on it. Section 4 analyzes the application characteristics in terms of the performance and energy efficiency, and finally Sect. 5 concludes this paper.

2 Multi-stage fire detection scheme and its validation

To evaluate the performance of the reference many-core model enabling real-time fire detection, we use the following four-stage fire detection algorithm, consisting of background subtraction-based MCRD, CS for detecting fire-like regions using the YCbCr color model, FE using a normalized wavelet energy which well describes the spatial behavior for fire/flame, and a SVM-based decision making which identifies between fire and non-fire in a processing movie frame (CLASSIFY). Figure 1 shows a flow diagram of the multi-stage fire detection scheme. More details about each step will be given below.

2.1 Movement-containing region detection (MCRD) based on background subtraction

Since the boundaries of a fire tend to continuously fluctuate, MCRD has been widely used as the first step of fire detection, which selects candidate regions of fire. As mentioned in the previous section, background subtraction is commonly utilized for MCRD. Background subtraction separates foreground objects from the background in a sequence of movie frames. A pixel positioned at (i, j) is assumed to be moving if the following condition is satisfied:

$$|I_n(i, j) - B_n(i, j)| > \mathrm{Th}, \tag{1}$$

where $I_n(i, j)$ represents the intensity value of the pixel at location (i, j) in the nth gray-level input movie frame, $B_n(i, j)$ is the background intensity value at the same pixel position, and Th is a threshold value which was experimentally set to 3 in this study. The background intensity value is iteratively updated using (2):

$$B_{n+1}(i, j) = \begin{cases} B_n(i, j) + 1 & \text{if } I_n(i, j) > B_n(i, j) \\ B_n(i, j) - 1 & \text{if } I_n(i, j) < B_n(i, j) \\ B_n(i, j) & \text{if } I_n(i, j) = B_n(i, j) \end{cases}, \tag{2}$$

where $B_{n+1}(i, j)$ is the estimated background intensity value of the pixel at location (i, j) and $B_n(i, j)$ is the previously estimated background intensity value at the same pixel position. Initially, the background intensity value $B_1(i, j)$ is set to the intensity value of the first movie frame, $I_1(i, j)$.

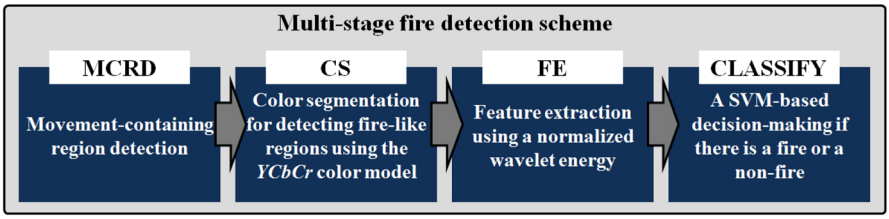

Fig. 1 A flow diagram of the multi-stage fire detection scheme

2.2 Color segmentation for detecting fire-like regions

As mentioned in Sect. 1, a number of moving objects (e.g., people, vehicles, animals, and so on) besides fire can be still included after MCRD. Thus, many researchers have used further information such as color variations and flicker of fire to get rid of spurious fire-like regions. To enhance fire detection ability in this study, we carry out color segmentation, classifying colors that are similar to those of fire after MCRD.

The color of fire is not a reflection of the natural light, but it varies with the chemical properties of the burnt material and its burning temperature (e.g., white, blue, gold, or green). Although a set of rules has been developed over the past few decades to classify fire pixels by utilizing raw red–green–blue (RGB) information in color movie sequences, the RGB color model has shortcomings of illumination dependence. This implies that the illumination change in a movie frame greatly influences RGB-based color segmentation rules. In addition, it is impossible to separate a pixel's value into intensity and chrominance in the RGB color space. However, the chrominance has been widely used in modeling the color of fire in practice because it gives more robust representation for fire pixels. Thus, color spaces, such as YCbCr, YUV, and CIE Lab, can be promising candidates for the purpose of fire-like region detection, in which the chrominance components (Cb and Cr in the YCbcCr color model, U and V in the YUV color model, a and b in the CIE Lab color model) and luminance component (Y in both YCbCr and YUV color models, L in the CIE Lab color model) of a movie frame can be processed independently. According to [45–47], the YCbCr color space is more effective for distinguishing luminance information from chrominance information than other color spaces. Thus, the YCbCr color space is used for the multi-stage fire detection algorithm. The color conversion from the RGB color space to the YCbCr color space is performed as follows:

$$\begin{bmatrix} Y \\ \mathrm{Cb} \\ \mathrm{Cr} \end{bmatrix} = \begin{bmatrix} 0.2568 & 0.5041 & 0.0979 \\ -0.1482 & -0.2910 & 0.4392 \\ 0.4392 & -0.3678 & -0.0714 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}, \tag{3}$$

where Y is the luminance and Cb and Cr are the chrominance components for blue-difference and red-difference, respectively. To model fire pixels, the defined rules for the RGB color space, i.e., R>G>B and $\mathrm{R}>\mathrm{R}_{\mathrm{mean}}$, can be translated into the YCbCr space such as Y>Cb and Cr>Cb. In addition, since the fire-containing regions are generally the brightest regions in the observed scene, the mean values of the three channels include important information, which can be expressed as follows [1]:

$$F_{\mathrm{candidate}}(i,j) = \begin{cases} 1, & \text{if } Y(i,j) > Y_{\mathrm{mean}}, \mathrm{Cb}(i,j) < \mathrm{Cb}_{\mathrm{mean}}, \mathrm{Cr}(i,j) > \mathrm{Cr}_{\mathrm{mean}} \\ 0, & \text{otherwise} \end{cases}, \tag{4}$$

where $F_{\mathrm{candidate}}(i,j)$ indicates that any pixel at the spatial location (i,j) which satisfies the condition given in (4) is labeled as a fire pixel. Likewise, the mean values of the three channels in the YCbCr color space for an $M \times N$ image can be defined as follows:

$$
\begin{aligned}
Y_{\text{mean}} &= \frac{1}{M \times N} \sum_{i}^{M} \sum_{j}^{N} Y(i, j) \\
\text{Cb}_{\text{mean}} &= \frac{1}{M \times N} \sum_{i}^{M} \sum_{j}^{N} Cb(i, j) \\
\text{Cr}_{\text{mean}} &= \frac{1}{M \times N} \sum_{i}^{M} \sum_{j}^{N} Cr(i, j),
\end{aligned} \tag{5}
$$

where $Y(i, j)$, $\text{Cb}(i, j)$, and $\text{Cr}(i, j)$ are the luminance, chrominance-blue, and chrominance-red values at the spatial location (i, j) after MRCD, respectively.

2.3 Feature extraction (FE) via the single-level wavelet decomposition

Due to the nature of turbulent fire flicker, there are generally more color variations in genuine fire-containing regions whereas there are few color variations in the candidate regions of fire, which may still include fire-colored objects after CS. Thus, many researchers have captured color variations in pixel values via the single-level and two-dimensional (2D) discrete wavelet transform (DWT), as depicted in Fig. 2. Note that the color conversion from the YCbCr color space to the RGB color space is performed after CS and the red component is further used for feature extraction. In practice, the wavelet decomposition for an $M \times N$ arbitrary image is performed by first applying one-dimensional (1D) decomposition low-pass and high-pass filters to the rows of the image, which generates two $M \times (N/2)$ sub-images. Then, the 1D decomposition filters are applied to the columns of the two sub-images. Among various Daubehices wavelets, we used a Daubehices 4 wavelet (i.e., db4) in this study because it is effective for avoiding bad localizations. Coefficients of both Daubechies 4 low-pass filter (h) and Daubechies 4 high-pass filter (g) are defined as $h = \{-0.106, 0.0329, 0.0308, -0.1870, -0.0280, 0.6309, 0.7148, 0.2304\}$ and $g = \{-0.2304, 0.7148, -0.6309, -0.0280, 0.1870, 0.0308, -0.0329, -0.0106\}$, respectively.

Since high-frequency information such as edges and texture around the fire is not sensitive to lighting change and more prominent signatures can discriminate irregular fire from the regular movement of fire-colored objects, the wavelet energy of high-frequency sub-images provides a good representation of turbulent fire flicker, which is calculated as follows [23]:

$$
E(n) = \frac{1}{\text{floor}(M/2) \times \text{floor}(N/2)} \left\{ |\text{LH}_n|^2 + |\text{HL}_n|^2 + |\text{HH}_n|^2 \right\}, \tag{6}
$$

where $E(n)$ is the normalized wavelet energy of the nth movie frame, and LH_n, HL_n, and HH_n contain the horizontal, vertical, and diagonal high frequency of the nth $\text{floor}(M/2) \times \text{floor}(N/2)$ sub-images resulting from the single-level wavelet decomposition, respectively. $E(n)$ is then used to train and test a SVM for early detection of fire in movie streams.

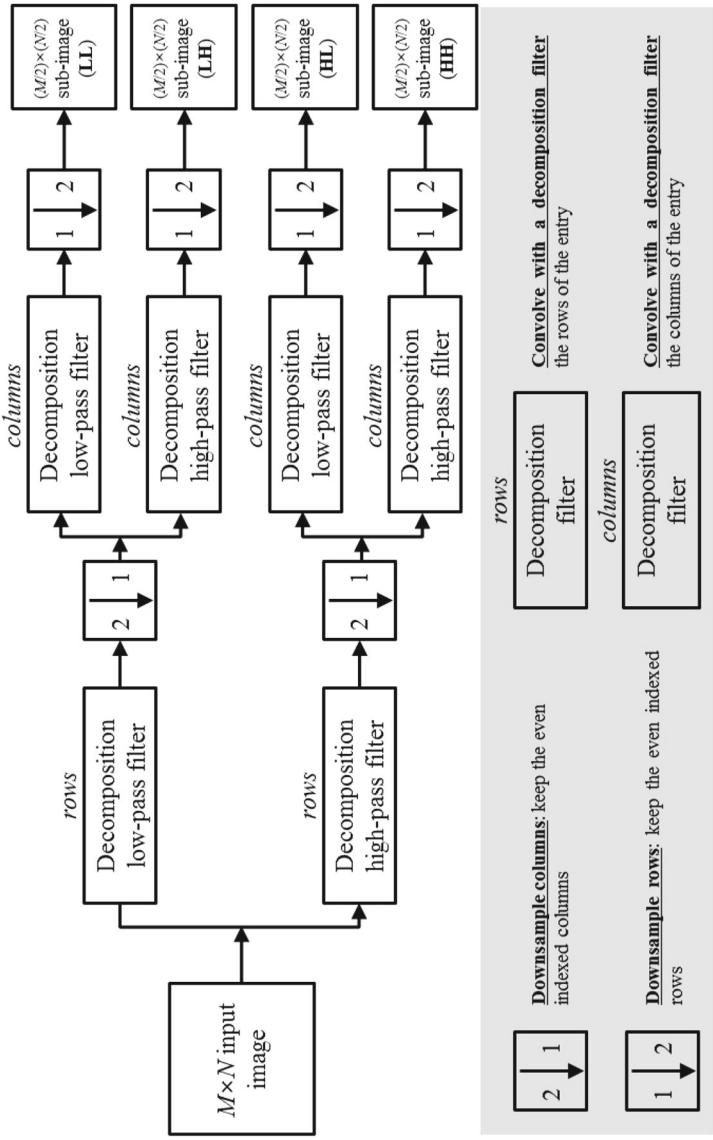

Fig. 2 Decomposition process of the single-level and two-dimensional DWT

2.4 A SVM-based decision making (CLASSIFY)

To classify candidate pixels as fire or non-fire pixels, a support vector machine (SVM) has been widely used since it offers high classification accuracy with limited training data and does not require heuristic parameters for detecting fire pixels. The SVM is a non-probabilistic binary classifier and its main goal is to find an optimal hyper-plane that correctly separates the largest fraction of data points while maximizing the distance between two classes on the hyper-plane. The SVM decision function is defined as:

$$f(x) = \text{sign}\left(\sum_{i=1}^{l} w_i \cdot k(x, x_i) + b\right), \tag{7}$$

where w_i are weights for outputs of each kernel, $k()$ is a kernel, b is a bias term, l is the number of support vectors of x_i, and sign() determines the class membership of x (i.e., +1 class and -1 class). The decision function is then used to measure how much a pixel belonging to the fire class (e.g., +1 class) is different from the non-fire class (e.g., -1 class). In this study, we use a one-dimension feature vector including fire signatures to identify fire in movie streams as mentioned in Sect. 2.3. However, since two classes (e.g., fire or non-fire) are not linearly separable, it is necessary to find an optimal hyper-plane that can split the non-linear feature vector by mapping it to a high-dimensional feature space. To address this issue, we use the radial basis function (RBF) kernel, which is defined as follows:

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \quad \text{for} \quad \sigma > 0, \tag{8}$$

where x and y are input feature vectors, and σ is a parameter that determines the width of the effective basis function, which affects the classification accuracy. In this study, we experimentally set the standard deviation (σ) to 0.1 yielding high classification performance. The input test value x and the support vectors x_i obtained from a training data set are non-linearly mapped features using the RBF kernel. A candidate fire pixel is finally classified as either a real fire pixel if the result is 1 or a non-fire pixel if the result is -1 using (7). To train the SVM, we build a training dataset that includes 200 wavelet energies from training fire pixels and 200 wavelet energies from fire-colored moving pixels, respectively. Table 1 presents a summary of the experimental setup used for the SVM in this study.

2.5 Accuracy evaluation of the multi-stage fire detection scheme

We implement the selected fire detection algorithm in MATLAB 2012b on an Intel Quad-Core 3.4 GHz PC platform. Furthermore, five movies are used for evaluating the accuracy of the fire detection algorithm, including 2,642 samples with dimensions of 256×256 (1,301 samples containing fire and 1,341 samples containing non-fire), as illustrated in Fig. 3.

Table 1 Summary of the experimental setup used for the SVM in this study

Parameters	Values
Number of fire signatures to train the SVM	200
Number of non-fire signatures to train the SVM	200
Kernel	RBF
σ Value for the RBF kernel	0.1
Number of support vectors	1×150
Number of weights	1×150

Table 2 presents accuracy comparison between the proposed multi-stage fire detection approach and four state-of-the-art fire detection approaches in terms of true positives (TP) and false negatives (FN). TP is the number of all frames that correctly detect a real fire as a fire and the percentage of TP (PTP) is the overall fire detection rate. Moreover, the FN is the number of all frames that detect a real fire as a non-fire and the percentage of FN (PTN) is the overall false non-fire detection rate.

As shown in Table 2, the results indicate that average fire detection and false non-fire detection rates of the proposed approach are 99.67 and 3.69 %, respectively, which are good enough for fire detection since they consistently increase the accuracy of fire detection while decreasing the error of false fire detection in all movies. However, the computational complexity of the fire detection algorithm may limit real-time processing for early detection of fire. Table 3 shows the average computational times at each stage for processing all the frames in a movie clip using the proposed multi-stage fire detection scheme. Likewise, Table 4 compares average computational times for processing all movie clips between the proposed fire detection scheme and the others.

As presented in Table 3, MRCD and CS require larger computational times of the reference fire detection algorithm than those of FE and CLASSIFY due to their per-pixel operations. Despite the fact that the proposed fire detection methodology demands the minimum computational time (i.e., the average computational time for all movie clips is about 220 ms), the computational complexity of the proposed approach limits its use in real-time fire detection applications (i.e., 30 frames-per-second). Thus, we propose a SIMD many-core computational model to meet the computational time required for the real-time fire detection.

3 Parallel implementation of the multi-stage fire detection scheme using a many-core model

This section includes the detailed description of the reference many-core model and parallel implementation of the multi-stage fire detection scheme on it.

3.1 Many-core architecture

The microarchitecture of the reference many-core model, shown in Fig. 4, mainly consists of a 2D processing element (PE) array and local memory [44]. In this study, the

Fig. 3 Examples of test movies used in this study

Table 2 Fire detection accuracy of the proposed and other approaches

	Movies (frames)									
	Movie 1 (500)		Movie 2 (599)		Movie 3 (199)		Movie 4 (946)		Movie 5 (393)	
	TP	PTP (%)	TP	PTP (%)	TP	PTP (%)	FN	PFN (%)	FN	PFN (%)
Approach 1 [1]	480	96.00	560	93.49	188	94.47	37	3.91	28	7.12
Approach 2 [5]	480	96.00	574	95.83	186	93.47	28	2.96	26	6.62
Approach 3 [14]	474	94.80	571	95.33	190	95.48	22	2.33	22	5.60
Approach 4 [48]	474	94.80	568	94.82	189	94.97	22	2.33	26	6.62
Proposed	500	100.00	598	99.83	198	99.50	0	0.00	29	7.38

Table 3 Average computational time of each stage of the reference fire detection approach

Stages	Unit	Movie 1	Movie 2	Movie 3	Movie 4	Movie 5
MCRD	[ms per frame]	150.904	149.723	157.701	149.512	155.331
CS		63.127	58.07	58.435	47.876	73.648
FE		6.729	6.425	6.339	5.89	6.233
CLASSIFY		0.437	0.397	0.39	0.398	0.403
OVERALL		221.197	214.615	222.85	203.676	235.615

Table 4 Average computational time per frame using fire detection approaches on an Intel Quad-Core 3.4 GHz PC platform with 256 × 256 resolution videos

	Approach 1 [1]	Approach 2 [5]	Approach 3 [14]	Approach 4 [48]	Proposed
Average computational times (s) per frame	0.36	1.17	0.89	1.1	0.22

PEs execute a set of instructions in lockstep fashion and are interconnected through a mesh network. Furthermore, each PE supports different amounts of data for different PE configurations to store distributed image data and temporary data produced during processing. It has a reduced instruction set computer (RISC) data path with the following characteristics:

- arithmetic logic unit (ALU) performs basic arithmetic and logic operations,
- multiply–accumulator unit (MACC) multiplies 32-bit values and accumulates them into a 64-bit accumulator,
- barrel shifter unit (BSU) performs multi-bit logic/arithmetic shift operations,
- SLEEP unit activates/deactivates PEs based on local information,
- communication unit (COMM) allows PEs to communicate with their four nearest neighbors (north, east, west, and south),
- a small amount of 32-bit word local memory and 16 32-bit three-ported general-purpose registers are included.

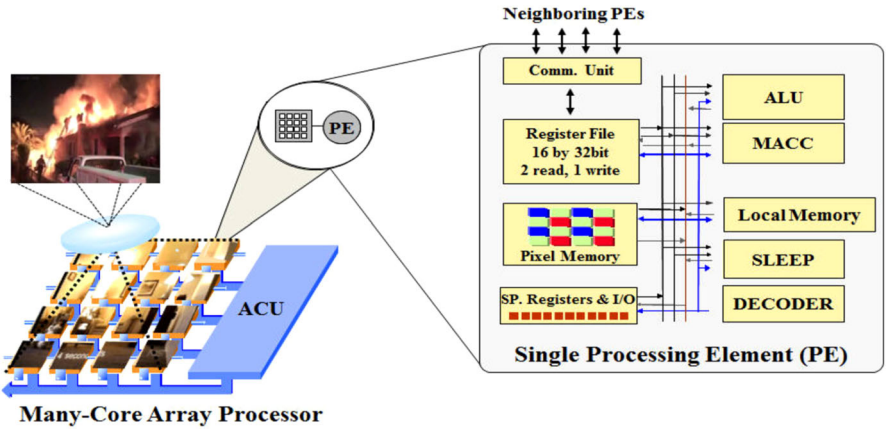

Fig. 4 The reference many-core architecture

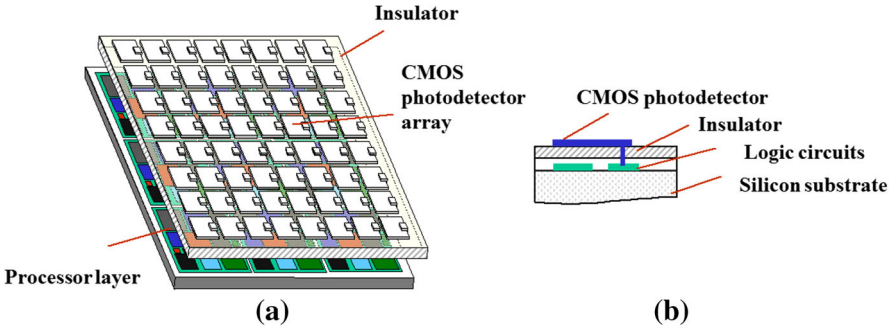

Fig. 5 **a** The tiled portion of the detector array is dedicated to a single PE and **b** the multi-level very-large-scale integration enables CMOS image sensors to be grown on the top of the PEs to maximize the fill factor [44]

A key feature of the reference many-core architecture compared to commercial GPUs is its processing-in-place computation. This reduces data movement since image data are directly transferred into the PEs by utilizing local complementary metal-oxide semiconductor (CMOS) image sensors that provide area-I/O data streams to directly access the PEs. Images are focused on an optical focal plane consisting of an array of optoelectronic devices stacked above the PEs, as illustrated in Fig. 5. This allows an entire image to be sampled and partitioned in parallel, reducing the overhead and distributing the image data to each PE, while providing the bandwidth data required by the fire detection algorithm [49]. The partitioned image data are stored in the pixel memory of each PE for further processing.

The reference many-core model has a three-stage instruction pipeline (i.e., fetch, decode, and execution) as illustrated in Fig. 6. In this three-state instruction pipeline, each stage is completed within one cycle except the MACC, which demands two cycles for its completion. More details about the instruction pipeline of the reference many-core architecture are summarized as follows:

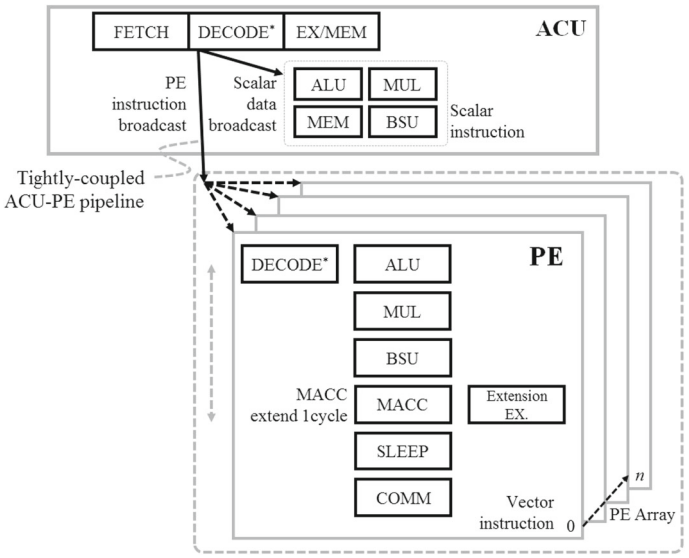

Fig. 6 Three-stage instruction pipeline of the reference many-core model

- At the fetch stage, the array control unit (ACU) fetches an instruction from instruction memory.
- At the decode stage, the instruction decoder of the ACU decodes the instruction and determines if it is a scalar instruction or a vector instruction that is transmitted to PEs.
- At the execution stage, scalar instructions are directly executed in the ACU, whereas vector instructions simultaneously are executed by all the PEs under the control of the ACU, where scalar instructions process the serial portion of the algorithm while vector instructions represent the parallel portion of the algorithm operating in the many-core architecture.

3.2 Parallel implementation using the reference many-core model

As mentioned in Sect. 2, the multi-stage fire detection algorithm is composed of background subtraction-based MCRD, CS for detecting fire-like regions using the YCbCr color model, FE using a normalized wavelet energy, and a SVM-based decision making (CLASSIFY). According to Tables 2, 3, and 4, the multi-stage fire detection algorithm achieves both high fire detection accuracy and a low false non-fire detection rate. However, its computational complexity does not meet real-time processing requirements. Hence, we accelerate this fire detection algorithm by exploiting massive parallelism inherent in it with the power of the reference many-core architecture. In the following sub-sections, we give more details about the parallel implementation of this algorithm on the reference many-core architecture.

A key system design issue of the many-core architecture for multimedia applications is to determine the ideal grain size that yields the highest performance and the

Table 5 A discrete set of the PE configurations and required memory space of each PE used in this study

N_{PE}	16	64	256	1,024	4,096	16,384
IDPE ratios	4,096	1,024	256	64	16	4
Local memory/PE [bytes]	16,384	4,096	1,024	1,024	1,024	1,024
System memory [bytes]	518,144	518,144	518,144	1,304,576	4,450,304	17,033,216

lowest energy consumption. To explore the effects of grain sizes on the many-core architecture, we introduce the IDPE ratio as a design variable that is the amount of image data directly mapped to each PE. In this study, six different IDPE ratios are used for evaluating the system performance and energy efficiency for real-time fire detection. In addition, the local memory sizes vary with the IDPE. As the IDPE ratio decreases, a smaller amount of local memory per PE is needed to store the image and temporary data produced during this process. Thus, each PE configuration is composed of different numbers of PEs and local and pixel memories. The IDPE ratio is computed as S_{IMG}/N_{PE}, where S_{IMG} is the image size (i.e., 256×256 in this study) and N_{PE} is the number of PEs, which is defined as $N_{PE} = 4^{6-i}$, $i = 0, 1, \ldots, 5$. For completing the multi-stage fire detection algorithm, the local memory size per PE required is calculated as:

$$\mathrm{MEM_{PE}} = \begin{cases} \mathrm{IDPE} \times (1 + \frac{8}{\mathrm{BAND_{dwt}}}) + \alpha, & \text{if } N_{PE} < 256 \\ 2 \times (\mathrm{NUM_{sv}} + \mathrm{NUM_{weights}}) + \alpha, & \text{Otherwise} \end{cases}, \tag{9}$$

where $\mathrm{BAND_{dwt}}$ is the number of sub-images resulting from the single-level 2D wavelet decomposition, α is a temporary memory space used to store the computational results during processing, $\mathrm{NUM_{sv}}$ is the number of support vectors, and $\mathrm{NUM_{weights}}$ is the number of weights. As mentioned in Sect. 2.3, we perform the single-level 2D wavelet decomposition to extract a fire signature that represents the nature of turbulent flicker of fire. In addition, the wavelet decomposition can be accomplished by carrying out convolution operations with Daubechies 4 wavelet decomposition filters. In (9), $\mathrm{IDPE} \times \frac{32}{\mathrm{BAND_{dwt}}}[bytes]$ corresponds to the amount of local memory of a PE to store convolution results by the single-level wavelet decomposition, where $\mathrm{BAND_{dwt}}$ is set to 4 in this study. The necessary memory space for CLASSIFY is constant regardless of the PE configuration while the required local memory of each PE for MCRD, CS, and FE varies with the IDPE ratios (or PE configurations). Table 5 presents a discrete set of PE configurations and the minimum local memory of each PE used in this study.

3.2.1 Parallel implementation: movement-containing region detection (MCRD)

Figure 7 depicts MCRD on a single PE of the reference many-core architecture, which consists of the following two steps. Note that $\mathrm{RGB}(i, j)$ in Fig. 7 is the pixel intensity in RGB color space at location (i, j), and $G(i, j)$ is the gray-level pixel intensity at location (i, j). Likewise, $G(\mathrm{BG_{init}}(i, j))$, $G(\mathrm{BG_{prev}}(i, j))$, and $G(\mathrm{BG_{cur}}(i, j))$ are the

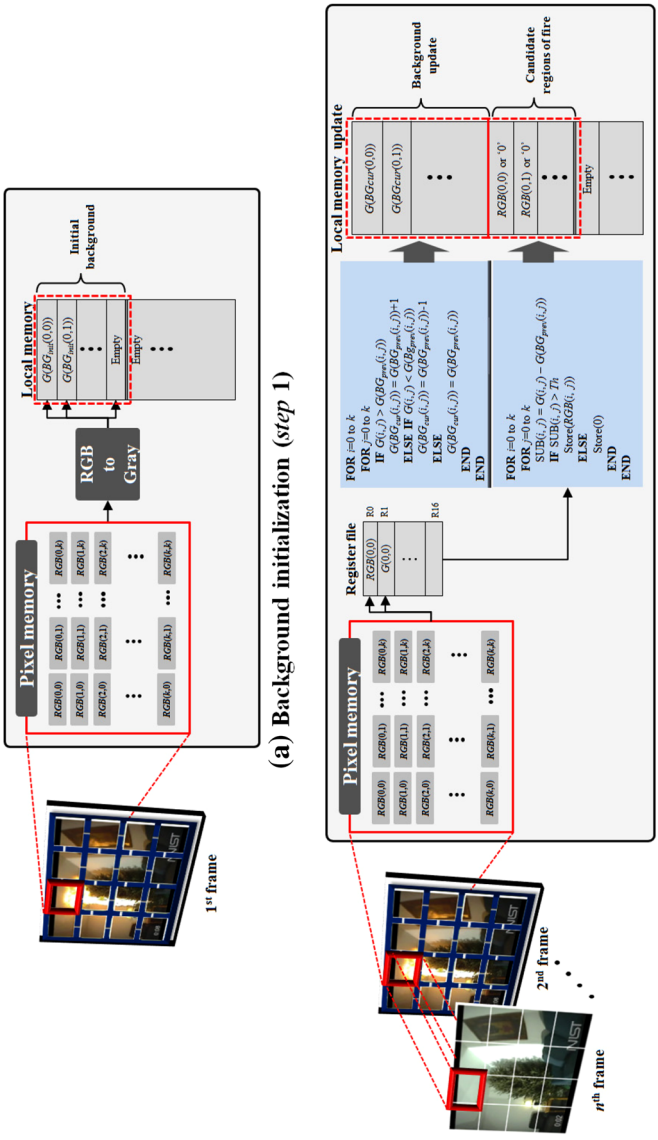

Fig. 7 MCRD on a single PE. **a** Background initialization (step 1) and **b** movement-containing region detection (step 2)

gray-level initial, previous, and current backgrounds at the location (i, j), respectively. Likewise, the index k corresponds to $\sqrt{IDPE}$ in this study.

- **Step 1**: This step is used to generate the initial background and is performed once during the whole process for a movie. It is useful to store the background in the local memory of each PE while all the frames of a movie are being processed; this is because the gray-level background is iteratively updated and is used as a reference for detecting movement-containing objects in the processing frame. In step 1, an initial background is generated by repeating the process of loading a single pixel of the $\sqrt{\text{IDPE}} \times \sqrt{\text{IDPE}}$ partitioned image of the first frame in a movie to a register of each corresponding PE from the pixel memory. In addition, the distributed RGB pixel intensity is converted to a gray-level intensity, and the converted gray-level intensity value is stored to a proper local memory space.
- **Step 2**: This step detects movement-containing regions in the processing frame. Unlike the background, the other frames of the movie do not need to be stored in local memory, which would only result in unnecessary memory usage. Our approach is to load an RGB pixel of the partitioned frame into two registers of each corresponding PE from the pixel memory. One register stores an original RGB intensity value while the other stores a gray-level intensity value of the distributed RGB pixel which is used to determine whether or not it is a movement-containing pixel by calculating an absolute value of the difference of the two intensity values between the gray-level background and the processing frame. In addition, it is compared to a predefined threshold that involves SLEEP instructions (e.g., SGT, SGE, SLT, and SLE), which are used to deactivate PEs based on the results of the comparison. In this study, once a distributed RGB pixel to a PE is determined as a movement-containing pixel, its original RGB pixel intensity in one of the registers is stored in the appropriate local memory space of the PE. Otherwise, a value of 0 is stored in the local memory of the PE. This is repeated until all the distributed RGB pixels to the PE are processed. Due to the data-parallel nature of MCRD, the execution time is expected to decrease linearly with the number of PEs in the many-core system, which is inversely proportional to the IDPE. Likewise, MCRD is able to maintain high system utilization for all PE configurations because of its data-parallel nature.

3.2.2 Parallel implementation: color segmentation (CS)

In this study, we use YCbCr color information for color segmentation and determine whether there are fire-colored objects in the candidate regions after MCRD using (4). Figure 6 describes the process of color segmentation on the reference many-core architecture, which is composed of the following two steps. Note that YCbCr(i, j) in Fig. 8 is the pixel intensity in the YCbCr color space at location (i, j) and $G(\text{BG}_{\text{cur}}(i, j))$ is the gray-level pixel intensity for the current background at location (i, j). Furthermore, k corresponds to $\sqrt{\text{IDPE}}$ in this study.

- **Step 1**: After MCRD on the reference many-core model, movement-containing RGB pixel values are located in the local memory space of PEs and these RGB pixel values are converted into the YCbCr color space. This color conversion is

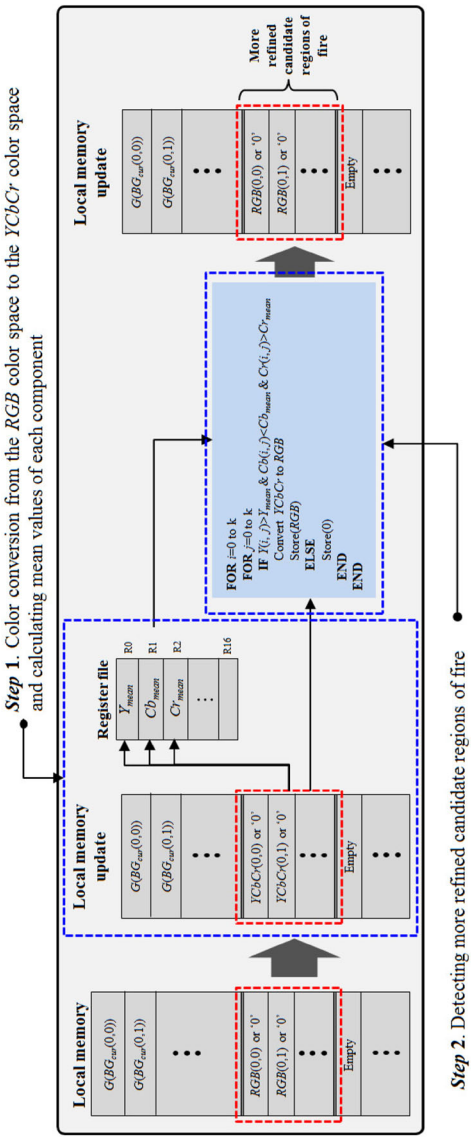

Fig. 8 CS on a single PE

performed by multiplying each of RGB components and the corresponding scalar values and summing the products to predefined values in (3). To execute the color conversion on the many-core architecture, the shift and arithmetic operations (e.g., LSH, MUL, and ADD) are needed. In this study, an arbitrary image pixel after MCRD is considered as a fire-colored pixel by comparing each of the YCbCr components with a corresponding mean value of the components in the YCbCr color space. Thus, it is necessary to calculate cumulative sums of the three components during the color space conversion, and finally to compute their mean values.

- **Step 2**: As mentioned earlier, more refined fire-colored pixels are detected based on the condition in (4), and this requires SLEEP instructions (e.g., SGT, SGE, SLT, and SLE) to deactivate PEs according to a result of comparison between each of the YCbCr components and the mean values of the components in this study. Once a movement-containing pixel in the YCbCr color space is determined as a fire-colored pixel using (4), we reconvert the pixel value into RGB format for further processing. Otherwise, we replace the pixel value with 0, which indicates that the pixel is a spurious fire-like pixel. Like MCRD, color segmentation is highly data parallel. Thus, high system utilization is maintained across the IDPE ratios.

3.2.3 Parallel implementation: feature extraction (FE)

Using the more refined fire-colored pixels, we extract fire signatures by calculating the normalized wavelet energy for high-frequency sub-images resulting from the single-level wavelet decomposition, which represents color variations due to the nature of turbulent fire flicker. As described in Sect. 2.3, we used the single-level wavelet decomposition by convolving with Daubechies 4 low-pass and high-pass decomposition filters in the rows and columns of the entry. During this process, inter-PE communication instructions (e.g., XFER NORTH, XFER EAST, XFER WEST, and XFER SOUTH) are needed to compute wavelet coefficients around the boundary of the $\sqrt{\text{IDPE}} \times \sqrt{\text{IDPE}}$ partitioned image. Accordingly, as the IDPE increases (or the number of PEs decreases), a lower percentage of inter-PE communication instructions is required. In addition, arithmetic instructions (MUL and ADD) are highly used regardless of the IDPE. Figure 9 gives an example of the single-level wavelet decomposition on the reference many-core model. Since each PE in Fig. 9 concurrently performs convolution operations with Daubechies 4 decomposition filters, it obtains sub-images (i.e., LL, LH, HL, and HH) for the $\sqrt{\text{IDPE}} \times \sqrt{\text{IDPE}}$ partitioned image after color segmentation. Thus, data rearrangement is needed to locate sub-images resulting from the $\sqrt{\text{IDPE}} \times \sqrt{\text{IDPE}}$ partitioned image in the upper left (LL), upper right (LH), lower left (HL), and lower right (HH) quadrants, and this requires a tremendous number of inter-PE communication instructions. Figure 10 illustrates more details about the data rearrangement process for completing the single-level wavelet decomposition.

Once the single-level wavelet decomposition process is completed, we finally calculate the normalized wavelet energy by activating PEs containing high-frequency sub-images (i.e., LH, HL, and HH). This requires a large number of arithmetic instructions (i.e., MUL and ADD) to compute local wavelet energies as well as communication instructions to broadcast the local wavelet energies to neighboring PEs. Despite the fact that a number of communication and SLEEP instructions are needed to complete

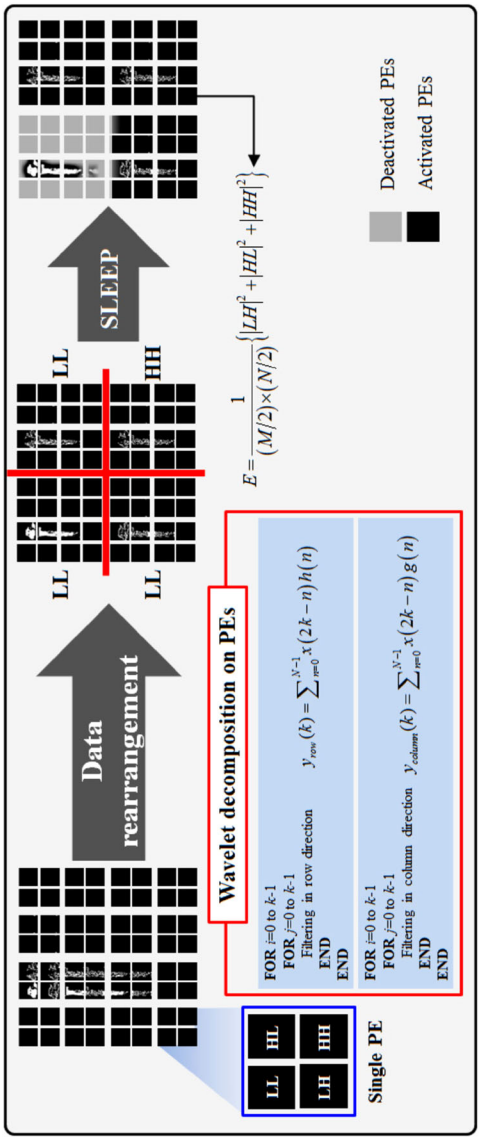

Fig. 9 Single-level wavelet decomposition on the reference many-core model, where k corresponds to $\sqrt{\text{IDPE}}$

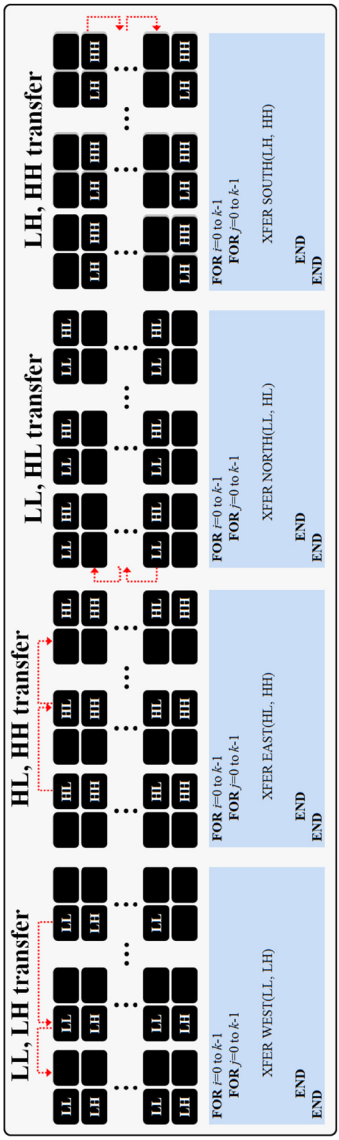

Fig. 10 Data rearrangement process for completing the single-level wavelet decomposition, where k is $\sqrt{\text{IDPE}}$

the single-level wavelet transform, the execution time is expected to decrease with the number of PEs in the system due to the data-parallel nature of wavelet decomposition.

3.2.4 Parallel implementation: a SVM-based decision making (CLASSIFY)

To classify a candidate fire pixel as either a real fire-containing pixel or a non-fire-containing pixel, we use the SVM with the RBF kernel to deal with non-linear property. However, the reference many-core architecture does not involve special units to accelerate transcendental instructions such as square roots and exponential functions. Thus, we use a Taylor series for the exponential term of the RBF kernel in this study:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}, \tag{10}$$

where x is an input of the exponential function and $n!$ is the denominator in the infinite sum, where the n is the only parameter to be determined. Although larger values of n offer more accurate results, it is necessary to select a suitable value of n to avoid high computational complexity. In this study, we introduce an evaluation metric which is a normalized error to select an appropriate value of n satisfying a certain level of the difference between $\mathrm{EXP}_{\mathrm{expfunc}}$ and $\mathrm{EXP}_{\mathrm{taylor}}$, where $\mathrm{EXP}_{\mathrm{expfunc}}$ is an exponential value calculated from the exponential function, exp(), in MATLAB and $\mathrm{EXP}_{\mathrm{taylor}}$ is an exponential value computed using (10). The evaluation metric, Error, is defined as follows:

$$\mathrm{Error} = \left| \frac{\mathrm{EXP}_{\mathrm{expfunc}} - \mathrm{EXP}_{\mathrm{taylor}}}{\mathrm{EXP}_{\mathrm{expfunc}}} \right|. \tag{11}$$

According to our experimental results, the difference between input patterns and support vectors is mostly in the range from -20 to -10. Thus, we set the input to the exponential function as a value in the range from -20 to -19 at intervals of 0.2 and measure error using (11) while varying n. Figure 11 shows the experimental results, and we select the minimum value of n which is 68 in this study.

Although the SVM is implemented on the many-core architecture, the speedup can be lower than that of MCRD and FE since it is difficult to exploit the massive parallelism inherent in the SVM. Figure 12 pictorially illustrates how the SVM works on the many-core architecture, where each PE calculates the inner term of Σ in (7) and transfers its own result to the left-top PE by communicating with its neighboring PEs. The left-top PE finally classifies a candidate fire pixel as a real fire or non-fire by summarizing all results from the other PEs and determining if the summarized result is either +1 or -1. In this study, we obtain 150 support vectors (i.e., the dimension of x_i in (7) is 150) as well as 150 weights (i.e., the dimension of w_i in (7) is 150), and a value for the bias term from the training data. Thus, we can achieve the highest parallelism when all 150 PEs are utilized. Each PE computes $w_i * k(x, x_i) + b$ in (7) with corresponding parameters (e.g., corresponding support vectors and weights) that are already stored in the local memory of the PEs. This results in lower system utilization when the number of PEs exceeds 64 (e.g., PEs = 256, 1,024, 4,096, and 16,384)

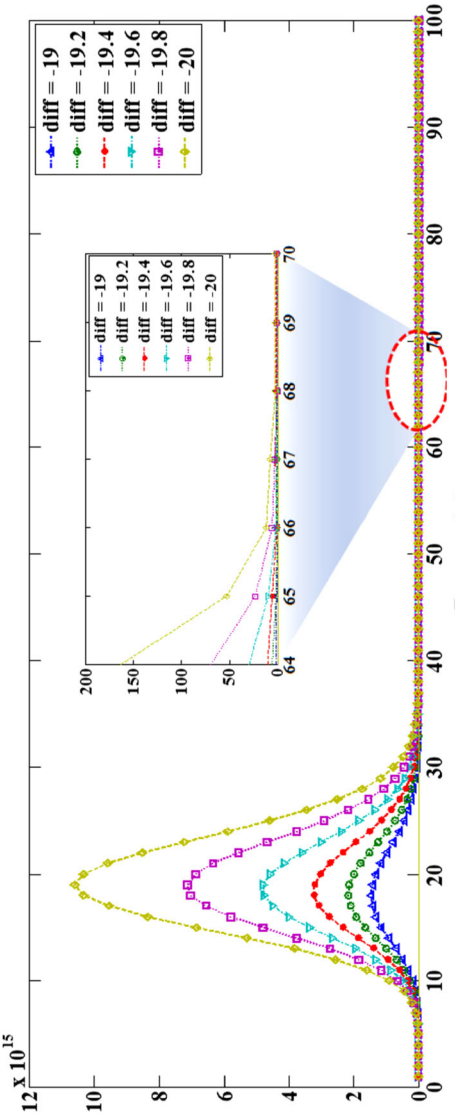

Fig. 11 Measuring errors with varying numbers of n for certain input values. The '*diff*' denotes the difference between the input patterns and support vectors, which assumes that the differences are in the range from −20 to −19 at 0.2 intervals. The 'diff' is utilized as an input of the exponential function

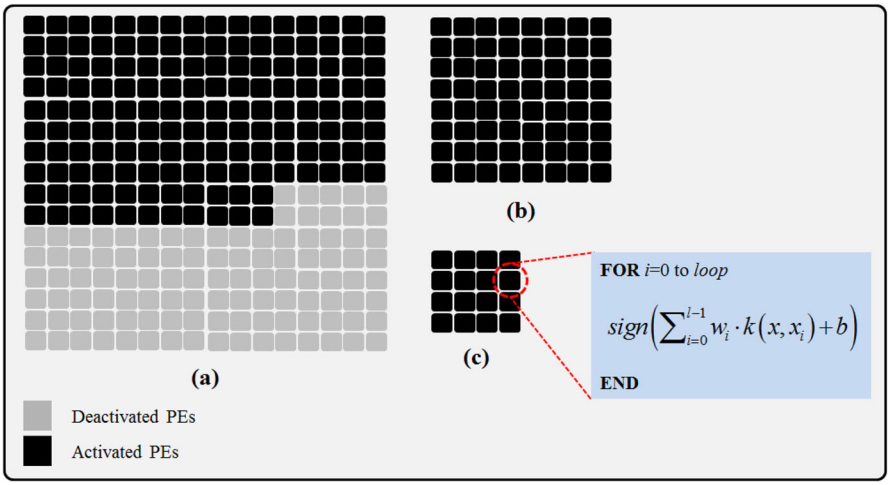

Fig. 12 The SVM on the reference many-core architecture. **a** $N_{PE} = 256$, **b** $N_{PE} = 64$, and **c** $N_{PE} = 16$, where the index '*loop*' is the dimension of support vectors

while yielding higher execution performance to perform the SVM on the many-core architecture.

4 Experimental results

To identify ideal grain sizes of the reference many-core model, this study explores the impact of IDPE ratios in terms of execution time and energy efficiency. In this section, analytical results are given below.

4.1 Simulation methodology and evaluation metrics

In Fig. 13, a simulation methodology infrastructure in this study is divided into three levels: application, architecture, and technology. At the application level, an instruction-level many-core simulator (see Fig. 13a) is utilized to profile execution statistics such as the cycle count, dynamic instruction frequency, and system utilization for six different PE configurations by retargeting the fire detection algorithm for each configuration based on the architecture and its execution properties. At the architectural level, the Xilinx ISE Design Suite 14.2 is used to generate register–transfer level (RTL) code for each function unit of the reference many-core architecture and to validate its functionality. At the technology level, RTL-to-gates synthesis is carried out by the Synopsys Design Compiler with TSMC 28 nm technology. As a result of TRL-to-gates synthesis, we obtain power consumption of each functional unit of the many-core architecture. Further, a design space analysis tool collects and combines all the parameters obtained from the application, architecture, and technology levels to determine execution time and energy efficiency for each PE configuration.

In this study, we select execution time and energy efficiency as the evaluation metrics for determining an optimal design point among six different configurations,

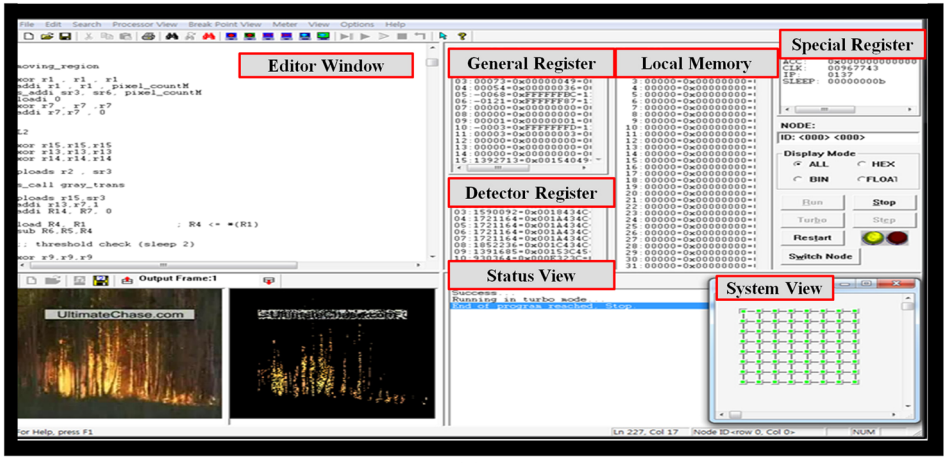

Fig. 13 **a** A screenshot of the instruction-level many-core simulator, **b** simulation methodology infrastructure, and **c** an example of RTL schematic of a single PE using the Xilinx ISE tool

Table 6 Summary of evaluation metrics

Execution time	$t_{\text{exec}} = \frac{N_{\text{executed}}}{f_{clk}}$
System utilization	$U = \frac{N_{\text{executed}}}{N_{\text{issued}}} \times 100\ [\%]$
Energy efficiency	$\eta_E = \frac{1}{t_{\text{exec}} \times \text{Energy}} \left[\frac{1}{\text{s}\cdot\text{Joules}}\right]$

N_{executed} is the total number of executed instructions on each PE, N_{issued} is the total number of issued instructions, f_{clk} is the clock frequency, and energy is the system energy required to complete the reference fire detection algorithm in 28 nm CMOS technology

which delivers real-time execution of the fire detection algorithm while maximizing energy efficiency. Reduction in energy requirements are as important as improved performance (or execution time) for portable electronic products. In addition, efficient system utilization is a dominant concern for the design of portable supercomputer design. Table 6 defines these evaluation metrics for the many-core model.

4.2 Execution time

Figure 14 shows average execution time per frame of all the movies for each task of the multi-stage fire detection algorithm versus IDPE. As expected, the execution time of each task monotonically decreases with the IPDE due to the increase in available PEs and the data-parallel nature of each task. However, the slopes of the execution time, which indicate speedup efficiency, are not equal. As mentioned in Sects. 3.2.1, 3.2.2, and 3.2.3, MCRD, CS, and FE are highly data-parallel. Thus, the execution time of these tasks linearly decreases as the IDPE decreases (or as the number of PEs increases). As discussed in Sect. 3.2.4, CLASSIFY generally requires a small number of arithmetic and shift operations compared to those of MCRD, CS, and FE. Thus, the slope of the execution time for CLASSIFY is almost flat for all of the PE configurations. To meet real-time requirements for online fire detection, a single movie frame should be processed within 33 ms (30 frames-per-second). The reference many-core architecture satisfies this requirement with all of the IDPE ratios.

4.3 System utilization

To achieve the level of performance required by its intended application domain, it is critical to maintain high system utilization. A presented in Table 6, system utilization is defined as the ratio between the number of active PEs and the total number of PEs to complete a given task and Fig. 15 illustrates average system utilization of all the movies at different IDPE ratios. As the IDPE ratios vary, MCRD, CS, and FE maintain higher system utilization than CLASSIFY, as shown in Fig. 15 by exploiting massive parallelism inherent in them. In addition, we observe that for IDPE <256, a decrease in system utilization for FE occurs due to the increased impact of communication overhead between collaborating neighboring PEs. Likewise, the system utilization for

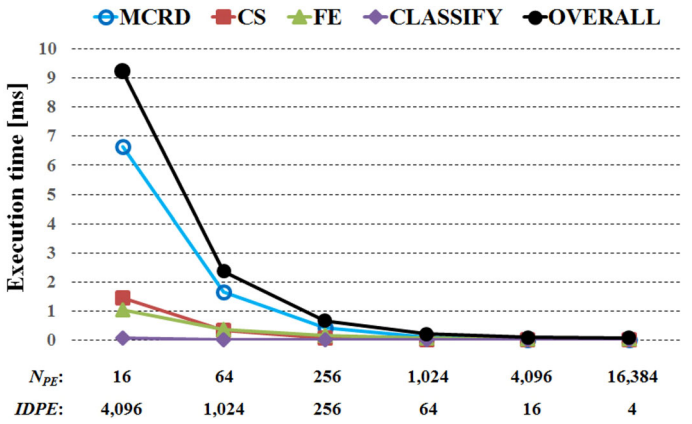

Fig. 14 Average execution time per frame of all the movies for each task of the multi-stage fire detection algorithm with different PE configurations

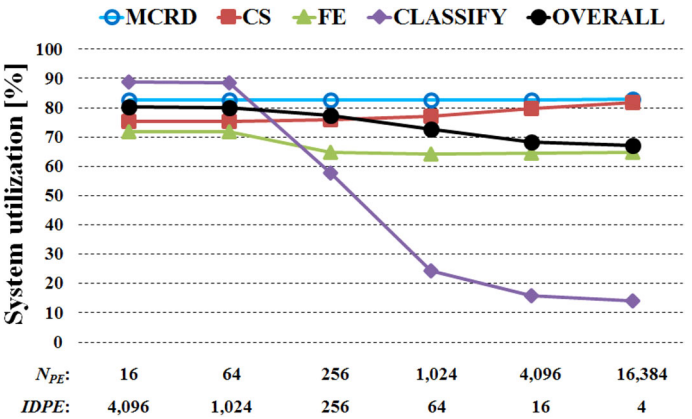

Fig. 15 Average system utilization of all the movies at different IDPE ratios

CLASSIFY dramatically decreases at IDPE <1,024 due to the nature of the algorithm as described in Sect. 3.2.4.

4.4 Energy efficiency

Energy efficiency is the task throughput achieved per unit of Joule. Figure 16 shows energy efficiency for each task of the multi-stage fire detection algorithm with all the movies used in this study. Increasing energy efficiency equivalently implies minimizing power dissipation, which translates into minimizing the energy per operation.

Since the purpose of this study is to identify the most efficient design point that yields the highest performance as well as energy efficiency, the shape of the energy efficiency curves is significant. In practice, energy efficiency can vary with the content of movie frames, and it correlates with the execution time and energy consumption.

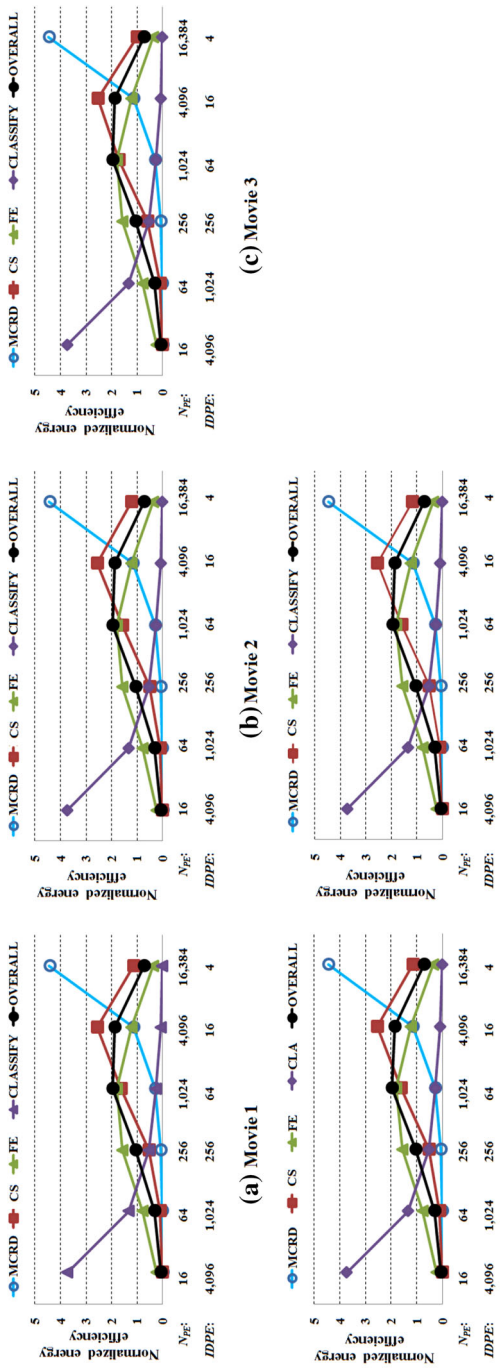

Fig. 16 Energy efficiencies of each task of the multi-stage fire detection algorithm with five different movie clips

To analyze energy efficiency, we exploit average execution time to complete all the movies at different PE configurations. An interesting observation in Fig. 16 is that the shape of curves is very similar. Although the energy consumption varies per frame as shown in Fig. 17, the variation in energy consumption is very small, which cannot affect the shape of the energy efficiency curves in Fig. 16. As expected, more energy consumption of the reference many-core architecture is required to process Movies 1–3 because these movies include fire pixels and thus require additional operations such as color conversion and inter-PE communications to broadcast temporary results. Meanwhile, Movie 4 is mainly composed of scenes including a red-colored car, and the fire detection algorithm recognizes these red-colored objects as candidate regions of fire, requiring high energy consumption at the frame including the red-colored objects. In Movie 5, both the fluorescent light in tunnel and car brake lights are candidate regions of fire. However, a bus slips on the road between the 100th frame and the 120th frame, and the scenery takes a large portion of those frames, which results in reducing the candidate regions of fire. After the 120th frame in Movie 5, the brakes of a red-colored car are shown for several frames and this causes high energy consumption of the many-core system.

Overall, the most efficient design point is achieved at 64 IDPE (or $N_{\mathrm{PE}} = 1{,}024$) in terms of computational performance and energy efficiency.

4.5 Performance comparison with a commercial GPGPU

Our analyses indicate that 64 IDPE (or 1,024 PEs) provides the most efficient operation for the multi-stage fire detection algorithm in terms of execution time and energy efficiency. Table 7 compares the performance of the selected optimal many-core configuration and that of the Nvidia GeForce GTX 480 commercial GPU system, which includes 480 processing cores. Even though a comparison between this optimal many-core configuration and the commercial GPU involves unavoidable errors, the purpose of this study is to show the potential of the most efficient many-core configuration enabling real-time fire detection rather than to conduct a precise performance comparison.

To implement the multi-stage fire detection algorithm on the GPU, we designed four kernels as follows: mcrdKernel for movement-containing region detection, csKernel for color segmentation, feKernel for fire feature extraction, and classifyKernel for decision making if there is a fire or a non-fire in a processing frame in a movie clip. Since both mcrdKernel and csKernel are fully parallelizable, this GPU-based parallel implementation greatly improves performance. However, both feKernel and classifyKernel are not parallelizable on the GPU. For example, the feKernel should calculate the sum of wavelet energies for high-frequency sub-images resulting from the single-level wavelet decomposition to calculate the normalized wavelet energy. In this study, we exploited a parallel reduction that builds a balanced binary tree of the input data and sweeps it from the top over data addition to exploit massive parallelism when calculating the sum of wavelet energies with utilization of the shared memory access. In addition, both execution time and energy efficiency are averaged over the five movies for the purpose of performance comparison.

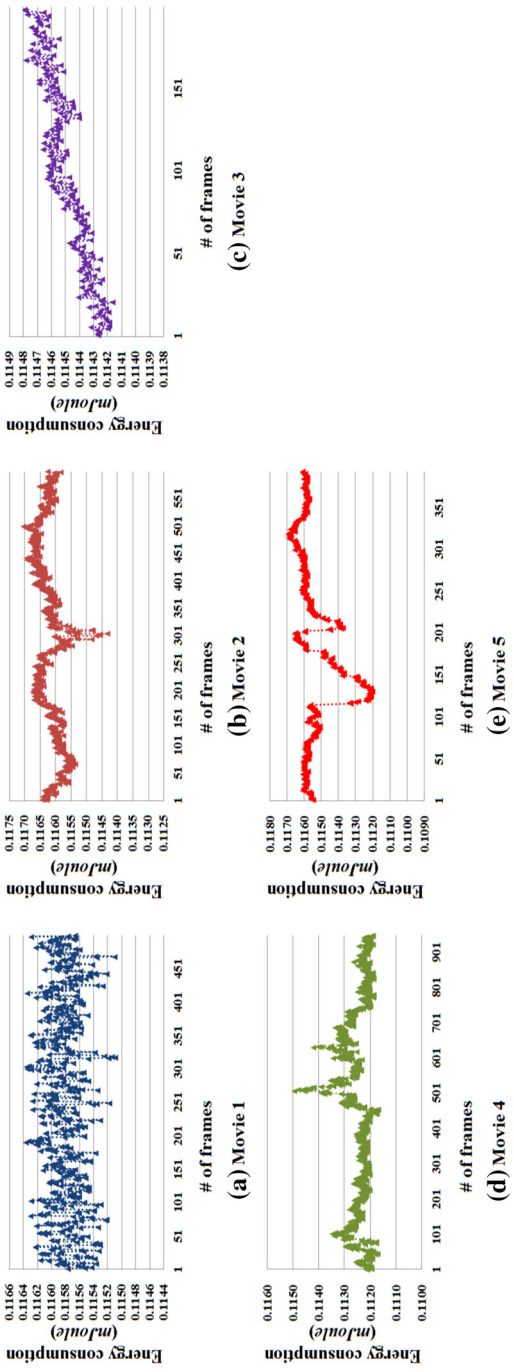

Fig. 17 Energy consumption per frame of all the movies

Table 7 Performance comparison between the optimal many-core configuration and Nvidia GPU for the multi-stage fire detection algorithm

Parameters	Units	NVIDIA GeForce GTX 480 running at 1,401 MHz	Proposed many-core running at 400 MHz
Execution time	[ms]	2.5966	0.69
Average power	[Watts]	98	0.14
Energy efficiency	[1/(s · Joules)]	1,513.435	15,002,850.5

The experimental result indicates that the optimal many-core processor outperforms the GPU in terms of execution time and energy efficiency: 3.8- and 9,913-fold better than the GPU in execution time and energy efficiency, respectively.

4.6 A prototyping system for the multi-stage fire detection

Figure 18a shows a block diagram of the prototyping system that includes 16 PEs, a camera controller, a liquid crystal display (LCD) controller, and a direct memory access (MDA) unit, and Fig. 18b illustrates a snapshot of the prototyping system using the HUINS system-on-a-chip (SoC) master 3 board. This prototyping system executes the fire detection algorithms as follows:

- **Step 1**: The camera controller transfers image data from the CMOS camera to image memory using a memory write operation;
- **Step 2**: The DMA unit accesses image data and transfers these data to PE's pixel memory;
- **Step 3**: The ACU fetches and decodes both scalar and vector instructions in the program memory and distributes the vector instructions to PEs;
- **Step 4**: PEs execute the given fire detection application in parallel and the processed image data are stored in the PE's pixel memory;
- **Step 5**: The LCD controller finally transfers the processed image data to the thin-film-transistor (TFT)-LCD.

In particular, Table 8 presents design parameters to implement 16 PEs in the Xilinx Virtex-4 FPGA device and Table 9 shows its synthesis results using the Xilinx ISE Design Suite 14.2.

5 Conclusions

The demand for high-performance and low-power fire detection systems has grown rapidly in recent years and has motivated research on many-core architectures including GPUs. In this study, we present the multi-stage fire detection approach that consists of background subtraction-based MCRD, CS for detecting fire-like regions using the YCbCr color model, FE using the normalized wavelet energy, and a SVM-based decision making to identify a fire in a processing frame of a movie clip. While this multi-stage fire detection algorithm achieves both fire detection and false non-fire detection

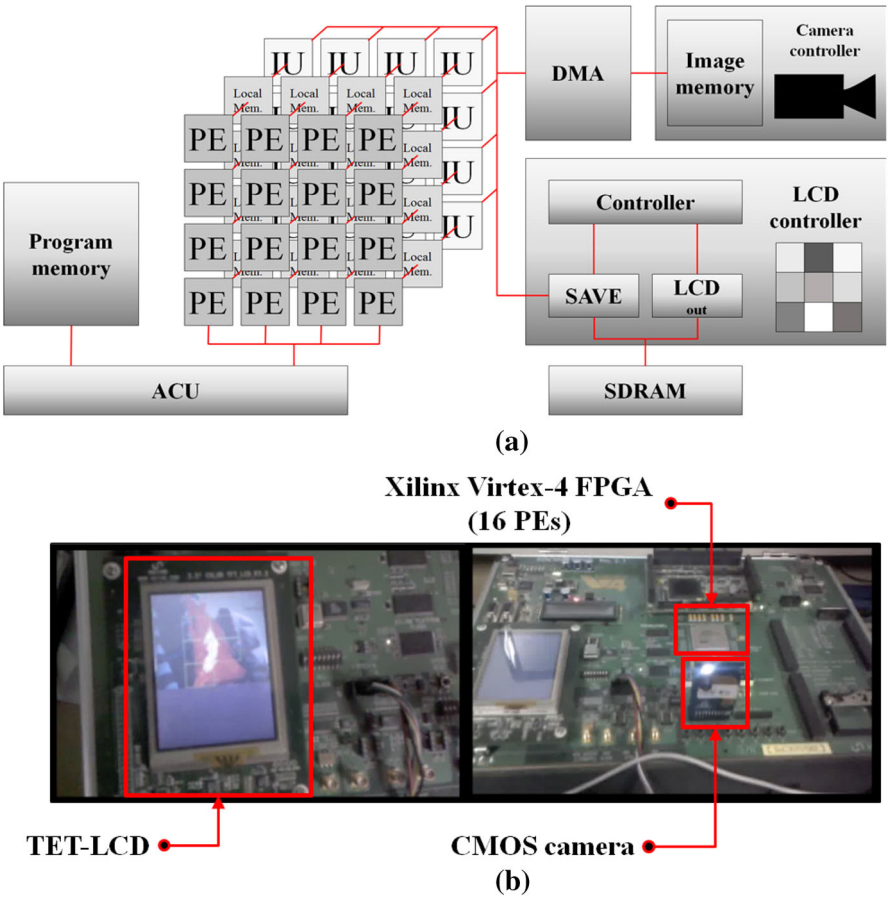

Fig. 18 **a** A block diagram of the prototyping system including 16 PEs, a camera controller, a LCD controller, and a DMA unit for the fire detection algorithm, **b** a snapshot of the multi-core prototyping system using the HUINS SoC master 3 board

Table 8 Design parameters of the multi-core system including 16 PEs

Design parameters	Values
Image size	256 × 256
System size	16 PEs
Number of pixel per PE	60 × 50
Clock frequency	50 MHz
Interconnection network	Mesh
Numbers of integer ALU/integer MACC/barrel shifter/SLEEP unit/integer communication unit	1/1/1/1/1
Pixel memory size per PE	8 kbytes
Local memory size per PE	16 kbytes

Table 9 Synthesis results of the multi-core system for online fire detection

	Used (utilization)	Available
Number of occupied slices	9,123 (34 %)	26,624
Total number of four input look-up tables (LUTs)	16,171 (30 %)	53,248
Total number of slice registers	2,740 (5 %)	53,248
Number of bonded IOBs	132 (20 %)	640
Number of RAMB16s	157 (98 %)	160
Number of DSP48s	20 (31 %)	64
Number of BUFG	3 (9 %)	32

rates of 99.67 and 3.69 %, respectively, its average execution time is 213.95 ms per frame, which limits its use in real-time applications (33 ms or 30 frames-per-second). Thus, this paper proposes the SIMD many-core approach to support this computationally complex algorithm. In addition, this study explores the effects of various IDPE ratios in terms of system performance and energy efficiency. In this study, six IDPE ratios are used to identify the most efficient PE configuration among them, which provides the highest performance as well as the highest energy efficiency enabling online fire detection while maximizing energy efficiency. Experimental results indicate that IDPE = 64 (or $N_{PE} = 1{,}024$) provides the most efficient operation for the fire detection algorithm. In addition, the optimal many-core model outperforms a commercial GPU in terms of execution time and energy efficiency. To guarantee the superiority of the optimal many-core model for fire detection, we will further compare the performance and efficiency with FPGA and customized design, DRRA, and ASIC in near future.

Acknowledgments This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea government (MEST) (No. NRF-2013R1A2A2A05004566).

References

1. Celik T, Demirel H (2009) Fire detection in video sequences using a generic color model. Fire Safety J 44(2):147–158
2. Qiu T, Yan Y, Lu G (2012) An autoadaptive edge-detection algorithm for flame and fire image processing. IEEE Trans Instrum Meas 61(5):1486–1493
3. Ko BC, Ham SJ, Nam JY (2011) Modeling and formalization of fuzzy finite automata for detection of irregular fire flames. IEEE Trans Circuits Syst Video Technol 21(12):1903–1912
4. Li M, Xu W, Xu K, Fan J, Hou D (2013) Review of fire detection technologies based on video image. J Theo Appl Inf Technol 49(2):700–707
5. Ko BC, Cheong KH, Nam JY (2009) Fire detection based on vision sensor and support vector machines. Fire Safety J 44(3):322–329
6. Jin H, Zhang RB (2009) A fire and flame detecting method based on video. In: Proceedings of 2009 International Conference on Machine Learning Cybernetics Baoding, pp 2347–2352
7. Rinsurongkawong S, Ekpanyapong M, Dailey MN (2012) Fire detection for early fire alarm based on optical flow video processing. In: Proceedings of 2012 9th International Conference Electrical Engineering Electronics Computer Telecommunications and Information Technology, Phetchaburi, pp 1–4

8. Dedeoglu Y, Toreyin BU, Gudukbay U, Cetin AE (2005) Real-time fire and flame detection in video. In: Proceedings of IEEE International Conference Acoustic Speech Signal Processing, Philadelphia, pp 669–672
9. Wang H, Finn A, Erdinc O, Vincitore A (2013) Spatial-temporal structural and dynamics features for video fire detection. In: Proceedings 2013 IEEE Workshop on Applications of Computer Vision, Clearwater Beach, pp 513–519
10. Chen J, He Y, Wang J (2010) Multi-feature fusion based fast video flame detection. Bldg Envir 45(5):1113–1122
11. Gunay O, Tasdemir K, Toreyin BU, Enis A (2010) Fire detection in video using LMS based active learning. Fire Technol 463:551–577
12. Kolesov I, Karasev P, Tannenbaum A, Haber E (2010) Fire and smoke detection in video with optimal mass transport based optical flow and neural networks. In: Proceedings 2010 IEEE International Conference Image Processing, Hong Kong, pp 761–764
13. Ko BC, Cheong K-H, Nam J-Y (2010) Early fire detection algorithm based on irregular patterns of flames and hierarchical Bayesian networks. Fire Safety J 45(4):262–270
14. Borges PVK, Izquierdo E (2010) A probabilistic approach for vision-based fire detection in videos. IEEE Trans Circuits Syst Video Technol 20(5):721–731
15. Hamme DV, Veelaert P, Philips W, Teelen K (2010) Fire detection in color images using Markov random fields. Lect Notes Comput Sci 6475:88–97
16. Celik T (2010) Fast and efficient method for fire detection using image processing. ETRI J 32(6):881–890
17. Truong TX, Kim J-M (2012) Fire flame detection in video sequences using multi-stage patten recognition techniques. Eng Appl AI 25(7):1365–1372
18. Morerio P, Marcenaro L, Regazzoni CS, Gera G (2012) Early fire and smoke detection based on colour features and motion analysis. In: Proceedings 2012 IEEE International Conferene Image Processing, Orlando, pp 1041–1044
19. Santana P, Gomes P, Barata J (2012) A vision-based system for early fire detection. In: Proceedings 2012 IEEE International Conference Sytsems, Man and Cybernetics, Seoul, pp 739–744
20. Kang M, Tung TX, Kim J-M (2013) Efficient video-equipped fire detection approach for automatic fire alarm systems. Opt Eng 52(1):1–9. doi:10.1117/1.OE.52.1.017002
21. Lee B, Han D (2007) Real-time fire detection using camera sequence image in tunnel environment. Lect Notes Comput Sci 4681:1209–1220
22. Ha C, Hwang U, Jeon G, Cho J, Jeong J (2006) Vision-based fire detection algorithm using optical flow. In: Proceedings 2012 International Conference Comput Intell. Softw. Int. Syst., Palermo, pp 526–530
23. Toreyin BU, Dedeoglu Y, Gudukbay UI, Cetin AE (2006) Computer vision based method for real-time fire and flame detection. Pattern Recogn. Lett. 27(1):49–58
24. Celik T, Demirel H, Ozkaramanli H, Uyguroglu M (2007) Fire detection using statistical color model in video sequences. J Vis Commun Image Rep 18(2):176–185
25. Wang Y, Wang D, Shi G, Zhong X (2011) GPR simulation for the fire detection in ground coal mine using FDTD Method. Commun Comput Inf Sci 159:310–314
26. Qi X, Ebert J (2009) A computer vision based method for fire detection in color videos. Int J Imaging 2(9):22–34
27. Marbach G, Loepfe M, Brupbacher T (2006) An Image processing technique for fire detection in video images. Fire Safety J 41(4):285–289
28. Hablboglu YH, Gunay O, Cetin AE (2012) Covariance matrix-based fire and flame detection method in video. Mach Vis Appl 23(6):1103–1113
29. Zhao J, Zhang Z, Han S, Qu C, Yuan Z, Zhang D (2011) SVM based forest fire detection using static and dynamic features. Comput Sci Inf Syst 8(3):821–841
30. Wang H, Li D, Wang Y, Yang W (2010) Fire detecting technology of information fusion using support vector machines. In: Proc. 2010 Intl. Conf. AI Comput. Intell., Sanya, pp 194–198
31. Nguyen T, Kim J (2013) Multistage optical smoke detection approach for smoke alarm systems. Opt Eng 52(5):1–12
32. Wu X, Jhang JQ, Huang X, Liu DL (2012) GPU-accelerated real-time IR smoke screen simulation and assessment of its obscuration. IR Phys Technol 55:150–155
33. Park IK, Singhal N, Lee MH, Cho S, Kim CW (2011) Design and performance evaluation of image processing algorithms on GPUs. IEEE Trans Parallel Distrib Syst 22(1):91–104

34. Kim Y, Kang M, Kim J-M (2013) Exploration of optimal many-core models for efficient image segmentation. IEEE Trans Image Process 22(5):1767–1777
35. Choi J, Kang M, Kim Y, Kim C-H, Kim J-M (2013) Design space exploration in many-core processors for sound synthesis of plucked string instruments. J Parallel Distrib Syst 73(11):1506–1522
36. Arabnia HR, Oliver MA (1989) A transputer network for fast operations on digitised images. Int J Eurogr Assoc 8(1):3–12
37. Arabnia HR (1990) A parallel algorithm for the arbitrary rotation of digitized images using process-and-data-decomposition approach. J Parallel Distrib Syst 10:188–192
38. Bhandarkar SM, Arabnia HR (1995) The REFINE multiprocessor—theoretical properties and algorithms. Parallel Comput 21:1783–1805
39. Arabnia HR, Smith JW (1993) A Reconfigurable interconnection network for imaging operations and its implementation using a multi-stage switching box. In: Proc. Calgary, June, New Horizons Supercomput. Symp., pp 349–357
40. Hamzacebi H (2011) CUDA based implementation of flame detection algorithms in day and infrared camera videos. M.S. thesis, Dept. Elect. Electron. Engr., Bilkent Univ., Ankara, Turkey
41. Xiao S, Feng W-C (2010) Inter-block GPU communication via fast barrier synchronization. In: Proc. 2010 IEEE Intl. Symp. Parallel Distrib. Process Atlanta 19–23:1–12
42. Feng WC, Xiao S (2010) To GPU synchronize or not GPU synchronize? In: Proc. 2010 IEEE Intl. Symp. Circuits Systs., Paris, pp. 3801–3804
43. Lee J, Sathisha V, Schulte M, Compton K, Kim NS (2011) Improving throughput of power-constrained GPUs using dynamic voltage/frequency and core scaling. In: Proc. 2011 Intl. Conf. Parallel Arch. Comp. Techn., Galveston, pp 111–120
44. Gentile A, Sander S, Wills L, Wills S (2004) The impact of grain size on the efficiency of embedded SIMD image processing architectures. J Parallel Distrib Comput 64:1318–1327
45. Patel P, Tiwari S (2012) Flame detection using image processing technique. Int J Comput Appl 58(18):13–16
46. Kurup AR (2012) Vision based fire flame detection system using optical flow features and artificial neural network. Int J Sci Res (article ID OCT14677) pp 2161–2168
47. Millan-Garcia L, Sanchez-Perez G, Nakano M, Toscano-Medina K, Perez-Meana H, Rojas-Cardenas L (2012) An early fire detection algorithm using IP cameras. Sensors 12(15):5670–5686
48. Toreyin BU et al (2006) Computer vision-based method for real-time fire and flame detection. Pattern Recog Lett 27(1):49–58
49. Gamal AE, Eltoukhy H (2005) CMOS image sensors. IEEE Trans. Circuits Devices Mag 21(3):6–20