IEICE TRANS. INF. & SYST., VOL.E101–D, NO.12 DECEMBER 2018

PAPER Special Section on Parallel and Distributed Computing and Networking

Real-Time and Energy-Efficient Face Detection on CPU-GPU Heterogeneous Embedded Platforms*

Chanyoung OH[†], **Saehanseul YI**[†], ***Nonmembers***, ***and*** **Youngmin YI**[†a)], ***Member***

SUMMARY As energy efficiency has become a major design constraint or objective, heterogeneous manycore architectures have emerged as mainstream target platforms not only in server systems but also in embedded systems. Manycore accelerators such as GPUs are getting also popular in embedded domains, as well as the heterogeneous CPU cores. However, as the number of cores in an embedded GPU is far less than that of a server GPU, it is important to utilize both heterogeneous multi-core CPUs and GPUs to achieve the desired throughput with the minimal energy consumption. In this paper, we present a case study of mapping LBP-based face detection onto a recent CPU-GPU heterogeneous embedded platform, which exploits both task parallelism and data parallelism to achieve maximal energy efficiency with a real-time constraint. We first present the parallelization technique of each task for the GPU execution, then we propose performance and energy models for both task-parallel and data-parallel executions on heterogeneous processors, which are used in design space exploration for the optimal mapping. The design space is huge since not only processor heterogeneity such as CPU-GPU and big.LITTLE, but also various data partitioning ratios for the data-parallel execution on these heterogeneous processors are considered. In our case study of LBP face detection on Exynos 5422, the estimation error of the proposed performance and energy models were on average −2.19% and −3.67% respectively. By systematically finding the optimal mappings with the proposed models, we could achieve 28.6% less energy consumption compared to the manual mapping, while still meeting the real-time constraint.

key words: *CPU-GPU heterogeneous execution, performance and energy estimation, task mapping, face detection*

1. Introduction

In this dark silicon era, heterogeneous manycore architecture has become a mainstream in order to achieve better energy efficiency. CPU vendors have released heterogeneous multi-core chips such as big.LITTLE [1] and 4-PLUS-1 [2] where the LITTLE cluster or Battery Saver core in each architecture is utilized for low performance application to maximize the overall energy efficiency. In addition to such heterogeneous multi-core CPUs, the recent embedded GPUs support general-purpose computing like the server GPUs: many embedded GPUs including Mali, Adreno and PowerVR support OpenCL, and Tegra supports CUDA. As GPUs are specialized for throughput computing, they achieve high GFLOPS/Watt when executed in a fine-grain data-parallel fashion. As a result, most high performance chips in these days are heterogeneous architectures and it is expected that more complex SoCs will emerge with the increasing number of heterogeneous cores in CPU-GPU architectures.

While the newly introduced heterogeneous architectures allow the potential for the better energy-efficiency, it is another matter to actually achieve it: one has to map the given application carefully onto the heterogeneous platform. Typically, task-parallel executions in a pipelined fashion and data-parallel executions are considered to exploit both CPUs and GPUs. Since designers must consider not only the different types of CPU cores and GPUs, but also the various data partitioning ratios for the data-parallel execution on theses cores, the design space for the optimal mapping is huge.

Although there have been a number of researches for the design space exploration methodology which finds optimal mappings for throughput and energy, many frameworks consider only task-parallel execution, not the data-parallel execution. In this paper, we present a case study of mapping the LBP-based face detection application onto a recent CPU-GPU heterogeneous platform to demonstrate the challenge in finding the optimal mapping of the real-life example onto the contemporary platform. We first introduce how we parallelized and optimized each task of the application for data-parallel execution on a GPU. Then, we propose the models for estimating response time and energy consumption considering both task-parallel and data-parallel execution on heterogeneous cores, so that they can be used in the design space exploration process. We use a Genetic Algorithm (GA) approach to systematically find the optimal mapping which minimizes the energy consumption while satisfying the given throughput constraints.

Exynos 5422 is used as a target platform, where an octa-core big.LITTLE CPU and two asymmetric Mali GPUs are integrated in a single chip. We have verified our models by actually measuring the performance and energy consumption of the implementations directed by GA. We compared the results with those obtained with the manually mapped implementations, and confirmed that the GA directed mapping result outperformed the manual mapping result of the face detection application in maximizing the throughput. Moreover, the proposed GA-based approach can find the minimal energy consumption mapping that meets a given throughput constraint, which is very difficult to achieve manually. Compared to manually optimized im-

Manuscript received December 26, 2017.
Manuscript revised April 5, 2018.
Manuscript publicized September 18, 2018.

[†]The authors are with the School of Electrical and Computer Engineering, University of Seoul, Korea.

*This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2015R1C1A1A01055212).

a) E-mail: ymyi@uos.ac.kr (Corresponding author)

DOI: 10.1587/transinf.2018PAP0004

Copyright © 2018 The Institute of Electronics, Information and Communication Engineers

plementation, we could achieve 28.6% of reduction in energy consumption while still meeting 30 FPS constraint for Full HD (FHD) images.

The rest of the paper is organized as follows. Section 2 reviews the related work, and the background for the application and the target platform is given in Sect. 3. Section 4 introduces the optimized parallelization techniques for the LBP-based face detection algorithm, and the proposed models for throughput and energy estimation are explained in Sect. 5. The experimental results are shown and discussed in Sect. 6, followed by a conclusion in Sect. 7.

2. Related Work

2.1 Real-Time Face Detection

There have been a lot of efforts to describe a face with Local Binary Pattern (LBP) [3], [4], and also many works exist to achieve real-time performance over high definition images. Oro et al. [5] presented a real-time face detector sustaining 35 fps while decoding H.264 video on GTX470. Sharma et al. [6] proposed Haar-based face detector using CUDA. Cho et al. [7] also proposed face detector based on Haar features but using FPGA. Hefenbrock et al. [8] proposed an Integral Image based face detector using multiple GPUs, and achieved 15.2 FPS with 4 GPUs for a VGA image. Especially, Li et al. [9] presented Haar-based face detector utilizing both CPU and GPU, but they only exploited task-parallelism without data-parallel execution. As a result, they achieved 556 ms of latency for a FHD image. All of these works accelerated Haar-like face detection algorithm using server GPUs or FPGA. Recently, Gao et al. [10] presented a parallel implementation of LBP based face detection on an embedded platform called Parallela that consists of Zynq and Epiphany. By offloading the classification task onto Epiphany manycore device with some data prefetching, they achieved 4.3 FPS for FHD images, which corresponds to 3.8 times of speedup compared to OpenCV's CPU implementation.

We had previously proposed optimization techniques [11] that achieved real-time processing of LBP-based face detection in embedded GPUs for HD (720p) images, where 3.8 times of speedup was achieved and both the OpenCL and CUDA implementations on two different devices were compared. In that work, only GPU acceleration using CUDA or OpenCL was discussed. In our recent work [12], we achieved real-time processing for Full HD (1080p) images by exploiting both the multicore CPU and the GPU in Tegra K1 SoC. However, the mapping was done manually and did not consider energy consumption. In contrast, this paper proposes the execution time and energy consumption models for task- and data-parallel executions on heterogeneous PEs and finds the optimal mapping systematically.

2.2 Task Mapping

It has been one of the most addressed problems in embedded system designs to map tasks optimally onto various processing elements while meeting various design constraints. A number of task mapping frameworks that support systematic design space exploration have been proposed [13], and the need for adopting such a design flow is rapidly increasing as heterogeneous and manycore architectures have become a mainstream. In particular, stream based multimedia applications are well suited to such a design flow, and they are usually modeled as data flow graphs so that the performance can be estimated or some properties can be verified statically [14], [15]. Also, pipelined parallel execution techniques have been widely used and studied for heterogeneous processors to solve throughput-oriented problems [16]–[18].

Performance and energy consumption tradeoff: There have been many works which focused on multiple objectives such as execution time, energy consumption, reliability, accuracy and so on. One of the most common optimization goals is performance and energy consumption [19]–[22]. Especially, Ascia et al. [21] proposed a GA-based technique to find Pareto-optimal points and verified it with an MPEG-2 encoder/decoder system. However, those approaches targeted homogeneous architectures.

Heterogeneous architecture: With the advent of the general purpose accelerators, exploiting the heterogeneous architecture has become important to meet various design objectives. There are many works that tackle the optimization on the heterogeneous platforms [23]–[26]. Singh et al. [23] proposed a design-time and run-time hybrid mapping approach to optimize throughput and energy consumption in MPSoC systems. Park et al. [26] proposed a run-time system for heterogeneous computing which partitions workload and finds the optimal frequency based on performance and power estimations. Those approaches, however, take into account only task-parallelism or data-parallelism.

Task- and data-parallelism: For some workloads including multimedia applications, exploiting both task- and data-parallelism can result in a significant improvement. Yang et al. [27] proposed a task mapping heuristic that considers data-parallel execution as well as task-parallel and pipelined executions. However, it is assumed that a task graph has a static sample rate such as in SDF graph and the data-parallel nodes are mapped to the same type of processors, and that the data partitioning are done evenly, meaning the ratios are assumed to be the same. Zhou et al. [28] proposed a task mapping heuristic for CPU-GPU heterogeneous systems to minimize the completion time. However, data partitioning of a task can be considered only if the estimated time is larger than the predefined threshold. In contrast to these works, our proposed approach considers data-parallel partitioning with arbitrary ratios across heterogeneous processors. While the design objective in both works is either to minimize completion time or to maximize throughput without considering energy consumption, the proposed approach

considers energy consumption and throughput at the same time.

In this paper, we extend our recent work of real-time face detection [12] such that energy consumption is considered as well as throughput, and that the tasks can be mapped automatically meeting the design constraint. In such a way, we could obtain a highly energy-efficient and real-time LBP-based face detector for FHD images on Exynos 5422. Although the main focus in this paper is to achieve real-time yet energy efficient face detection on heterogeneous embedded systems, the proposed model in task mapping which considers both task-parallel and data-parallel execution on heterogeneous PEs can be used also for other general stream based applications.

3. Backgrounds

3.1 Face Detection Algorithm

Figure 1 illustrates the overall structure of a conventional LBP-based face detection algorithm. As a typical pre-processing step, an input image is first converted to a gray image, and Histogram equalization is applied so that a more sharpened image can be obtained. After pre-processing, a set of tasks called Resize, LBP, and Scanning are executed to find a face with LBP features. To find a face of arbitrary size, the algorithm resizes the input image while having a fixed search window, rather than resizing the search window with a fixed-sized input image. Finally, the Grouping task simply groups the same faces detected by nearby search windows into one.

It is a chain-like task graph with incoming input images, but has a cycle from the Scanning task back to the Resize task since an input image is resized recursively with a constant factor. This concept of resizing images for scale-invariant processing is referred to as Pyramid of Images. The *scale factor* determines the number of images and their sizes in the pyramid of images; the original image is first resized such that the relative size of the fixed-sized search window is not smaller than *minSize*, which is the minimum possible face size. Then, the image size is recursively divided by *scale factor* until it reaches the size in which the relative size of the search window is not larger than *maxSize*, the maximum possible face size.

Since LBP-based face detection uses LBP feature to classify whether the search window contains a face or not, LBP task needs to convert the input image into an LBP image. LBP operation compares the center pixel with each of the eight neighbor pixels, and assigns 1 to the pixel whose value is greater than that of the center pixel, and 0 otherwise. Then, 8-bit binary value so called LBP is generated by concatenating them. In such a way, each pixel in the original input image is transformed into an LBP image.

The Scanning task goes through all the search windows, comparing some LBP values in a search window with the pre-trained LBP features for a face. It employs a cascade classifier proposed by Viola and Jones [29], where the cascade classifier uses a set of weak classifiers in a series of stages.

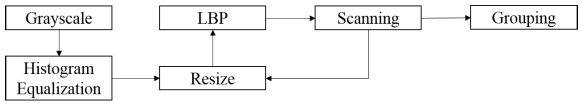

Fig. 1 The conventional task pipeline structure of LBP-based face detection. After pre-processing (Grayscale and Histogram equalization), the processing of Resize, LBP, and Scanning is repeated to extract LBP features of each resized image and compare them with the pre-trained features for a face. Finally, the detected candidates for the same face are grouped into one.

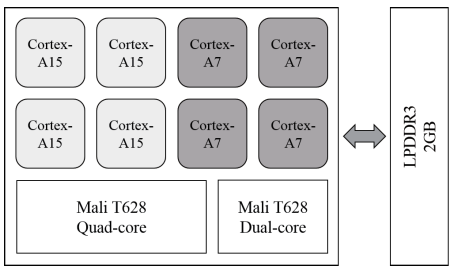

Fig. 2 The overview of the target architecture. It has four types of heterogeneous PEs and a shared memory.

3.2 Target Architecture

The target platform used in this work is Exynos 5422, which adopts big.LITTLE architecture. It has two different set of processors called big and LITTLE clusters in order to satisfy both performance and power efficiency. Such a heterogeneous solution has been driven to mitigate the problem of mobile processors that the total amount of energy is limited. Global Task Scheduling (GTS), also known as Heterogeneous Multi-Processing (HMP) techniques [1], enables each core in any cluster to be activated individually, making it possible to run all cores at the same time.

Exynos 5422 has a quad-core Cortex-A15 as a big processor and a quad-core Cortex-A7 as a LITTLE processor. Also, Mali-T628 MP6 GPU is integrated in the single chip. It is a hexa-core GPU that supports OpenCL 1.1 full profile. However, this hexa-core is not a single GPU, but consists of a quad-core and a dual-core GPUs. Therefore, users need to specify both GPU devices and launch the kernels separately to fully exploit Mali-T628 MP6.

Exynos 5422 has four different types of processing elements (PEs) which have significantly different capacities in processing their workloads: a quad-core Cortex-A15 CPU, a quad-core Cortex-A7 CPU, a quad-core Mali T628 GPU and a dual-core Mali T628 GPU.

4. Parallelization of Face Detection Algorithm

We have previously proposed efficient parallelization strategies for face detection and recognition using embedded GPGPUs [11]. In order to achieve real-time processing for

FHD images, we have recently proposed CPU-GPU cooperative approach [12]. In this section, we introduce some of the main ideas for parallelizing face detection exploiting both CPU and GPU.

4.1 Aggregation

In the original structure of the algorithm, LBP and Scanning are repeated for each resized image. If we map each task in this structure onto a core, the communications between cores would be too frequent. Moreover, the execution time for each resized image varies depending on its size, making it harder to achieve the maximum throughput of the pipelined execution. In addition, too small images cannot fully utilize the GPU.

To mitigate these problems, we aggregate the resized images into single one. Resize now generates a pyramid of images at once, not one by one recursively. Since the generated pyramid of images as a whole are treated as a single large input for the subsequent tasks such as LBP and Scanning, those tasks can be launched only once, reducing communication overheads.

More importantly, we can now exploit the data parallelism in pixel level rather than in image level with different size. This allows flexibility in data partitioning: each task can be divided into sub-tasks with an arbitrary ratio for the single large input, making it possible to have an optimal partitioning ratio for better load-balancing.

4.2 Multi-Phase Scanning

A cascade classifier consists of several weak classifiers in a series of stages, and evaluates an image patch in a search window to detect an object. If the patch passes through the last stage in the cascade classifier, the patch is assumed to have a face. If it does not pass the threshold at any stage, it terminates immediately without further processing the remaining stages and moves on to the next patch. The number of passed stages tends to increase as the search window approaches the image patch containing a face. It would be more efficient if we could skip the ones that seem to have no face.

In order to exploit this characteristic on GPU, multi-phase kernel launching has been suggested [11]. In this scheme, we make a kernel deal with only parts of its input data, and launch it multiple times sequentially with different input indices. A work-item in the kernel, which is assigned a search window, runs the cascade classifier only if the neighboring search window did not terminate at the very first stage in the cascade classifier. This can be figured out since the neighboring search window has been already processed in the previous kernel launch.

4.3 Pipelined Execution

The Fig. 3 illustrates an example of the pipelined execution

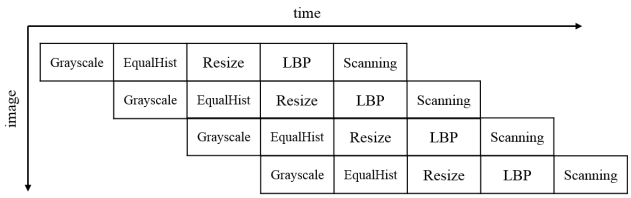

Fig. 3 An example of pipelined execution for LBP based face detection [12].

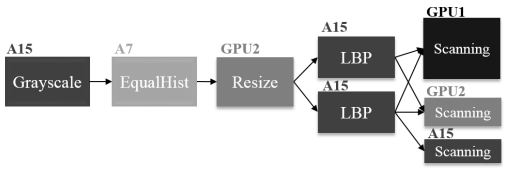

Fig. 4 A single pipeline with 5 stages for LBP-based face detection. The arrows indicate data transferring.

for the LBP-based face detection application. Since the efficiency of the pipelined execution becomes optimal when the execution time of each pipeline stage is identical, it is very important to have a load balance among the various PEs in the heterogeneous platforms. The execution time of a task in a stage differs depending on the type of a PE it is running on and the characteristic of the assigned task. For example, compute-intensive task on a LITTLE core empirically shows roughly four times slower performance than a big core. To have a good load balance, a big core should be assigned four times larger workload than a LITTLE core. However, it is difficult to have a good load balance in such a heterogeneous platform if we only consider task parallelism. Data parallelism should be also considered: if the execution time of a task on a PE is too large, it can be divided into multiple sub-tasks, each with only parts of the original data, running on multiple PEs. In this way, one can achieve better performance by alleviating the performance bottleneck in a pipelined task-parallel execution, with data-parallel execution. The aggregation makes it easy for each task to run in this fashion and allows fine-grain data partitioning. Figure 4 shows an example of the pipelined execution of the LBP-based face detection on Exynos 5422 with data-parallel execution.

5. The Estimation Model and Mapping Strategy

Our goal is to find the optimal mapping of the face detection application onto the recent CPU-GPU heterogeneous platform, which consumes the minimal energy while meeting the given throughput constraint. In order to achieve this objective, as mentioned earlier, each task should be divided into sub-tasks for data-parallel execution with an arbitrary data partitioning ratio. Thus the design space for mapping such tasks onto multiple heterogeneous PEs is huge, and it is necessary to explore the design space efficiently and systematically to find the optimal mapping. In this section, we propose a performance and energy consumption estimation

model of the heterogeneous architectures, which are used in GA as the fitness functions.

5.1 Performance Estimation of the Pipelined Execution

Estimating an accurate throughput of the pipelined execution is key to finding the optimal mappings in design space exploration. We model the throughput of the pipelined execution of an application, considering both task-parallel and data-parallel executions. We use the notations and definitions introduced in [16]. R is defined as *maximal response time*, which in turn determines the performance of the pipeline (i.e., *the maximum throughput*, $TH(\cdot)$), as following:

$$R = \max_{1 \le j \le N} \{e_j + c_j + o_j\} \tag{1}$$

$$TH(\cdot) = \frac{1}{R} \tag{2}$$

where N is the number of pipeline stages and e_j denotes the time for receiving input data, c_j the time for computation, and o_j the time for sending output data in the stage j. The throughput of the pipelined execution is related only to the stage j which has the maximal sum of e_j, c_j, and o_j. Note that the throughput of a single stage j is denoted as $TH(j)$.

Jahn et al. also introduced the *malleability* property to pipeline models. By combining multiple consecutive pipeline stages into a single stage, the *fusion* operation reduces communication overheads. And its inverse operation, *fission*, restores a fused stage to the original multiple stages.

Basically, the performance of a pipelined execution can be estimated using Eq. (1). However, the Eq. (1) assumes that the task cannot be divided into sub-tasks since they do not consider data-parallel execution. Note that the fission operation defined in [16] is to restore the fused task back to the original tasks, not dividing a task into arbitrary size of sub-tasks. Also, only one task or a fused task can be assigned to a processor. In contrast, we propose that a task can be divided into multiple sub-tasks with an arbitrary partitioning ratio, and more than one (sub-)task or a fused task can be assigned to a processor. We call it *partitioning* to divide a task τ_j into a subtask τ_j^i. Figure 5 (a) and (b) illustrate the fusion operation, and (b) and (c) illustrate partitioning operation.

In order to estimate the execution time of the proposed task- and data-parallel executions, we define the following equations. Let R_i be the response time of each stage i, and R_p the sum of the response time of the stages that are mapped to processor p. Then, we denote R_i and R_p as follows:

$$\begin{aligned} R_i &= \nu_k^i e_k + \nu_l^i o_l \\ &\quad + \sum_{j=k}^{l-1} \{\nu_j^i c_j + \nu_j^i (1 - \nu_{j+1}^i) o_j + (1 - \nu_j^i) \nu_{j+1}^i o_j\} \\ &= \nu_k^i e_k + \nu_l^i o_l \end{aligned}$$

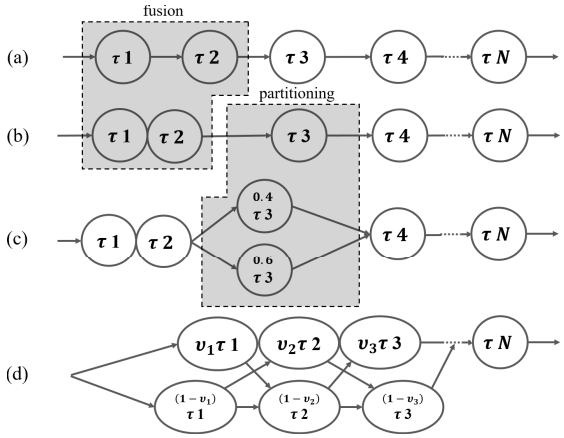

Fig. 5 An example of fusion and partitioning operation in our model. (a)-(b) The fusion operation combines consecutive tasks in the task graph into one task. (b)-(c) The partitioning operation divides a task into multiple sub-tasks for data-parallel execution. (d) A general case for fusion and partitioning is illustrated.

$$+ \sum_{j=k}^{l-1} \{\nu_j^i c_j + (\nu_j^i + \nu_{j+1}^i - 2\nu_j^i \nu_{j+1}^i) o_j\} \tag{3}$$

$$R_p = \sum_i R_i \tag{4}$$

$$R_{max} = \max_{1 \le p \le M} R_p \tag{5}$$

where ν_j^i is the partitioning ratio of the original task τ_j into a subtask τ_j^i which is assigned to the stage i. Thus, $\sum_i \nu_j^i = 1$. When a task is not partitioned at all, ν_j^i is simply 1. We assume that the transfer time and size as well as the execution time of a sub-task is proportional to the partitioning ratio. The first two terms in Eq. (3) denote the transfer time for input and output data respectively. The remaining terms in Eq. (3) account for the execution time of a (sub-)task (i.e., when l=k) or the execution time of a fused task that consists of $l - k + 1$ (sub-)tasks, as well as the time for sending the output data from (sub-)task τ_j^i in this fused task to another (sub-)task τ_{j+1}^h that does not belong to this fused task ($h \neq i$), and the time for receiving the input data from (sub-)task τ_j^h to (sub-)task τ_{j+1}^i . Figure 5 (d) illustrates this general case of a fused task with multiple partitioned sub-tasks, where each task τ_j^i is denoted together with its partitioning ratio ν_j^i for convenience. Note that $(1 - \nu_{j+1}^i)$ or $(1 - \nu_j^i)$ in Eq. (3) still holds for the cases in which there are more than one τ_{j+1}^h or τ_j^h.

Since multiple (sub-)tasks or a fused task can be mapped to a processor in our execution model, the response time of the processor is defined as the sum of the response time of the stages that are assigned to the processor (Eq. (4)). Finally, we find the largest R_p among M processors as Eq. (5), which is the maximal response time R_{max} for our case. Note that the transfer time considered in R_i is refined later in GA framework depending on whether it transfers

data to the task on the same processor or not.

Although the illustrated example for fusion and partitioning operations is a chain-like task graph, the proposed model is also applicable to any type of task graphs such as one with cycles or with multiple paths since the model considers only the amount of incoming data, computation, and outgoing data of each stage. In such cases, e_k or o_l would be the total transfer time for multiple incoming paths or outgoing paths.

5.2 Energy Consumption

In this subsection, we propose an energy consumption model. Basically, the total energy consumption of the system is the sum of dynamic energy consumption and static energy consumption.

$$E_{total} = E_{dynamic} + E_{static} \tag{6}$$

Dynamic energy consumption can be estimated by adding all the energy required to run the tasks mapped onto each PE. In addition to this general idea, we have introduced a *correlation factor* that needs to be multiplied when multiple PEs of the same type are used. We observed that the power demand does not increase linearly as the number of active processing cores increases. In fact, when multiple processing cores are utilized, the power consumption is slightly less than the linearly projected amount. This is because they share some logics among the cores in the same cluster. By conducting the experiments for each PE type, we can figure out the rate at which the power consumption increases as another core becomes active and utilized. As a result, the proposed dynamic energy consumption is defined as following Eq. (7).

$$E_{dynamic} = \sum_{p=1}^{M} C_n(Type(p))E_p \tag{7}$$

where M is the number of PEs in the platform and $Type()$ is processor type and $C_n()$ is the correlation factor when n cores are utilized in the processor. As mentioned, $C_n() < 1$ regardless of processor type.

E_p, the energy consumption of a processor p, is estimated using Eq. (11). It is calculated by adding the energy consumption of each task i that is mapped onto the processor p, which can be divided into the computation part and the communication part. The former is obtained by multiplying the power consumption of processor p that a task i is running on (P_i), with the response time of the computation part ($R_{i,comp}$). The latter is obtained similarly with the power consumption of the communication channel in processor p (P_{comm}) and the response time of the communication part in task i ($R_{i,comm}$). As Eq. (3) can be simply rewritten as Eq. (8), $R_{i,comp}$ and $R_{i,comm}$ are defined as Eq. (9) and (10) respectively.

$$R_i = R_{i,comp} + R_{i,comm} \tag{8}$$

$$R_{i,comp} = \sum_{j=k}^{l-1} v_j^i c_j + v_l^i c_l \tag{9}$$

$$R_{i,comm} = v_k^i e_k + v_l^i o_l + \sum_{j=k}^{l-1} \{(v_j^i + v_{j+1}^i - 2v_j^i v_{j+1}^i) o_j\} \tag{10}$$

$$E_p = \sum_i (P_i R_{i,comp} + P_{comm} R_{i,comm}) \tag{11}$$

On the other hand, the static energy consumption is calculated by multiplying the static power consumption required for operating devices in idle state, with the total execution time for the application. As the application is executed in a pipelined way with streaming input (images), if we neglect the initial delay to fill the pipeline, the total execution time of an image is equal to the maximal response time. Thus, the static energy consumption can be defined as Eq. (12). Consequently, Eq. (6) can be rewritten as Eq. (13).

$$E_{static} = P_{IDLE} R_{max} \tag{12}$$

$$E = \sum_{p=1}^{M} C_n(Type(p))E_p + P_{IDLE} R_{max} \tag{13}$$

Note that we did not consider the thermal effect on the energy consumption. Although the actual energy consumption will vary depending on thermal conditions, we found that the proposed model without thermal conditions satisfied our purpose.

5.3 GA-Based Task Mapping

As we mentioned earlier, the design space is huge due to the arbitrary partitioning ratio and the heterogeneity of PEs. Therefore, we adopt Genetic Algorithm (GA) to explore the design space. The task or sub-task mappings are encoded as a gene and their execution time and energy consumption are estimated using the Eq. (5) and (13) in the fitness function. A certain portion of the fittest are selected and the new population for the next generation are obtained from these survivals.

Figure 6 shows two genotypes that we configured as an example. The first one is of integer type which represents the processor ID that a (sub-)task is mapped onto. We start with five tasks in five stages like in Fig. 4: GrayScale (GS), EqualHist (EH), ReSize (RS), LBP (LB), and Scanning (SC). Then, we assume each task can be divided into two sub-tasks with an arbitrary partitioning ratio with two exceptions; EH is not partitioned at all and SC can be partitioned into three sub-tasks and the more details will be explained in Sect. 6. As a result, the total number of (sub-)tasks is 10, and Fig. 6 (a) shows one example of the processor mapping of 10 sub-tasks in the LBP-based face detection application.

The second one is of double type which represents the

Table 1 Average dynamic power consumption of each device as the number of active cores varies. From the measurements, Cortex-A15 and Mali-T628 seem to have relatively large sharing among the cores, while Cortex-A7 has little sharing among the cores.

	A7				A15				T628		
# of active cores	1	2	3	4	1	2	3	4	Quad	Dual	Both
Avg. dynamic power [W]	0.14	0.28	0.42	0.55	1.45	2.69	4.15	4.50	0.84	0.58	1.17
factor	-	1.00	1.00	0.98	-	0.93	0.95	0.78	-	-	0.82

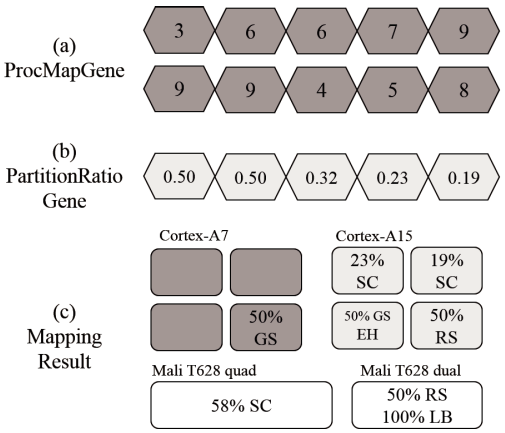

Fig. 6 An example of genes and its corresponding mapping result. GS denotes the Grayscale task, EH the Histogram equalization task, RS for the Resize task, LB for the LBP task, and SC for the Scanning task, respectively. (a) ProcMapGene encodes a processor mapping of 10 sub-tasks, where the integer value represents a PE ID: 0-3 for A7, 4-7 for A15, 8 for T628 Quad and 9 for T628 dual. (b) PartitionRatioGene encodes the partitioning ratio of each task. (c) visualizes the mapping result that (a) and (b) imply.

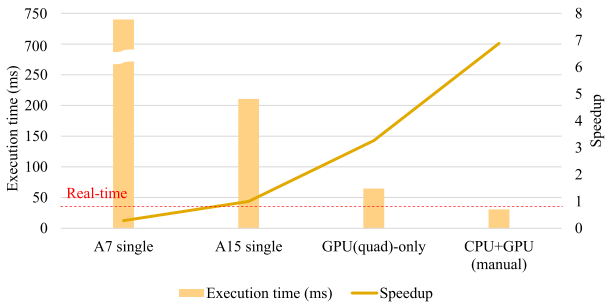

Fig. 7 Execution time and speedup of the LBP based face detection application on different PE types. *CPU+GPU*, a manual mapping by an expert, achieves 2.11 times higher throughput than *GPU-only*. However, we cannot guarantee the optimality of energy efficiency in a manual *CPU+GPU*.

ratios for data partitioning when a task is divided into sub-tasks. For example, the first value in Fig. 6 (b) is the partitioning ratio for GS, and the second for RS, and so on. If an individual has the set of genes as shown in Fig. 6 (a) and (b), then it means the mapping illustrated in Fig. 6 (c) was obtained.

6. Experimental Result

In the experiments, we used ODROID-XU3 as a target platform. It has an Exynos 5422 SoC as a main processor in which Cortex-A15 (big) cores and Cortex-A7 (LITTLE) cores run at 2.0 GHz and 1.4 GHz respectively, and equipped with 2 GB of LPDDR3 memory. Note that ODROID-XU3 does not support DVFS. Instead, it migrates a task from big cores to LITTLE cores in order to achieve a similar effect to DVFS. The design space for a DVFS-supported target platform would be even larger, making it more necessary to use the proposed approach.

Input images used in the face detection application are in FHD, which were extracted from an official movie trailer titled '50/50', and each frame generally contains one or two faces [30]. The *minSize*, *maxSize*, and *scale factor* were set to 30x30, 720x720, and 1.2 respectively, resulting in 19 images in the image pyramid of a single FHD image. Opt4J framework [31] was used for GA implementation.

Figure 7 shows the execution time of the proposed LBP-based face detection application on an each PE in Exynos 5422: a single Cortex-A7, a single Cortex-A15, and a quad-core Mali-T628. Then, CPU+GPU denotes the task- and data-parallel implementation explained in Sect. 4 with the manual mapping shown in Table 4, which will be explained later. Compared to a single-threaded Cortex-A15 version, a quad-core GPU version is 3.26 times faster and CPU+GPU is about 6.88 times faster. However, we cannot guarantee whether the achieved speedup or energy efficiency is the optimal or not. This necessitates the proposed approach to systematically find the mapping that minimizes the energy consumption while meeting the throughput constraint.

Now, let us show the mapping results obtained by the proposed GA-based design space exploration process. First, details about the energy consumption model is given. As explained earlier, when multiple PEs in the same cluster are utilized at the same time, the power consumption does not increase proportionally to the active number of PEs. We defined such a correlation factor as $C_n()$ in Eq. (7), which is obtained by using regression: To figure out C_n of each PE type as n varies, we conducted an experiment in which matrix multiplication with 800×800 matrices are executed on each PE type. For the case of CPU cores, OpenMP was used. As shown in Table 1, utilizing two cores of Cortex-A15 requires 1.86 (= 93% $\times$ 2 cores) times of the power consumption of the single core, rather than the twice of it. Likewise, utilizing all the four cores has a correlation factor of 78%. In contrast, Cortex-A7 cores seem to affect rarely each other, showing the correlation factor to be almost 1.0. For Mali-T628 MP6 GPU, we used OpenCL implementa-

Table 2 Execution time (unit: ms) and dynamic power consumption (unit: W) of each task in LBP based face detection on various PEs in Exynos 5422, when processing single FHD image.

	A7		A15		T628 Quad		T628 Dual	
Task	Time	Power	Time	Power	Time	Power	Time	Power
Grayscale	33.94	0.15	15.06	1.96	INF	INF	INF	INF
EqualHist	22.99	0.14	7.31	1.56	INF	INF	INF	INF
Resize	157.98	0.12	36.80	1.74	10.57	1.43	17.92	0.93
LBP	129.36	0.16	42.87	1.74	6.21	1.42	11.90	0.96
Scanning	395.16	0.23	108.71	2.24	32.56	0.84	61.10	0.58

Table 3 Some of the mapping results by GA and their corresponding estimates.

#	Gene	Task										Time	Energy
		GS	GS	EH	RS	RS	LB	LB	SC	SC	SC	[ms]	[mJ]
	Mapped Processor	3	3	1	2	9	9	9	9	0	8		
1	Partitioning Ratio	0.50	0.50	-	0.08	0.92	0.31	0.69	0.12	0.03	0.85	37.7	112
	Mapped Processor	3	6	4	7	9	9	9	9	5	8		
48	Partitioning Ratio	0.50	0.50	-	0.00	1.00	0.50	0.50	0.04	0.24	0.72	33.1	156
	Mapped Processor	1	6	4	7	9	9	9	9	5	8		
100	Partitioning Ratio	0.50	0.50	-	0.50	0.50	0.32	0.68	0.23	0.19	0.58	27.2	198

tion, and the correlation factor is defined between a quad-core GPU and a dual-core GPU, not among the shader cores inside each GPU since a GPU is used as a whole and an OpenCL kernel would occupy as many cores in the GPU as possible when it is launched. It shows 1.17 Watts of average dynamic power consumption, which corresponds to 82% of the sum of power consumption in both GPUs. Note that Table 1 describes only dynamic power consumption excluding static power consumption.

To estimate the execution time in Opt4J, Eq. (3) is used. Substituting each term with the corresponding value in Table 2, results in R_p. Then, R_{max} is obtained using Eq. (4) and (5). Similarly, energy consumption is estimated using Eq. (13), referring to Table 2. Table 2 shows the profiling information of our face detector in each PE type. We measured the power consumption of each task processing a single FHD image, through the built-in energy monitor. ODROID-XU3 has four sensors which can measure the current in big.LITTLE CPUs, memory, and GPUs. Assuming that the voltage is uniform, we can calculate the energy consumption of each PE. Note that we chose not to execute Grayscale and Histogram equalization on GPUs since they are not suitable for GPU: Grayscale does not show meaningful improvement due to its poor arithmetic intensity and Histogram equalization cannot be parallelized efficiently due to the inherent sequential characteristics. Those tasks are marked as unavailable by assigning infinity as shown in the Table 2.

In Table 3, we present some of the Pareto optimal mapping results when Opt4J was executed with the default parameters except for the generation value; it was raised to 15,000 to make sure that it was saturated. The exploration time is less than one minute. We restricted a task to be divided into only two sub-tasks since dividing a task into many sub-tasks would increase the synchronization overhead. There are two exceptions though: the Histogram equalization task cannot be divided due to the dependency,

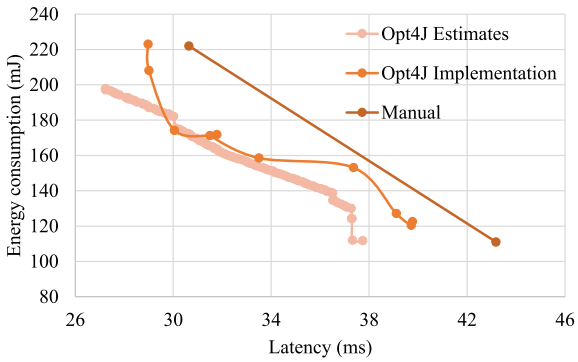

Fig. 8 Pareto-optimal graph between latency (x-axis, ms) and energy consumption (y-axis, mJ), which is denoted as *Opt4J Estimates*. It includes the results presented in Table 3. *Opt4J Implementation* denotes the actual latency and energy consumption pair measured with the implementations directed by the Opt4J mapping results. *Manual* denotes the manual mapping results by an expert who has a comprehensive understanding of both the application and the target platform.

and the Scanning task is divided into three sub-tasks as its workload is too large.

Finally, Fig. 8 plots all the estimates obtained from GA that form the Pareto-optimal graph of latency and energy consumption pairs. It also plots the actual measured values of the implementations directed by the GA mapping results, which is denoted as Opt4J implementation. They are slightly inferior to the estimated results since the proposed model used in GA estimation assumed no overheads, considering only the computation time and the memory transfer time between stages. For example, the overhead for offloading input images onto the memory is not considered in the model, nor the kernel launching overhead even if a GPU task is partitioned into several sub-tasks. Nevertheless, we can clearly see a high correlation between our estimates and the measured values. The average errors of the GA estimates are −2.19% and −3.67% for throughput and energy

Table 4 Manual mapping results for energy minimization (EN) and throughput maximization (TH) solutions, and their measured execution time and energy consumption.

#	Gene	Task										Time	Energy
		GS	GS	EH	RS	RS	LB	LB	SC	SC	SC	[ms]	[mJ]
EN	Mapped Processor	0	3	1	2	9	9	9	8	8	8	43.2	111
	Partitioning Ratio	0.50	0.50	-	0.20	0.80	0.50	0.50	0.30	0.30	0.40		
TH	Mapped Processor	4	4	0	9	9	5	6	7	9	8	30.6	222
	Partitioning Ratio	0.50	0.50	-	0.50	0.50	0.50	0.50	0.15	0.09	0.77		

consumption, respectively. The ranges of the errors are from −11.51% to 16.21%, and from −13.47% to 12.42%, respectively. The estimated minimal energy consumption by the model is 111.8 mJ with 37.7 ms of latency. The measured energy consumption from the same mapping is 122.6 mJ with 39.8 ms of latency. Likewise, the estimated maximal throughput by the model is 36.7 FPS (27.2 ms) with 197.8 mJ, while the measured value is 34.5 FPS (29.0 ms) with 223.1 mJ. For energy minimization with 30 FPS (i.e., 33.3 ms) of a throughput constraint, the estimate by the model is 159.3 mJ while the measured energy consumption is 158.6 mJ, which is only 0.43% of error.

For maximizing throughput, the experiment shows that GA-based result (28.97 ms) is faster than the manual mapping one (30.64 ms) by 5.7% while consuming almost the same energy. On the other hand, the manual mapping result for minimizing energy consumption (111.0 mJ) is better than the GA-based one (122.6 mJ). It turns out that the model disregard this mapping during the evaluation since its estimated energy consumption is 126.7 mJ, which is 12.42% of error and is higher than the lowest estimate.

As for manual mappings, we partitioned and mapped each task onto PEs as shown in Table 4, with the following strategies. For the throughput-oriented mapping, we first mapped the Scanning task to the quad-core GPU as it takes too long on the other PEs. Then, we distributed Resize and LBP to the dual-core GPU and Cortex-A15 CPUs since, regardless of their mappings, the execution time of Scanning on the quad-core GPU would be larger anyway, not affecting the overall throughput. The other tasks such as Grayscale and EqualHist can be assigned to any of the remaining PEs. Note that we also fine-tuned the mappings. After the aforementioned mapping, a fractional part of the most time-consuming task was further partitioned and mapped to the remaining PE in order to reduce the overall latency. The fine-tuning was repeated until the total execution time was saturated.

For energy-oriented manual mapping, we mapped again the Scanning task to the quad-core GPU which has the highest energy efficiency as well as the highest throughput. The energy consumption of Resize or LBP is similar on the dual-core GPU or on Cortex-A7 CPUs. Thus we mapped the most portion of those tasks to the dual-core GPU since the execution time is much shorter on the GPU than on Cortex-A7 CPUs.

Figure 9 summaries the experimental results with the various solutions. We are now able to find the most energy-efficient solution for a given throughput (e.g., real-time constraint). With a real-time constraint of 30 FPS, our GA-based solution (GA #48) shows 28.6% less energy consumption, compared to the manually designed CPU+GPU version.

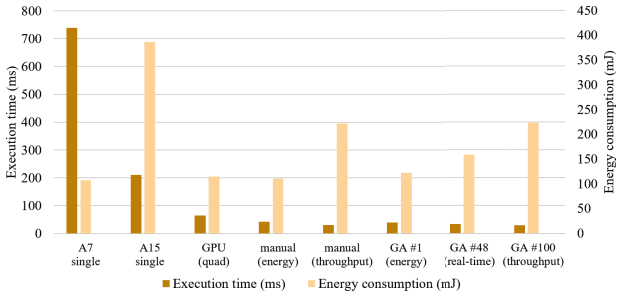

Fig. 9 Execution time and energy consumption of various implementations. The manual and GA-based solutions correspond to those in Tables 3 and 4.

7. Conclusion

Embedded systems have entered heterogeneous manycore era, where manycore accelerators such as GPUs are integrated with heterogeneous CPU cores to satisfy various design objectives, especially minimizing energy efficiency with a throughput-constraint. In this paper, we have presented real-time and energy-efficient LBP-based face detection for FHD input images on a recent CPU-GPU heterogeneous embedded platform. Not only have we shown the optimized parallelization techniques to achieve such a high throughput, but also we have proposed models for energy consumption and throughput estimation that can be used in design space exploration frameworks. The proposed models consider data-parallel execution with arbitrary partitioning ratios, in addition to the task-parallel execution in a pipelined fashion.

We have verified the proposed models with a GA framework in mapping the LBP-based face detection application onto Exynos 5422, where 10 PEs with four different types exist. Compared to the manual mapping results obtained by the experts, the output of GA-based mapping result excelled in throughput by 5.5%. Moreover, we demonstrated we can easily find an energy-efficient solution with a throughput constraint, which in contrast would be very difficult or infeasible to find manually. Our GA-based mapping consumed 28.6% less energy still meeting the real-time constraint.

References

[1] B. Jeff, "big. little technology moves towards fully heterogeneous global task scheduling," White Paper., 2013.
[2] NVIDIA, "K1: A new era in mobile computing," NVIDIA White Paper, 2014.
[3] T. Ahonen, A. Hadid, and M. Pietikainen, "Face description with local binary patterns: Application to face recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.28, no.12, pp.2037–2041, 2006.
[4] C. Shan, S. Gong, and P.W. McOwan, "Robust facial expression recognition using local binary patterns," IEEE International Conference on Image Processing, 2005, ICIP 2005, pp.II–370, IEEE, 2005.
[5] D. Oro, C. Fernández, J.R. Saeta, X. Martorell, and J. Hernando, "Real-time gpu-based face detection in hd video sequences," 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops), pp.530–537, IEEE, 2011.
[6] B. Sharma, R. Thota, N. Vydyanathan, and A. Kale, "Towards a robust, real-time face processing system using cuda-enabled gpus," 2009 International Conference on High Performance Computing (HiPC), pp.368–377, IEEE, 2009.
[7] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using haar classifiers," Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, pp.103–112, ACM, 2009.
[8] D. Hefenbrock, J. Oberg, N.T.N. Thanh, R. Kastner, and S.B. Baden, "Accelerating viola-jones face detection to fpga-level using gpus," 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.11–18, IEEE, 2010.
[9] E. Li, B. Wang, L. Yang, Y.-T. Peng, Y. Du, Y. Zhang, and Y.-J. Chiu, "Gpu and cpu cooperative accelaration for face detection on modern processors," 2012 IEEE International Conference on Multimedia and expo (ICME), pp.769–775, IEEE, 2012.
[10] F. Gao, Z. Huang, S. Wang, and X. Ji, "Optimized parallel implementation of face detection based on embedded heterogeneous many-core architecture," International Journal of Pattern Recognition and Artificial Intelligence, vol.31, no.7, 1756011, 2017.
[11] S. Yi, I. Yoon, C. Oh, and Y. Yi, "Real-time integrated face detection and recognition on embedded GPGPUs," 2014 IEEE 12th Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), pp.98–107, IEEE, 2014.
[12] C. Oh, S. Yi, and Y. Yi, "Real-time face detection in Full HD images exploiting both embedded CPU and GPU," 2015 IEEE International Conference on Multimedia and Expo (ICME), pp.1–6, IEEE, 2015.
[13] A.K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," Proceedings of the 50th Annual Design Automation Conference, p.1, ACM, 2013.
[14] C. Lee, W.W. Ro, and J.-L. Gaudiot, "Cooperative heterogeneous computing for parallel processing on cpu/gpu hybrids," 2012 16th Workshop on Interaction between Compilers and Computer Architectures (INTERACT), pp.33–40, IEEE, 2012.
[15] L. Schor, A. Tretter, T. Scherer, and L. Thiele, "Exploiting the parallelism of heterogeneous systems using dataflow graphs on top of opencl," The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia), pp.41–50, IEEE, 2013.
[16] J. Jahn, S. Pagani, S. Kobbe, J.-J. Chen, and J. Henkel, "Optimizations for configuring and mapping software pipelines in many core systems," Proceedings of the 50th Annual Design Automation Conference, p.130, ACM, 2013.
[17] A. Tumeo, M. Branca, L. Camerini, M. Ceriani, M. Monchiero, G. Palermo, F. Ferrandi, and D. Sciuto, "Prototyping pipelined applications on a heterogeneous fpga multiprocessor virtual platform," Proceedings of the 2009 Asia and South Pacific Design Automation Conference, pp.317–322, IEEE Press, 2009.
[18] S.L. Shee and S. Parameswaran, "Design methodology for pipelined heterogeneous multiprocessor system," Proceedings of the 44th annual Design Automation Conference, pp.811–816, ACM, 2007.
[19] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular noc architectures," IEEE Transactions on computer-aided design of integrated circuits and systems, vol.24, no.4, pp.551–562, 2005.
[20] C.A.M. Marcon, E.I. Moreno, N.L.V. Calazans, and F.G. Moraes, "Comparison of network-on-chip mapping algorithms targeting low energy consumption," IET Computers & Digital Techniques, vol.2, no.6, pp.471–482, 2008.
[21] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based noc architectures," Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pp.182–187, ACM, 2004.
[22] E.W. Brião, D. Barcelos, and F.R. Wagner, "Dynamic task allocation strategies in mpsoc for soft real-time applications," Proceedings of the Conference on Design, Automation and Test in Europe, pp.1386–1389, ACM, 2008.
[23] A.K. Singh, A. Kumar, and T. Srikanthan, "Accelerating throughput-aware runtime mapping for heterogeneous mpsocs," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol.18, no.1, p.9, 2013.
[24] V. Petrucci, O. Loques, D. Mossé, R. Melhem, N.A. Gazala, and S. Gobriel, "Energy-efficient thread assignment optimization for heterogeneous multicore systems," ACM Transactions on Embedded Computing Systems (TECS), vol.14, no.1, p.15, 2015.
[25] A. Colin, A. Kandhalu, and R.R. Rajkumar, "Energy-efficient allocation of real-time applications onto single-isa heterogeneous multicore processors," Journal of Signal Processing Systems, vol.84, no.1, pp.91–110, 2016.
[26] J. Park and W. Baek, "Rchc: A holistic runtime system for concurrent heterogeneous computing," 2016 45th International Conference on Parallel Processing (ICPP), pp.211–216, IEEE, 2016.
[27] H. Yang and S. Ha, "Pipelined data parallel task mapping/scheduling technique for mpsoc," Proceedings of the conference on Design, automation and test in Europe, pp.69–74, ACM, 2009.
[28] H. Zhou and C. Liu, "Task mapping in heterogeneous embedded systems for fast completion time," Proceedings of the 14th International Conference on Embedded Software, p.22, ACM, 2014.
[29] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2001, pp.I–511, IEEE, 2001.
[30] "50/50 The Movie - Official Trailer [HD]," available: http://www.50-50themovie.com [accessed: 18 October 2018], 2011.
[31] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4j: a modular framework for meta-heuristic optimization," Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp.1723–1730, ACM, 2011.

Chanyoung Oh received the B.S. degree in Electrical and Computer Engineering from the University of Seoul in 2015. He is currently a Ph.D. student in University of Seoul. His research interest includes parallel software design, heterogeneous computing, embedded GPU platforms, computer vision and medical imaging.

Saehanseul Yi received the B.S. and M.S. degrees in Electrical and Computer Engineering from the University of Seoul in 2013 and 2015, respectively. He is currently a Ph.D. student in the University of California, Irvine. His research interest includes parallel software design, heterogeneous computing, embedded GPU platforms, computer vision and high-performance distributed framework using manycore accelerators.

Youngmin Yi received the B.S. degree in Computer Engineering and Ph.D. degree in Electrical Engineering and Computer Science from Seoul National University in 2000 and 2007 respectively. He was a Postdoctoral Researcher at the University of California, Berkeley from 2007 and 2009, and a senior researcher at Samsung Advanced Institute of Technology from 2009 to 2010 before he joined the School of Electrical and Computer Engineering in the University of Seoul, where he is currently an Associate Professor. His research interest includes algorithm/architecture codesign for heterogeneous manycore platforms, GPU computing, and computer vision applications.