Received December 11, 2019, accepted February 16, 2020, date of publication March 2, 2020, date of current version March 12, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2977496

Scheduling of Deep Learning Applications Onto Heterogeneous Processors in an Embedded Device

DUSEOK KANG[1], (Member, IEEE), JINWOO OH[1], JONGWOO CHOI[1], YOUNGMIN YI[2], AND SOONHOI HA[1], (Fellow, IEEE)
[1]Department of Computer Engineering, Seoul National University, Seoul 08826, South Korea
[2]Department of Electrical and Computer Engineering, University of Seoul, Seoul 02504, South Korea

Corresponding author: Soonhoi Ha (sha@snu.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant 2018R1D1A1B07050463, and in part by the Ministry of Science and ICT under Grant 2019R1A2B5B02069406.

ABSTRACT As the need for on-device machine learning is increasing recently, embedded devices tend to be equipped with heterogeneous processors that include a multi-core CPU, a GPU, and/or a DNN accelerator called a Neural Processing Unit (NPU). In the scheduling of multiple deep learning (DL) applications in such embedded devices, there are several technical challenges. First, a task can be mapped onto a single core or any number of available cores. So we need to consider various possible configurations of CPU cores. Second, embedded devices usually apply Dynamic Voltage and Frequency Scaling (DVFS) to reduce energy consumption at run-time. We need to consider the effect of DVFS in the profiling of task execution times. Third, to avoid overheat condition, it is recommended to limit the core utilization. Lastly, some cores will be shut-down at run-time if core utilization is not high enough, in case the hot-plugging option is turned on. In this paper, we propose a scheduling technique based on Genetic Algorithm to run DL applications on heterogeneous processors, considering all those issues. First, we aim to optimize the throughput of a single deep learning application. Next, we aim to find the Pareto optimal scheduling of multiple DL applications in terms of the response time of each DL application and overall energy consumption under the given throughput constraints of DL applications. The proposed technique is verified with real DL networks running on two embedded devices, Galaxy S9 and HiKey970.

INDEX TERMS Deep learning scheduling, genetic algorithm, heterogeneous processor, mobile device.

I. INTRODUCTION

As deep learning (DL) is making significant progress in almost all areas of machine learning, more applications based on DL will be seen in our daily life. To avoid any concern of privacy and network condition, running the DL applications directly in a mobile embedded device instead of resorting to the cloud system will be more popular. In order to cope with the high computing demand of DL applications under a limited power budget, an embedded device is becoming more heterogeneous, equipped with multi-core CPU, GPU, and other accelerators such as Neural Processing Unit (NPU) and/or Digital Signal Processors (DSP) array. In addition, there is a growing need to run multiple DL applications concurrently in the emerging embedded systems such as self-driving cars and smartphones. In those systems, it is necessary to schedule multiple DL applications on the shared heterogeneous processing elements, which is a challenging problem tackled in this paper.

A DL application is usually developed with DL frameworks and libraries such as TensorFlow [1], Caffe2 [2], PyTorch [2], ARM Compute Library (ACL) [3], and so on. The current practice of running a DL application in an embedded system is to run the DL framework itself, and the DL framework usually uses a single processing element (PE) to run the application. For instance, ACL that can be used to run a DL application in a Galaxy S9 smartphone assumes that the application is run on the multi-core CPU or a Mali GPU, not both. It is expected that we can improve the performance of the smartphone for DL applications if we can

The associate editor coordinating the review of this manuscript and approving it for publication was Zhipeng Cai.

use heterogeneous PEs cooperatively. In particular, we aim to increase the throughput performance and minimize energy consumption by using both CPU and GPU in embedded devices for a single DL application. In addition, we find the Pareto optimal mapping of multiple concurrent DL applications onto heterogeneous processors (PEs) in terms of a response time of each DL application and overall energy consumption under the given throughput constraints of DL applications.

A DL inference algorithm is usually specified with an acyclic task graph where a node represents a computation task, called layer [1], and an arc represents the dependency between two end nodes. Since a task or a layer contains massively parallel computation inside, a task can be mapped to multiple processing cores to exploit *data-level* parallelism of a task or *intra-layer* parallelism. On the other hand, multiple tasks that have no dependency on each other can run in parallel to exploit *task-level* parallelism of the algorithm or *inter-layer* parallelism.

If the computation workload of each task does not vary at run-time, we can perform static scheduling of a DL application onto the available PEs, considering both task-level and data-level parallelism. In this paper, we propose a scheduling framework for DL applications that covers from profiling on real embedded devices to verifying the scheduler results on the devices. We use a Genetic Algorithm (GA)-based scheduling technique for the effective scheduling of DL applications onto heterogeneous PEs, exploring both data-parallelism and task-parallelism to find Pareto-optimal schedules in terms of real-time performance and energy consumption. In the parallel scheduling of multiple DL applications with different throughput constraints, a critical issue is to check the schedulability of mapped tasks on each processor. We have to consider not only the task dependency between tasks in the same application but also the interference among tasks that belong to different applications when *processor sharing* is allowed.

There are several issues to be considered in the scheduling of a DL application on an embedded device. First, the device usually runs a Dynamic Voltage Frequency Scaling (DVFS) governor to dynamically change the frequency of a PE, based on the utilization. It means that the task execution time may vary at run-time dynamically due to the DVFS policy adopted in the device. Second, a multi-core CPU can be configured in various ways. For instance, a 4-core CPU can be configured as a single PE with 4 cores at one extreme, or it can be configured as 4 PEs with a single core at the other extreme. It should be determined which configuration is the best. Third, an embedded device typically has a restriction on the CPU utilization below a certain threshold in order not to overheat the system. If the chip becomes hotter than a given temperature threshold, it reduces the frequency to the minimum and cools down the chip. Considering such the thermal management policy, the schedule that can avoid this unexpected behavior should be selected.

[1]We use the terms, task and layer, interchangeably throughout this paper.

While parallel scheduling of task graphs has been extensively researched so far, there is no previous work that considers the aforementioned practical issues to the best of our knowledge. The proposed methodology is verified by running 2 widely used Convolutional Neural Networks (CNNs) on a Galaxy S9 smartphone [4] and a HiKey970 board [5]. For the experimentation, we implemented a DL inference engine that can utilize heterogeneous PEs using a low-level library of the ACL, which is another contribution of this work.

We summarize our contributions as follows:

- We propose a scheduling framework that maps (sub-) layers of a CNN on heterogeneous PEs such as CPU, GPU, and NPU, taking into account several practical issues in the embedded devices.
- This is the first work to schedule multiple DL applications on heterogeneous PEs with processor sharing, considering both inter-layer parallelism and intra-layer parallelism.
- The proposed methodology is verified by running CNNs on two different embedded devices: a Galaxy S9 smartphone and a Hikey970 board.

The remainder of this paper is organized as follows. The related work on on-device deep learning frameworks and scheduling techniques on heterogeneous PEs are discussed in Section II. The hardware platform and system model used in this work are described in section III. In section IV, we describe the overall flow and how to profile execution time and communication time of each layer on different PEs. After the scheduling technique for a single application is explained in section V, section VI describes how to extend our scheduling framework to multi-application scheduling. Section VII verifies our scheduling method by comparing the scheduling results with the results obtained from the actual implementation. Section VIII concludes this paper.

II. RELATED WORK

A. ON-DEVICE DEEP LEARNING FRAMEWORK

There are many recent studies for executing DL applications on embedded devices. While they mostly consider a single DL application, we also consider the scheduling of multiple applications that share the processors. CNNDroid [6] is a library that accelerates a DNN by dividing layers into GPU and CPU. It simply maps the convolution and fully-connected layers to GPU and the other layers to CPU. RSTensorflow [7] is a framework for users to use heterogeneous processors such as CPU and GPU easily. It uses RenderScript [8] to manually parallelize the computation workloads in a layer, such as matrix multiplication and convolution, across CPU cores and GPUs.

Mirhoseini *et al.* [9] proposed a hierarchical DNN model for the efficient placement of a neural network graph onto hardware devices. The *Grouper* groups graph operations, and the *Placer* maps them to the devices. Even if they also try to utilize not only GPUs but also CPUs in the system, unlike our approach, they do not consider the various configurations with multi-cores in a CPU, nor the pipelining scenario: only task-parallelism with parallel edges is considered.

μLayer [10] is the latest work to accelerate a DNN on heterogeneous processors on Samsung Galaxy Note 5 and Galaxy A5. It aims to exploit only data-level parallelism of each layer using the heterogeneous PEs, unlike ours which also exploits task-level parallelism. Since all PEs need to be synchronized and data communication between PEs is necessary at the end of each layer, the communication and synchronization overhead is significant. Also, it did not consider a DVFS policy and CPU utilization constraints of the smartphone. Nonetheless, experimentation with the real smartphone is laudable since practical issues need to be resolved like this work.

DeepX [11] considers data-level parallelism in the mapping and scheduling of DL applications on heterogeneous multi-processor platforms. DeepX performs layer-wise partitioning first and divides the workload of a layer into a group of unit blocks that are defined as the computation requirements to update a single output node in a layer. The authors propose an Integer Linear Programming (ILP) formulation to find a trade-off between energy consumption and latency by allocating the unit blocks to PEs layer by layer. While they show the experimental results of layer-wise partitioning onto the heterogeneous PEs, no performance comparison is reported between the simple layer-wise partitioning and the ILP-based block-level partitioning. Their work differs from ours in that they do not consider task-level parallelism and their objective function does not consider throughput constraint.

B. SCHEDULING MULTIPLE APPLICATIONS ON HETEROGENEOUS PROCESSORS

We schedule multiple applications on heterogeneous processors with processor sharing, exploiting data parallelism inside a task as well as task-level parallelism. This section presents some related work on scheduling techniques.

1) SCHEDULING TECHNIQUES ON HETEROGENEOUS PROCESSORS

Since scheduling acyclic task graphs on a heterogeneous system is a well-known NP-hard problem, several heuristics have been proposed to solve this problem. Topcuoglu *et al.* [12] proposed a Heterogeneous Earliest Finish Time (HEFT) algorithm and a Critical Path On a Processor (CPOP) algorithm, which both reduce task finish time in a greedy manner. HEFT uses the concept of *rank*, which calculates the execution time of a critical path from one task to the last task, to prioritize each task, and schedule tasks by mapping them to the processor in a way that minimizes the finish time. CPOP, on the other hand, reduces the overall latency of the application by first mapping all tasks in the critical path to one fastest processor. When HEFT and CPOP schedule tasks on heterogeneous PEs, the rank of a task is determined by the average execution time on all processors. On the other hand, the Predict Earliest Finish Time (PEFT) algorithm proposed by Arabnejad and Barbosa [13] uses the Optimistic Cost Table (OCT), which has different task execution times for each processor. These heuristics are concerned about the scheduling of a single task graph. Roy *et al.* [14] proposed an Integer Linear Programming (ILP) algorithm to find the optimal schedule result in terms of a makespan, or response time, of an application. This work differs from other works in that it finds the optimal schedule. While this work considers only one application, our method can schedule multiple applications. Also, our method maximizes throughput when scheduling a single application, while other studies aim to reduce the makespan.

Zhao and Sakellariou [15] have addressed the scheduling problem of multiple applications. Their solution is to simply merge multiple applications into one large task graph. Hence it does not allow applications to have different periods and random starting offsets. Xie *et al.* [16] proposed two static heuristic algorithms, F_MHEFT and D_MHEFT, which are global scheduling algorithms that schedule multiple applications with mixed criticality levels on heterogeneous PEs. They present two scheduling algorithms; F_MHEFT aims to improve the system performance based on the fairness policy while D_MHEFT aims to meet the deadline of high-criticality applications. D_MHEFT is similar to our method in that it schedules the target applications with deadlines of tasks in mind. However, we schedule multiple applications with different periods and starting offsets, and also consider the worst interference by other application tasks. To apply D_MHEFT to multiple applications with different periods and starting offsets, it would have to simulate all possible cases.

2) SCHEDULABILITY WHEN SCHEDULING MULTIPLE APPLICATIONS

When we schedule multiple applications onto a multiprocessor system, a popular solution is to partition the processors to the applications spatially and/or temporarily. In other words, each PE is assigned exclusively to a single application. Then, each application can run on the assigned set of processors exclusively without worrying about schedulability. If we allow a processor to be shared among multiple task graphs, however, any parallel scheduling should check if the mapped tasks are schedulable on each processor.

There are two approaches to tackle this scheduling problem. One is to transform the task graph into a set of independent tasks that have different starting offsets and relative deadlines [17]. In this approach, the starting offsets and deadlines should be conservatively assigned, considering the dependency between tasks. After transformation, the conventional schedulability analysis method for independent tasks is applied.

The second approach is to compute the time range in which each task should be scheduled to guarantee the satisfaction of the deadline and to check if the possible interference from the other applications is smaller than the range [18]. Since the former approach, transformation approach, checks schedulability pessimistically [18], we choose the latter approach, schedule-based approach.

3) SCHEDULING TECHNIQUES CONSIDERING DATA PARALLELISM INSIDE A TASK

Even though there exist numerous scheduling techniques that have been proposed for homogeneous or heterogeneous multi-core systems, they mostly assumed that a task is run on a processor, and seldom considered data parallelism inside a task. Some recent studies considered data parallelism of tasks as well as task parallelism. Liu *et al.* [19] proposed heuristic algorithms to minimize the scheduling length of a task graph with data-parallel tasks. Yang *et al.* [20] proposed an evolutionary algorithm to schedule a task graph, considering task parallelism, data parallelism, and pipelining, with an objective to maximize the throughput performance. The same authors proposed an ILP based technique for minimizing the total processor cost while satisfying the time constraints.

The previous works are usually based on a static model of a hardware platform. They do not consider the characteristics of the actual hardware platform on which the scheduling algorithm will run. Thus it is simply assumed that the execution time of a task on each PE is given and the processor configuration is also fixed. On the other hand, we integrate the characteristics of the hardware platform into problem formulation by profiling each task considering various processor configurations. This makes our approach distinguished from the existent ones. Moreover, the scheduling results are verified with actual implementations, which has not been carried out in most of the previous works.

III. HARDWARE PLATFORM AND SYSTEM MODEL

Since the proposed framework is applied to embedded devices, we first explain the characteristics of hardware platforms used in this work: Galaxy S9 smartphone and HiKey970 board. Galaxy S9 is a heterogeneous system that consists of a Mali-G72 MP18 GPU and big.LITTLE CPUs with a quad-core M3 CPU running at 2.7GHz and a quad-core Cortex-A55 CPU at 1.79GHz. HiKey970 is also a heterogeneous system that consists of a Mali-G72 MP12 GPU and big.LITTLE CPUs with a quad-core A73 running at 2.36GHz and a quad-core Cortex-A53 at 1.8GHz. Besides, it has an NPU that can accelerate DL applications. We believe that our approach can be applied to other hardware platforms since it uses a black box model for each processing element (PE) with the profiled execution time and communication time at the task level without assuming a specific hardware architecture.

Since reducing the energy consumption is critical in mobile embedded devices, Galaxy S9 adopts an aggressive DVFS policy, called *schedutil*, that lowers the frequency and the voltage level of CPU cores if the average utilization of cores is below a pre-specified threshold. No DVFS policy is used in HiKey970. Galaxy S9 even shutdowns some cores dynamically using a CPU hot-plug feature supported by Linux. Since the overheating induces unexpected slow-down of an application, the maximum CPU core utilization is usually set

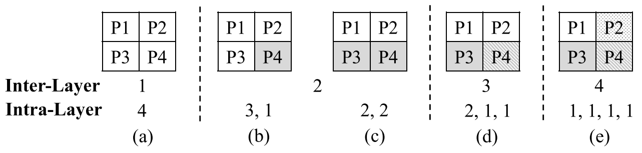


FIGURE 1. Five different CPU core configurations considering inter-layer parallelism and intra-layer parallelism.

to avoid such an unpleasant situation. We profile DL networks considering DVFS and hot-plug, and propose a mapping technique that can limit CPU utilization.

For software implementation of DL applications on ARM processors, ARM provides an open source software development kit, called ARM NN [21]. Unfortunately, it does not support Galaxy S9. Moreover, it does not support the parallel execution of a DL network on a heterogeneous system. Thus partitioned DL applications need to be written manually using ACL [3] that contains OpenCL implementation of DL operations for GPUs and NEON [2] implementation for CPU. APIs for utilizing the NPU are more limited. Even though HiKey970 has an NPU inside, the software development environment is not open to the public so that the NPU could not be used to verify our schedule results in section VII.

Most deep learning libraries and previous works consider a multi-core CPU as a single processor and exploit the data-parallelism of the mapped layer using multi-threading. For instance, ACL executes a convolution layer either on a multi-core CPU or a GPU exploiting the data-parallelism in the layer to reduce overall inference time. In contrast, we model the quad-core CPU as a set of logical processing elements since multiple layers may run concurrently on different cores in the CPU. Figure 1 illustrates five different configurations that we can choose with four CPU cores. Each figure in Figure 1 shows that four CPU cores can be utilized differently with different combinations of *inter-layer* and *intra-layer* parallelism degree that is indicated by numbers below the figures. The degree of inter-layer parallelism indicates how many layers are executed concurrently, and the degree of intra-layer parallelism is given as a tuple that represents how many cores are assigned to the layers running concurrently. For example, Figure 1 (b) illustrates the case in which the degree of inter-layer parallelism is 2 and that of intra-layer parallelism is (3, 1), meaning that two layers are mapped on the CPU of which one layer is executed by 3 cores while the other layer by 1 core.

IV. PROPOSED SCHEDULING FRAMEWORK AND PROFILING

Figure 2 displays the overview of the proposed scheduling framework of deep learning (DL) applications on an embedded device. While the proposed methodology is applied to two specific embedded devices in this work, it is applicable to other embedded devices. Before a scheduling decision is

[2]NEON is a SIMD architecture extension for ARM Cortex-A processors.

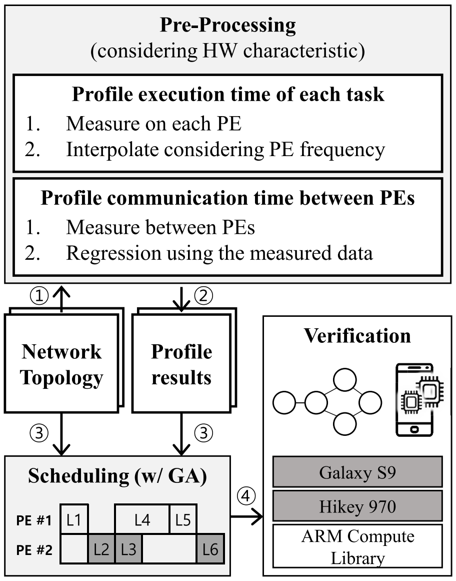


FIGURE 2. The proposed deep learning (DL) applications scheduling flow (The numbers in the small circles indicate the order of the scheduling flow.).

made, it should be known how much time is taken to execute a task on each processing element (PE) and the communication overhead between two dependent tasks, which would vary if they are mapped to different PEs. Such profiles are obtained by running the DL application on each PE in the preprocessing step.

With the profiled task information, we perform static scheduling of DL applications on the heterogeneous processors in the given hardware platform to optimize a given objective function under the constraint on the CPU utilization. For a single DL application, the objective is to increase the throughput and to minimize energy consumption. The output of the framework is a set of Pareto optimal schedules, which includes mapping information of layers onto PEs.

When scheduling multiple DL applications, we assume that the period of each application is given as the throughput constraint. While we can use other objectives, the scheduling objective assumed in this paper is to minimize the response time of each application and the total energy consumption of the system. The deadline is set to be the same as the period. The framework finds a set of Pareto optimal schedules that meet the deadline of each application. Before we explain the proposed scheduling techniques in subsequent sections, we elaborate on the profiling and performance estimation technique in this section.

A. TASK PROFILING AND ESTIMATION

Profiling is performed with an in-house deep learning framework [22] that generates an OpenCL code for the Mali GPU and multi-threaded NEON code for the multi-core ARM CPU in the device using ACL. By adding a time-stamping code at each task (or layer) boundary, the elapsed time between any two points of interest can be measured. For the GPU, the kernel time can be measured by using OpenCL profiling APIs.

TABLE 1. An example of profiling (network: SqueezeNet).

Layer	4 core	3 core	2 core	1 core	1 core*	GPU
c1	3,951	6,958	8,178	14,569	13,358	1,628
c9	734	1,205	1,362	2,253	1,755	536
c14	324	400	477	589	417	590

1 core* means that no ACL thread is created. (Unit: μs)

The GPU execution time is easy to obtain since OpenCL utilizes all GPU cores, and the task execution time does not vary in a DL application. On the other hand, CPU profiling of a task is tricky, with many issues that should be taken into account. First, the five different CPU configurations should be considered as discussed in Figure 1: a task should be profiled with a varying number of cores from 1 to 4, as shown in Table 1. For profiling with the different number of CPU cores, a Linux command *taskset* is used to assign specific CPU cores to a process.

Second, the CPU execution time should be adjusted since the embedded device may run a Dynamic Voltage Frequency Scaling (DVFS) governor that changes the CPU frequency depending on the utilization. Moreover, the Linux kernel may turn on and off a CPU core at run-time, supporting the CPU hot-plug feature. For example, if a task is executed on a single core and no other applications or processes are running on the other cores, Galaxy S9 would turn off the other cores to reduce the energy consumption and increase the CPU frequency up to the maximum of 2.7 GHz. If a task is mapped to all four cores, on the other hand, the DVFS governor would lower the CPU frequency to 1.79GHz to reduce the heat and power consumption. With two cores assigned, a task is run at the frequency of 2.31GHz. Since other processes may be running while the target DL applications are running, it is safe to assume that all CPU cores will be busy in reality. Thus the profiled CPU execution time is calibrated based on the observed CPU frequency, assuming that all cores are busy even though a task is not mapped onto four cores. For instance, the execution time profiled on a PE with two CPU cores has to be increased by $\frac{2.31}{1.79}$. This simple interpolation is based on the assumption that computation time will increase inversely proportional to the CPU frequency, which is a source of error between our profiling results and the actual measurements.

More detailed profiling of convolution layers is performed to examine the cause of varying speedup ratio. Table 2 shows the measured execution time of four kernels involved in each convolution layer for c1, c9, and c14. Note that a convolution operation is computed by GEMM (general matrix multiplication) after converting the input image to a suitable matrix. If a target device has four cores, the ACL creates three threads in addition to the main thread. The main thread first distributes the workload to the threads and computes the remaining workload. Then, it synchronizes with other threads waiting for them to be joined. In the table, *Sync Overhead* is this

TABLE 2. Intra-Layer parallelism analysis.

Layers	Kernel	Computation Time	Sync Overhead
c1	Im2Col	182,341	18,449
	GEMM	623,818	215,803
	Col2Im	2,567,736	310,425
	ReLU	198,487	51,220
c9	Im2Col	25,097	32,840
	GEMM	95,607	15,418
	Col2Im	307,815	52,362
	ReLU	59,717	38,172
c14	Im2Col	30,455	30,312
	GEMM	111,703	6,195
	Col2Im	23,583	38,762
	ReLU	7,328	29,714

Unit: ns

waiting time including the conditional wait API overhead. In c1 and c9, the synchronization overhead is small compared to the computation time. In c14, however, the overhead is comparable to or even larger than the computation time. It makes the CPU cores idle and hinders the performance improvement of c14 when intra-layer parallelism is four, as shown in Table 1.

It is noteworthy that, in some layers, *Col2Im* time is greater than the GEMM time. This is because the memory access pattern of *Col2Im* has very poor locality: the kernel mainly consists of memory operations, but the stride of the write operations is the 2-D output tensor size (width × height), and the size per access is only 4 Bytes.

B. COMMUNICATION OVERHEAD PROFILING AND ESTIMATION

Communication overhead between two tasks should be considered in making a scheduling decision. In ACL, the input and output tensor buffers of each layer (or task) should be defined statically. If two adjacent tasks are mapped onto the same PE, the output tensor buffer of a task can be shared with the input tensor buffer of its successor task, resulting in no communication overhead between them. If they are mapped onto different PEs, however, a buffer cannot be shared, but separate buffers are required for each tensor in order to run tasks in a pipelined fashion. Likewise, the two adjacent tasks that are mapped to different *logical* PEs among four cores in a CPU cannot share a tensor buffer. Data should be copied between separate buffers using *memcpy*. For the communication between CPU and GPU, *map* and *unmap* OpenCL APIs are used. The *map* API is used to access the data in the GPU memory address space from the CPU, while the *unmap* releases the mapping so that the mapped data can then be computed by the GPU.

Thus communication overhead is estimated differently depending on the types of communicating PEs. Communication time between different CPU PEs is equal to *memcpy* time, and the time from GPU to CPU is set to (*map*+*memcpy*) time since the OpenCL maps GPU memory to the host and then copies the data to CPU. The time from CPU to GPU is set to (*map*+*memcpy*+*unmap*) time since it has to map GPU

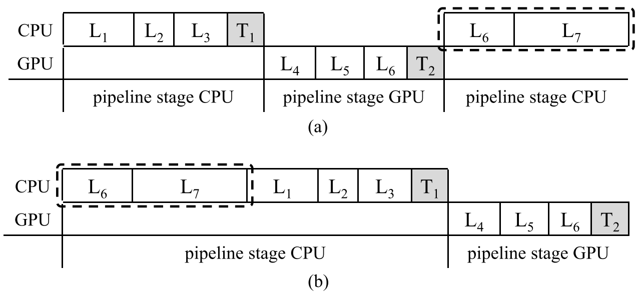


FIGURE 3. Task-clustering mapping of an application.

memory to the host (CPU) in order to send data to GPU and then unmap it for the next layer's GPU processing.

To measure the overhead of those APIs, we made a micro-bench and ran it 1000 times, from which the averaged overhead was obtained. By changing the data size in the APIs, the overhead of *memcpy* on Galaxy S9 is approximated as $1.06 \times 10^{-2} \times (DataSize)^2 + 2.75 \times 10^{-1} \times (DataSize) + 1.70$, *unmap* as $9.97 \times (DataSize) + 526.39$, and *map* as $21.75 \times (DataSize) + 569.25$.

C. NPU PROFILING AND ESTIMATION

Accurate profiling on an NPU could not be made since the API for executing a layer on an NPU is not available, which makes direct profiling impossible. Instead, performance estimation on the NPU in HiKey970 is made indirectly by assuming that the NPU is about 6.5 times faster than that of the quad-core CPU based on the performance comparison reports in [23], [24]. Also, the communication time between CPU and NPU is assumed to be the same as the time between CPU and GPU. For energy consumption, a similar estimation is made with the comparison reported in [23], [24] since it is not possible to measure the power consumption of a task on each PE. As a result, the power consumption of CPU, GPU, and NPU is assumed to be 3.5W, 4W, and 2W, respectively.

V. SCHEDULING OF A SINGLE DEEP LEARNING (DL) APPLICATION

A. BASELINE TASK-CLUSTERING SCHEDULER

As a baseline scheduling method, we partition the DNN into a set of sub-networks, map them onto processors, and run them in a pipelined fashion. The intuition for this is to minimize the communication overhead, also keeping the code complexity low. The mapping problem is defined as the task-clustering problem in which the input, a DL application, is partitioned into a number of PEs. A processing element(PE) is assigned a single task-cluster only. One exception is allowed for a logical PE of CPU, as illustrated in Figure 3 (a) where the CPU is assigned two task-clusters: the first cluster and the last cluster. If we attach the scheduler of the second iteration and move the cluster of one period, however, the schedule becomes as shown in Figure 3 (b) so that the restriction still holds. We implement this scheduler using ILP to find the best solution in this restricted solution space.

Let L = $\{L_1, L_2, \ldots, L_n\}$ be a set of layers, or tasks, in a DL application sorted in the topology order, and PE = $\{PE_1, PE_2, \ldots, PE_m\}$ be a set of logical PEs in the device.[3] E_i^j is the computation time of L_i on PE_j and $T_i^{j,k}$ is the communication time taken when transferring L_i's output from PE_j to PE_k.

There are three sets of constraints in the ILP formulation. First, a task is mapped onto only one PE. When M_i^j is a binary decision variable that indicates whether L_i is mapped to PE_j, the following constraint, expressed by Eq. (1), makes each task mapped onto only one PE. Let $pred(L_i)$ and $succ(L_i)$ be the set of preceding and succeeding layers of layer L_i, respectively.

$$\forall L_i \in L, \ \sum_{j=1}^{m} M_i^j = 1 \tag{1}$$

The second set of constraints is related to task-clustering mapping. We introduce two additional binary variables, dep_i^j and $Pipe_i^j$. The former indicates whether any predecessor of L_i is mapped on the PE_j. The $Pipe_i^j$ variable indicates whether a cluster or pipeline stage starts from layer L_i on PE_j. The definitions of dep_i^j and $Pipe_i^j$ are presented in Eq. (2) and Eq. (3), respectively:

$$dep_i^j = \begin{cases} 1, & \exists L_k \in pred(L_i), \ M_k^j = 1 \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

$$Pipe_i^j = \begin{cases} M_i^j, & i = 1 \\ M_i^j \wedge \neg dep_i^j, & \text{otherwise} \end{cases} \tag{3}$$

With these two variables, Eq. (4) tells that only one pipeline stage is allowed for each PE, except the PE onto which the first and last stages are mapped.

$$\forall PE_j \in PE, \ \sum_{i=1}^{n} Pipe_i^j - (M_n^j \wedge M_1^j) = 1 \tag{4}$$

Lastly, we need to specify dependency constraints. Let $Start(L_i)$ be the start time of a task or layer L_i, $End(L_i)$ be the end time of L_i, and $m(L_i)$ be the index of the PE onto which layer L_i is mapped. Then, the following two constraints of Eq. (5) and Eq. (6) enforce all dependencies between tasks to be satisfied.

$$\forall L_i \in L, \ \forall L_j \in succ(L_i),$$
$$Start(L_j) \geq End(L_i) + T_i^{m(L_i),m(L_j)} \tag{5}$$
$$\forall L_i \in L, \ \forall L_j \notin succ(L_i), \ j > i,$$
$$Start(L_j) \geq End(L_i) - ((1 - M_i^{m(L_j)}) \times \infty) \tag{6}$$

Note that we aim to maximize the throughput of a DL application, which is determined by the longest cluster in the pipelined execution. Thus we define the cluster execution time, $CT(PE_j)$, as follows:

$$CT(PE_j) = \sum_{i=1}^{n} M_i^j \times (E_i^j + \sum_{L_k \in succ(L_i)} T_i^{j,m(L_k)}) \tag{7}$$

[3] n is the number of layers in the DL application, and m is the number of PEs in the device.

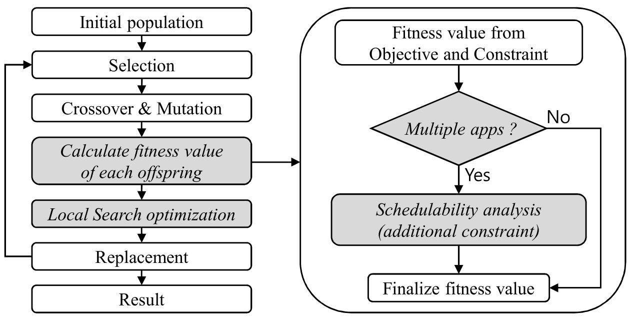

Initial population
Selection
Crossover & Mutation
Calculate fitness value of each offspring
Local Search optimization
Replacement
Result
Fitness value from Objective and Constraint
Multiple apps ?
No
Yes
Schedulability analysis (additional constraint)
Finalize fitness value

FIGURE 4. Overview of the proposed GA scheduler.

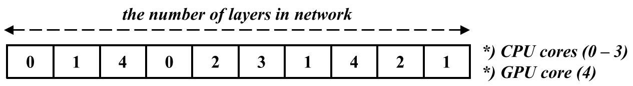


FIGURE 5. GA chromosome structure with PE configuration (1, 1, 1, 1).

Then the objective function is to minimize the longest cluster execution time, which is presented as follows:

$$minimize(\max_{PE_j \in PE}(CT(PE_j))) \tag{8}$$

B. PROPOSED GA-BASED SCHEDULER

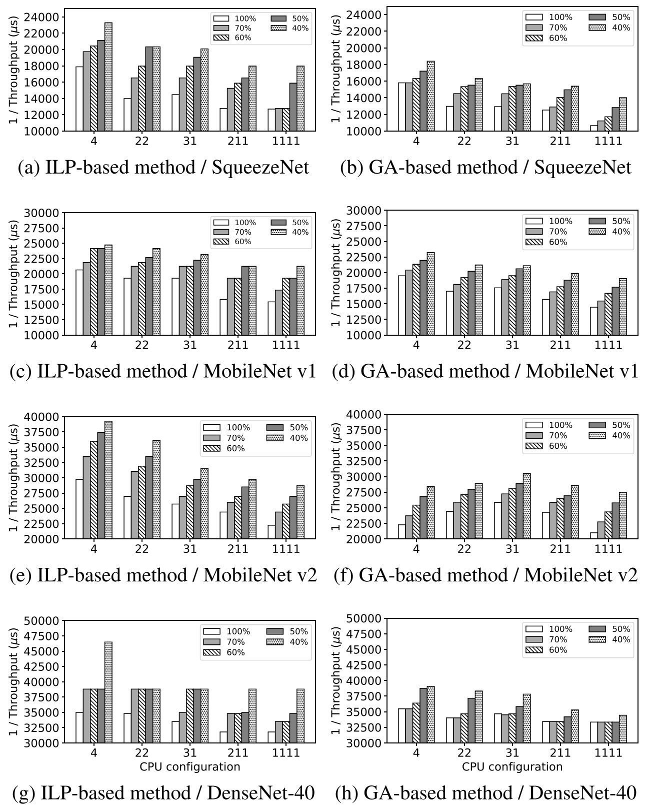
Genetic algorithm (GA) is a widely used meta-heuristic inspired by evolutionary processes in nature, where a solution of the problem is encoded as a chromosome, and the fitter survives to the next generation, populating new chromosomes with operations such as crossover and mutation on their chromosomes. The proposed scheduling algorithm is displayed in Figure 4. The overall flow and population generation procedure is not much different from the standard GA algorithm.

Figure 5 shows an example of a chromosome configured to solve the problem using the proposed GA scheduler. A chromosome consists of an array, where each element represents a layer, and the number in the array indicates the PE onto which the layer is mapped. For example, the chromosome shown in Figure 5 indicates that the first layer is mapped on PE 0 (CPU PE) and the third layer is mapped on PE 4 (GPU PE), and so on. Since encoding varies depending on the configuration of the multi-core CPU, we iterate this encoding with different number of PEs (i.e., different maximum number in the array) for each possible configuration.

The objective of the proposed scheduling is to maximize throughput and to minimize energy consumption. The CPU utilization should not be higher than some threshold to avoid overheating. Thus we define the fitness function with the following three terms and let GA find the solutions with the minimal fitness value. The first term is the inverse of the throughput that is obtained by Eq. (9) [25].

$$Throughput(graph) = \lim_{n\to\infty} \frac{n}{time\ to\ finish\ n\ iterations} \tag{9}$$

The second term is the total energy consumption, which is computed by multiplying the execution time of the mapped tasks on each PE and the power consumption of the PE. The



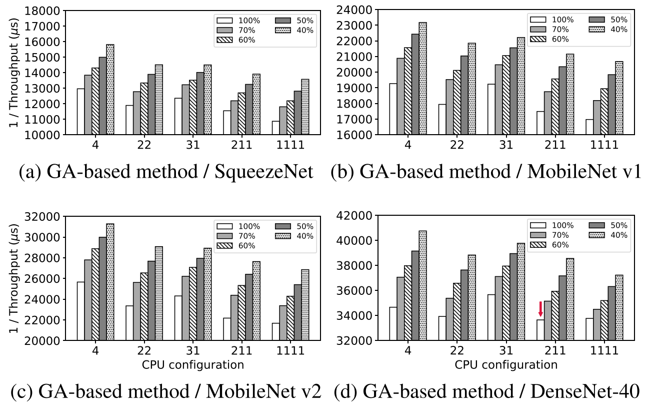
(a) ILP-based method / SqueezeNet (b) GA-based method / SqueezeNet

(c) ILP-based method / MobileNet v1 (d) GA-based method / MobileNet v1

(e) ILP-based method / MobileNet v2 (f) GA-based method / MobileNet v2

(g) ILP-based method / DenseNet-40 (h) GA-based method / DenseNet-40

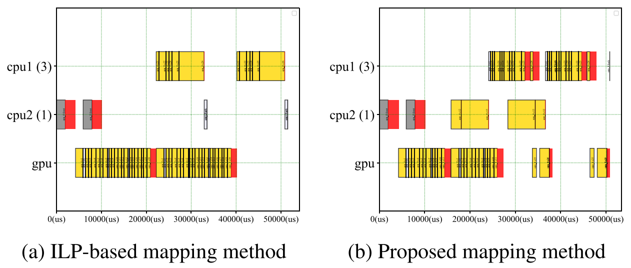
FIGURE 6. Comparison of the GA-based method and the ILP-based method (Device: Galaxy S9 / Unit: μs).



(a) GA-based method / SqueezeNet (b) GA-based method / MobileNet v1

(c) GA-based method / MobileNet v2 (d) GA-based method / DenseNet-40

FIGURE 7. Scheduling results from Hikey 970 (Unit: μs).



(a) ILP-based mapping method (b) Proposed mapping method

FIGURE 8. Scheduling results from two different methods.

third term is the penalty in case the CPU utilization is greater than a given utilization constraint.

C. EXPERIMENTAL RESULTS

The proposed scheduler is implemented with DEAP [26] and SPEA2 [27] is used as the selection algorithm, which is known to perform well for multi-objective problems. MobileNet v1 [28], MobileNet v2 [29], SqueezeNet [30], and DenseNet-40 (k = 32) [31] are selected as benchmark applications in this work.

1) THROUGHPUT PERFORMANCE

We first set the optimization criteria of single application scheduling to be the throughput performance ignoring energy consumption. Figure 6 shows the scheduling results of the GA-based scheduler and the ILP-based task-clustering scheduler. The x-axis represents the CPU configurations that determine how to exploit *intra-* and *inter-layer* parallelism on a CPU. The y-axis represents the average inference time of a single input image in μs. The throughput performance is the reciprocal of the average inference time. Since the typical CPU utilization limit to avoid overheating is within the range of [40%, 70%], we set the constraint within the range and compare the result with the one without any constraint (100%). Note that the CPU utilization is the average utilization across the whole CPU cores.

From Figure 6, we make three observations. First, as the CPU utilization constraint gets tighter, the inference time increases and the throughput decreases, as expected. In Figure 6(g), the inference time does not decrease with larger CPU utilization constraint, which is indicating that the GPU execution is the performance bottleneck.

Second, the throughput performance is the best with (1,1,1,1) PE configuration in most cases, which implies that exploiting *inter-layer* parallelism is more beneficial than exploiting *intra-layer* parallelism. From this observation, we decided to use (1,1,1,1) PE configuration for scheduling multiple applications, as will be explained in the later section. Figure 7 shows that (1,1,1,1) configuration also gives the best throughput performance for HiKey970. There is one exception in the proposed GA-based scheduling result: the inference time of (2,1,1) PE configuration, indicated by a red arrow, is smaller than (1,1,1,1) configuration when the utilization constraint is 100% as shown in Figure 7(d).

The third observation is that the GA-based scheduling outperforms the task-clustering scheduling in almost all cases. It is because the partitioning restriction imposed on the task-clustering method prunes the solution space too much. To understand the difference between the two methods, Figure 8 compares the results of scheduling methods for a single problem instance (SqueezeNet with (3,1) CPU configuration under 60% CPU utilization constraint). The figure shows that the GA-based method can better utilize the CPU cores than the task-clustering method.

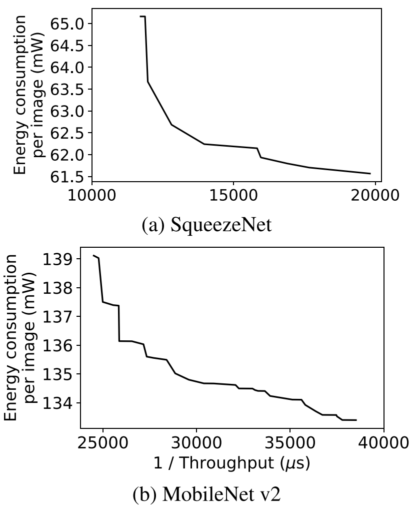


FIGURE 9. **Multi-objective scheduling results.**

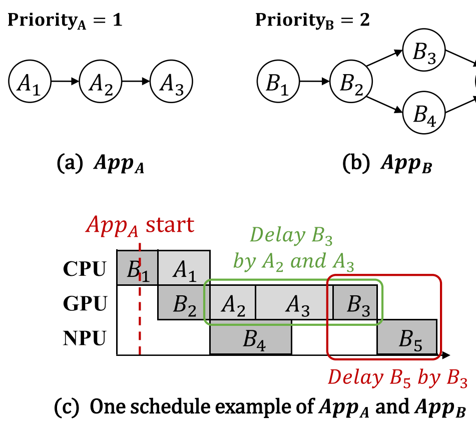


FIGURE 10. **Multiple application scheduling example.**

2) MULTI-OBJECTIVE SCHEDULING

As shown in Figure 9, the proposed GA scheduler generates Pareto optimal solutions in terms of throughput and energy consumption. Each Pareto optimal solution is associated with a different scheduling result. Thus a user can find the best trade-off from the results.

VI. SCHEDULING MULTIPLE DL APPLICATIONS

A. SCHEDULABILITY ANALYSIS

Since the schedule of an application affects the other applications, schedulability analysis can be performed only after the mapping and scheduling decisions made. On the other hand, we need to consider the schedulability when making the mapping and scheduling decisions. The proposed GA-based scheduler solves this cyclic-dependency naturally in an iterative fashion.

Suppose that we map two applications on three heterogeneous processors as shown in Figure 10 and App_A has a higher priority than App_B.[4] Figure 10(c) shows a scheduling example of App_A and App_B. The green box in Figure 10(c) illustrates that task B_3 in application App_B is delayed by the tasks A_2 and A_3 of application App_A which has higher priority. As we do not know the relative offset of App_A to App_B, we have to consider the worst-case scenario of interference from App_A in calculating the delay of task B_3. In addition, task B_5 on another PE is delayed due to the dependency with task B_3 as shown in the red box in Figure 10(c).

There are two methods to check the schedulability of multiple task graphs. One is to convert each task graph into a set of independent real-time tasks with relative deadlines and starting offsets. Dependency between tasks is expressed by the relative starting offsets of the tasks. The other method is the schedule-based analysis method proposed in [18].

The schedule-based analysis is based on the scheduling results of each application. For this analysis, the maximum resource demand (MRD) and maximum allowable interference (MAI) are computed. The former, MRD, means the maximum duration for which a high priority application can delay lower priority applications, while the latter (MAI) means the maximum duration for which a task can be delayed by higher priority tasks. MRD can be obtained by using the demand bound function (DBF) that returns the maximum processor execution time during a time interval given as an argument [32]. MAI is derived from the mobility concept of behavioral synthesis by subtracting the As-Soon-As-Possible (ASAP) schedule offset from the As-Late-As-Possible (ALAP) schedule offset [33].

For each task T_i, let P_i be the period of the task T_i, M(T_i) be the processor onto which T_i is mapped, and H(T_i) be the application set whose priority is higher than T_i. Also, let $dbf_{pe}(A, t)$ be the demand bound value of the tasks in an application A which are mapped onto pe during time interval t. Then, the application is schedulable if each task T_i satisfies the following equation: Eq. (10).[5]

$$MAI_{T_i} - \sum_{hp \in H(T_i)} dbf_{M(T_i)}(hp, P_i) \geq 0 \qquad (10)$$

Since we assume non-preemptive scheduling as GPU and NPU cannot support preemptive scheduling, we modify Eq. (10) such that it considers the maximum possible interference by a lower priority task. If we denote a set of applications whose priority is lower than T_i as L(T_i) and the computation time of task lp as C_{lp}, then the final equation becomes Eq. (11).

$$MAI_{T_i} - \sum_{hp \in H(T_i)} dbf_{M(T_i)}(hp, P_i) - \max_{lp \in L(T_i)} (C_{lp}) \geq 0 \qquad (11)$$

B. GA-BASED SCHEDULER

The scheduler shown in Figure 4 is applied to the scheduling of multiple applications with a similar chromosome structure. For the scheduling of multiple DL applications, the fitness

[4] A lower number means a higher priority.

[5] In the general case, after subtracting the interference of one higher priority application from the MAI, the MAI value should be updated iteratively, but this equation does not include that part for simplicity.

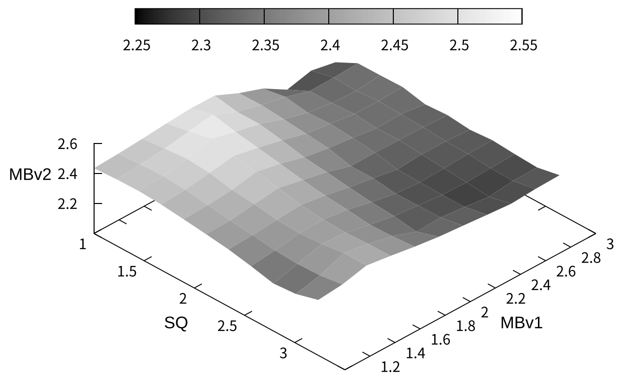


FIGURE 11. Pareto-optimal solutions in terms of relative response time when scheduling SqueezeNet(SQ), MobileNet v1(MBv1), and MobileNet v2(MBv2) .

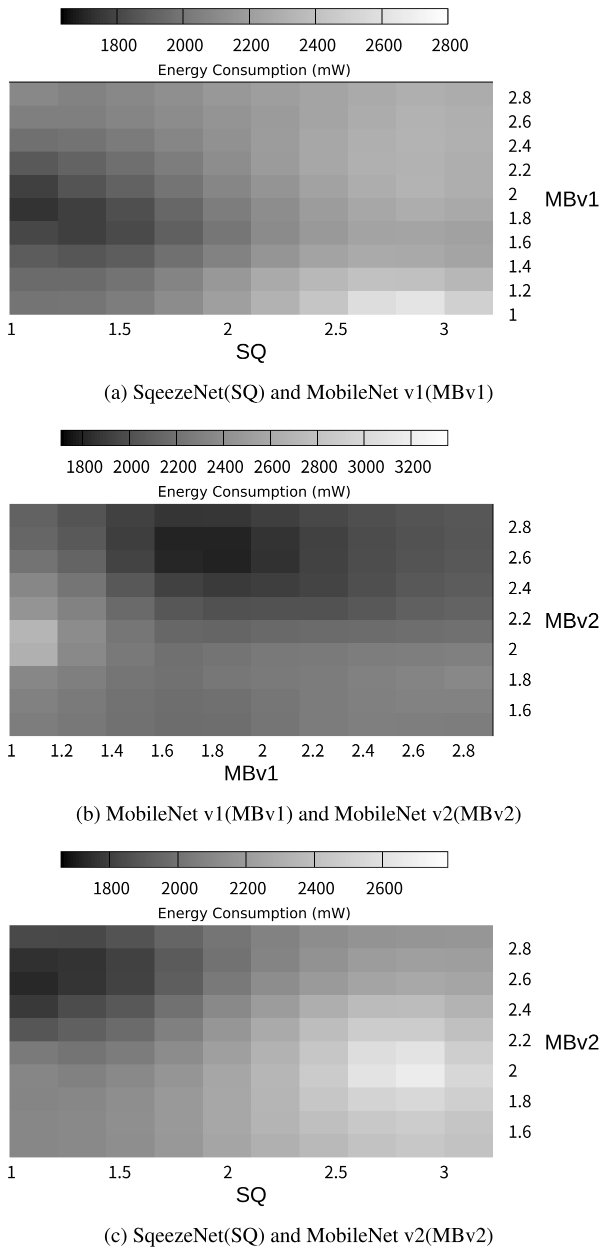


FIGURE 12. Pareto-optimal solutions in terms of energy when scheduling SqueezeNet(SQ), MobileNet v1(MBv1), and MobileNet v2(MBv2).

function is redefined since the objective and the constraints are changed. We set multiple objectives, minimizing the response time of each application, and minimizing the total energy consumption. Thus the fitness function has as many terms as the number of applications, each of which is the response time of the corresponding DL application, plus the term for the total energy. If the response time of an application is greater than the throughput constraint of the application, a large penalty is added to the term so that it cannot be selected.

C. EXPERIMENTAL RESULTS

The same configurations are used as in Section V-C except that the three CNNs are scheduled together and the objectives are different: to minimize response times of three applications and total energy consumption of the device. The CPU configuration is fixed to (1,1,1,1), as this configuration has resulted in better solutions for a single DL application in most cases.

Figure 11 and Figure 12 show the scheduling result of three benchmark CNNs with different periods; SqueezeNet, MobileNet v1, and MobileNet v2. Each application has periods of 33, 40, and 50 *ms*, respectively. In this experiment, the highest priority is assigned to SqueezeNet while the lowest one to MobileNet v2.

Figure 11 shows 288 Pareto-optimal solutions in terms of the relative response times of the three applications. We do not include energy information on this chart for a simple illustration. The grayscale indicates the response time of MobileNet v2: the darker is the color, the smaller is the response time. The x-axis indicates the relative response time of SqueezeNet to the fastest response time of SqueezeNet on the device. The minimum response times of SqueezeNet, MobileNet v1, and MobileNet v2 are 10.3 *ms*, 13.7 *ms*, and 16.9 *ms*, respectively. The y-axis and z-axis represent the relative response time of MobileNet v1 and MobileNet v2 respectively. We could observe that the application with the higher priority has the smaller response time, as expected: the minimal relative response times of SqueezeNet, MobileNet V1, and MobileNet v2 are 1, 1, and 1.43, respectively.

Figure 12 shows the change in energy consumption as the response time of two different application changes. The darker color indicates less energy consumption. In Figure 12(a), we can observe that the energy consumption is low when the response times of SqueezeNet and MobileNet v1 are 1 and 1.8, respectively.

VII. VERIFICATION WITH REAL HARDWARE PLATFORMS

Based on the scheduling results of a single DL application, we parallelize two benchmark networks, MobileNet v1 and MobileNet v2, on two hardware platforms: Galaxy S9 and HiKey970. Since there is no existent software framework to generate the parallelized code, we extended our own deep

TABLE 3. Verification results on two hardware platforms.

HW	Conf.	Net	Schedule	Actual	Error
HiKey970	C1	MN v1	18.5	19.7	-6.1%
HiKey970	C2	MN v1	21.1	19.8	6.6%
HiKey970	C1	MN v2	23.9	28.8	-17.0%
HiKey970	C2	MN v2	27.5	27.4	0.4%
Galaxy S9	C1	MN v1	15.4	16.5	-6.7%
Galaxy S9	C2	MN v1	23.5	23.2	1.3%
Galaxy S9	C1	MN v2	24.8	25.5	-2.7%
Galaxy S9	C2	MN v2	31.1 (31.2)	31.2	-0.3 (0)%

MN: MobileNet / Unit: sec

learning software framework [22]. For pipelined execution, we make a separate thread for each logical PE in CPU and implement double buffering for the tensor of which the source layer and the destination layer are mapped onto different PEs. And we use the conditional wait API for synchronization.

Due to the limitation imposed by ACL, however, we could not implement all PE configurations of the multi-core CPU, and we could not verify the scheduling of multiple DL applications. The current ACL implementation does not allow more than one task to use multi-threading for data-parallel execution simultaneously. For example, (2,2) configuration is not allowed since it would have two tasks, each of which in turn would create two threads. In (1,1,1,1) configuration, we disable the threading option for data-parallelism. Consequently, we could verify the scheduling results of a single DL application for the following two cases:

- (C1): With ACL scheduler disabled, scheduling with (1, 1, 1, 1) PE configuration
- (C2): With ACL scheduler enabled, scheduling with a quad-core CPU: (4) configuration

In this experiment, the scheduling objective is to maximize the throughput under no CPU utilization constraint, and comparison is made in terms of the processing time for 1000 images. Since we can change the DVFS governor and GPU frequency of HiKey970, unlike Galaxy S9, we chose the *performance* CPU governor and 767MHz GPU frequency for HiKey970. Neural Processing Unit (NPU) is not used even though HiKey970 has it.

Table 3 shows the verification results. It can be observed that the performance difference of the parallelized MobileNet v1 between the scheduling result and the measured one is less than 7% on both hardware platforms for two PE configurations. For MobileNet v2, the performance gap is relatively larger, particularly on HiKey970 for (C1) configuration. The main reason is that we underestimate the communication cost between processors in HiKey970. We found that Galaxy S9 uses only two CPU cores instead of four for (C2) configuration with MobileNet v2, turning off two CPU cores as the CPU utilization becomes lower than the given threshold that is not known to us. Thus we re-run the GA-based scheduler with profiling results with two CPU cores for the layers. Then, we could obtain no performance difference as shown with parentheses in the last row of Table 3.

VIII. CONCLUSION

In this paper, we propose a scheduling framework of deep learning (DL) applications for embedded devices with heterogeneous processors. We exploit both task-level and data-level parallelism in the scheduling of a DL application. In particular, we propose to consider five different PE configurations for a multi-core CPU to exploit both types of parallelism in various ways. We also consider the DVFS policy and the CPU utilization constraints to avoid thermal throttling in the profiling of tasks on each processing element. We use a GA-based method for scheduling a single DL application. The GA-based scheduler is also applied to the scheduling of multiple DL applications. We modify the schedule-based schedulability analysis to apply to non-preemptive tasks.

We verified the proposed scheduling methods by comparing the results with the measured ones in two real hardware platforms: Galaxy S9 and HiKey970. By considering several practical issues in the real implementation, we could achieve quite accurate estimation, with the error smaller than 7% except for one case. The primary source of error is the inaccuracy of task and communication profiling. Nonetheless, we believe the comparison with the actual implementation is a meaningful contribution.

REFERENCES

[1] M. Abadi. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: http://tensorflow.org/

[2] A. Paszke, "Automatic differentiation in PyTorch," in *Proc. NIPS Autodiff Workshop*, 2017, pp. 1–4.

[3] ARM. (2017). *Arm Compute Library*. [Online]. Available: https://developer.arm.com/technologies/compute-library

[4] Samsung Electronics. (2018). *Galaxy S9*. [Online]. Available: https://www.samsung.com/global/galaxy/galaxy-s9

[5] Kirin. (2018). *Hikey970*. [Online]. Available: https://www.96boards.org/product/hikey970/

[6] S. S. L. Oskouei, H. Golestani, M. Hashemi, and S. Ghiasi, "CNNdroid: GPU-accelerated execution of trained deep convolutional neural networks on android," in *Proc. ACM Multimedia Conf. (MM)*, 2016, pp. 1201–1205.

[7] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "RSTensorFlow: GPU enabled TensorFlow for deep learning on commodity Android devices," in *Proc. 1st Int. Workshop Deep Learn. Mobile Syst. Appl. (EMDL)*, 2017, pp. 7–12.

[8] Google. *Renderscript*. Accessed: Dec. 6, 2019. [Online]. Available: https://developer.android.com/guide/topics/renderscript/compute

[9] A. Mirhoseini, "A hierarchical model for device placement," in *Proc. ICLR*, 2018, pp. 1–11. [Online]. Available: https://openreview.net/forum?id=Hkc-TeZ0W

[10] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "μLayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in *Proc. 14th EuroSys Conf.*, 2019, p. 45.

[11] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "DeepX: A software accelerator for low-power deep learning inference on mobile devices," in *Proc. 15th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2016, pp. 1–12.

[12] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[13] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.

[14] S. K. Roy, R. Devaraj, A. Sarkar, S. Sinha, and K. Maji, "Optimal scheduling of precedence-constrained task graphs on heterogeneous distributed systems with shared buses," in *Proc. IEEE 22nd Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2019, pp. 185–192.

[15] H. Zhao and R. Sakellariou, "Scheduling multiple DAGs onto heterogeneous systems," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2006, p. 14.
[16] G. Xie, G. Zeng, L. Liu, R. Li, and K. Li, "High performance real-time scheduling of multiple mixed-criticality functions in heterogeneous distributed embedded systems," *J. Syst. Archit.*, vol. 70, pp. 3–14, Oct. 2016.
[17] H. I. Ali, B. Akesson, and L. M. Pinho, "Generalized extraction of real-time parameters for homogeneous synchronous dataflow graphs," in *Proc. 23rd Eur. Int. Conf. Parallel, Distrib., Netw.-Based Process.*, Mar. 2015, pp. 701–710.
[18] S.-H. Kang, D. Kang, H. Yang, and S. Ha, "Real-time co-scheduling of multiple dataflow graphs on multi-processor systems," in *Proc. 53rd Annu. Design Automat. Conf. (DAC)*, 2016, p. 159.
[19] H. Tomiyama, I. Taniguchi, L. Meng, and Y. Liu, "A dual-mode scheduling approach for task graphs with data parallelism," *Int. J. Embedded Syst.*, vol. 9, no. 2, pp. 147–156, 2017.
[20] H. Yang and S. Ha, "Pipelined data parallel task mapping/scheduling technique for MPSoC," in *Proc. Design, Automat. Test Eur. Conf. Exhibit.*, Apr. 2009, pp. 69–74.
[21] ARM. (2018). *Arm NN*. [Online]. Available: https://www.arm.com/products/silicon-ip-cpu/machine-learning/arm-nn
[22] D. Kang, E. Kim, I. Bae, B. Egger, and S. Ha, "C-GOOD: C-code generation framework for optimized on-device deep learning," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.
[23] A. Frumusanu. (2018). *The Samsung Galaxy S9 and S9+ Review*. [Online]. Available: https://www.anandtech.com/show/12520/the-galaxy-s9-review/
[24] (2018). *Hisilicon Kirin 970—Android Soc Power & Performance Overview*. [Online]. Available: https://www.anandtech.com/show/12195/hisilicon-kirin-970-power-performance-overview/
[25] T.-H. Shin, H. Oh, and S. Ha, "Minimizing buffer requirements for throughput constrained parallel execution of synchronous dataflow graph," in *Proc. 16th Asia South Pacific Design Automat. Conf. (ASP-DAC)*, Jan. 2011, pp. 165–170.
[26] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *J. Mach. Lang. Res.*, vol. 13, pp. 2171–2175, Jul. 2012.
[27] E. Zitzler, "Spea2: Improving the strength Pareto evolutionary algorithm," Swiss Federal Inst. Technol. (ETH) Zürich, Zürich, Switzerland, TIK-Rep. 103, 2001, vol. 103.
[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*. [Online]. Available: http://arxiv.org/abs/1704.04861
[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.
[30] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*. [Online]. Available: http://arxiv.org/abs/1602.07360
[31] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, vol. 1, no. 2, p. 3.
[32] S. K. Baruah, A. K. Mok, and L. E. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *Proc. 11th Real-Time Syst. Symp.*, 1990, pp. 182–190.
[33] B. M. Pangrle and D. D. Gajski, "Slicer: A state synthesizer for intelligent silicon compilation," in *Proc. ICCAD*, 1987, pp. 42–45.

DUSEOK KANG (Member, IEEE) received the B.S. degree from the Department of Semiconductor System Engineering, Sungkyunkwan University, Suwon, South Korea, in 2014. He is currently pursuing the Ph.D. degree of the M.S.-Ph.D. integrated program with Seoul National University, Seoul, South Korea. His research interests include embedded deep learning and hardware-aware software optimization. He took first place in the 2017 IEEE Low Power Image Recognition Challenge (LPIRC). In the following year, in 2018, he ranked first on track 2 and second on track 3 at LPIRC.

JINWOO OH received the B.S. degree from the Department of Computer Science and Engineering, Soongsil University, Seoul, South Korea, in 2018. He is currently pursuing the M.S. degree with Seoul National University, Seoul. His research interest includes hardware-aware software optimization.

JONGWOO CHOI received the bachelor's degree from the Business School, Korea University, Seoul, South Korea, in 2015. He served in South Korean military as a Public Service Agent, from 2006 to 2009, and worked as a Lecturer in SBS Game Academy, in 2016. He translated and published Unity 5 game programming optimization by Han-Bit Media, in 2017, and has also published "Task-Processor ILP Scheduling for CNN on Heterogeneous Computing Environment Embedded Device" by KIISE, in 2018. He is currently attending the Co-design and Parallel Processing Lab, Seoul National University, as the master's degree student.

YOUNGMIN YI received the B.S. degree in computer engineering and the 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, CA, USA, from 2007 and 2009, and as a Senior Researcher at the Samsung Advanced Institute of Technology, from 2009 to 2010, before he joined the School of Electrical and Computer Engineering, University of Seoul, where he is currently a Professor. His research interests include on-device deep learning, algorithm/architecture codesign for heterogeneous manycore platforms, and GPU computing.

SOONHOI HA (Fellow, IEEE) received the B.S. and M.S. degrees in electronics engineering from Seoul National University, Seoul, South Korea, in 1985 and 1987, respectively, and the Ph.D. degree in electrical engineering and computer science from the University of California at Berkeley, Berkeley, CA, USA, in 1992. He is currently a Professor with Seoul National University. His current research interests include HW/SW codesign of embedded systems, embedded machine learning, and the IoT. He is a member of the ACM. He has actively participated in the premier international conferences in the EDA area, for instance, serving CODES+ISSS 2006, ASPDAC 2008, as the Program Co-Chair and ESWeek 2018 as the General Chair.