Received 4 May 2023, accepted 29 May 2023, date of publication 1 June 2023, date of current version 7 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3281913

RESEARCH ARTICLE

A Novel Technique to Support Deep Learning Applications in a Model-Based Embedded Software Design Methodology

JANGRYUL KIM, (Graduate Student Member, IEEE), JAEWOO SON, (Graduate Student Member, IEEE), AND SOONHOI HA, (Fellow, IEEE)
Department of Computer Engineering, Seoul National University, Gwanak-gu, Seoul 08826, South Korea

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

This work was supported by Samsung Electronics Company Ltd., under Grant IO201210-07941-01.

ABSTRACT As deep learning applications are getting popular in embedded systems, how to support deep learning applications in the model-based embedded software design methodology becomes a challenging problem. A previous solution is to represent each deep learning application with a model. However, it requires significant efforts to translate specifications and obtain good performance by applying optimization techniques to deep learning applications. In this work, we propose a novel methodology that leverages the benefits of using deep learning software development kit (SDK) for performance optimization. In the proposed methodology, we first obtain the Pareto-optimal mapping solutions of deep learning applications using the SDK associated with the hardware platform. Afterward, we perform mapping of dataflow tasks and selection of mapping solutions of deep learning (DL) applications together through a genetic algorithm. Experiments with a real-life example and randomly generated graphs show that we could reduce at least 5% of the maximum utilization compared to our previous work that maps DL applications and dataflow applications sequentially.

INDEX TERMS Model-based software design, mapping exploration, deep learning applications, heterogeneous processors.

I. INTRODUCTION

Model-based design (MBD) methodology is widely adopted for embedded software development since it enables us to specify an application behavior independently of the hardware platform that is continually evolving over time. Since appropriate models vary depending on the application domain, various models and methodologies have been proposed. For example, the statechart model is widely used for control-oriented applications [1], and a timed discrete event model is used for power-aware real time scheduling [2]. The works of [3] and [4] adopt the dataflow model for the specification of multimedia or streaming applications. In addition, there exist some works that use more than one model: a combination of the dataflow model and the finite state machine is used in [5], [6] and [7], and the work of [8] deploys both the discrete event model and the dataflow model.

In this work, we are concerned about the embedded software design methodology based on a dataflow model. In a dataflow graph, a node represents a computation task, and an arc indicates a data dependency between adjacent tasks. A key benefit of dataflow models is that it is easy to exploit the task-level parallelism of an application by simply mapping nodes to processing elements in a given hardware platform. If the number of data samples that are transferred on each arc is known at compile time, a dataflow model is said to be *decidable* [9]. For a decidable dataflow, we can determine the mapping and scheduling of tasks at compile time and detect some critical errors in the specification, such as buffer

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

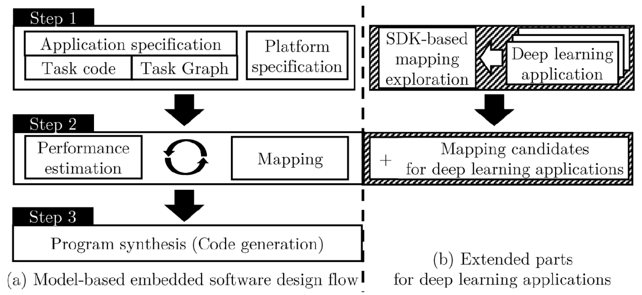

Step 1
Application specification
Task code
Task Graph
Platform specification
Step 2
Performance estimation
Mapping
Step 3
Program synthesis (Code generation)
(a) Model-based embedded software design flow
SDK-based mapping exploration
Deep learning application
Mapping candidates for deep learning applications
(b) Extended parts for deep learning applications

FIGURE 1. Overall flow of the model-based embedded software design and the proposed extension.

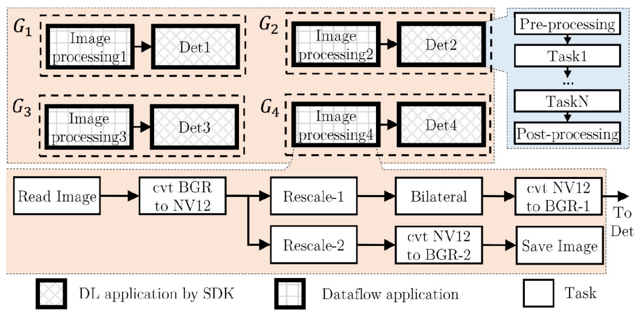


FIGURE 2. A motivational example.

overflow and deadlock ([10], [11]). In this work, we assume that a decidable dataflow model is used in the model-based design methodology for embedded software development.

Figure 1 (a) shows the traditional embedded software design flow based on the dataflow model. Each application is represented by a dataflow graph in which the internal behavior of a task is defined by the task code written in a conventional programming language such as C or C++. The hardware platform information on the available processing elements and communication architecture is given separately from the application specification. For a given hardware platform, we find an optimal mapping of tasks onto processing elements by comparing the estimated performance among various mappings, which is referred to as the design space exploration (DSE) step. Lastly, the application code on each processing element is generated based on the mapping decision made in the DSE step.

Remarkable advances in deep learning (DL) techniques make DL applications getting popular in embedded systems. Figure 2 displays a motivational example where DL networks and other applications are running together. The *Image processing* application is represented by a task graph that consists of eight tasks that process images. After processing the image, the *Det* network gets the output data from the *Image processing* application and performs object detection. The object detection network consists of many layers for inference in addition to pre-/post-processing tasks. The example has four sets of such combinations of a dataflow application and a DL application as shown in Figure 2.

How to support DL applications in the model-based design methodology has emerged as a challenging problem. Even though the layer structure of a DL application looks similar

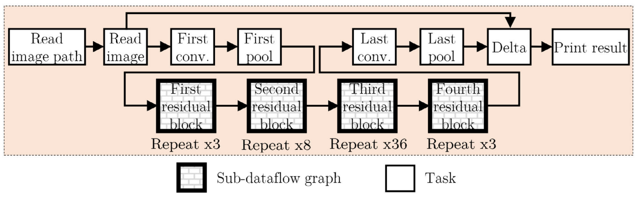


FIGURE 3. A dataflow graph specification of *Resnet152* in [7].

to a dataflow graph, it is challenging to specify it with a decidable dataflow model. To tackle this problem, previous studies ([12], [13]) have proposed to specify DL networks with a specific dataflow model in order to treat them with other applications in the model-based design framework. The former work [12] extends a dataflow model to specify loop structures explicitly, while the latter [13] transforms a DL network into a cyclo-static dataflow (CSDF) graph [11]. Figure 3 displays the *Resnet152* network represented as a dataflow graph using an extended SDF model called SDF/L [12]. The bold box indicates a loop construct whose internal subgraph is repeated as many times as the given parameter value.

However, this approach has the following drawbacks. First, it requires a lot of effort to specify a DL application with a dataflow model. The number of data samples produced or consumed per task execution needs to be explicitly specified, and the internal behavior of tasks may need to be redefined. Second, the number of tasks grows significantly to make the DSE step more difficult as a DL network usually consists of numerous layers. Third, the previous works usually target CPU-GPU heterogeneous processor systems without including a neural processing unit (NPU). Since recent hardware platforms tend to include an NPU for accelerating deep learning applications, it is necessary to consider NPUs in the design methodology. Last but not least, it is not possible to apply the optimization techniques that are provided by the deep learning software development kit (SDK). As a result, the synthesized DL application from the MBD framework is likely to perform poorly compared with the conventional deep learning application that runs with the deep learning SDK.

To overcome those shortcomings of the previous studies, we propose a novel technique to support DL applications in a model-based design methodology, leveraging the optimization capability of a deep learning SDK. Figure 1 (b) shows the proposed extensions highlighted by the slash-pattern boxes. First, we find the Pareto-optimal mapping candidates for each DL application onto available processing elements with multiple objectives, such as cost, latency, utilization of each processing element, and so on. We explore the design space of partitioning and mapping of the application for pipelined execution, similar to the first phase of the work [14], independently of the MBD framework. The deep learning SDK optimizes layers by combining or changing layers, depending on the mapping decision. Since per-layer profiling might not be possible on hardware accelerators, it is not appropriate to explore mappings on a layer-by-layer basis [14]. Therefore, we select mapping candidates by pipelining the network in this phase.

In the DSE step, we explore the mapping candidates of DL applications and the mappings of dataflow applications simultaneously. Note that we do not explore the mapping of DL layers in this step. Instead, we explore the mapping of dataflow applications for each combination of mapping candidates of DL applications. In this step, we check the schedulability of applications with deadline constraints by the worst-case response time (WCRT) analysis.

The proposed extension is used by a model-based design framework [7]. In the last step of program synthesis, the interface code between dataflow applications and the DL applications is automatically synthesized. The viability of the proposed methodology is verified with a non-trivial aforementioned motivational example running on a real hardware platform and randomly generated task graphs.

The main contributions of this paper can be summarized as follows:

- We propose a novel technique to support DL applications in a model-based design methodology without translating DL applications to dataflow models, leveraging the optimization capability of a deep learning SDK.
- Differently from our previous work that maps the DL applications and dataflow applications sequentially, we propose a mapping technique to consider them together, using an evolutionary algorithm.
- The proposed methodology supports heterogeneous processor systems that include an NPU, considering the characteristics and limitations of the hardware platform and the associated SDK.
- Experiments with a real-life example and randomly generated graphs show that we could reduce at least 5% of the maximum utilization compared to our previous work that maps DL applications and dataflow applications sequentially.

The rest of the paper is organized as follows. In the next section, we review the related works. After stating the background and system model in Section III, we introduce the proposed methodology in Section IV. After the experimental results are presented and discussed in Section V, conclusions are made in Section VI.

II. RELATED WORKS

We review the previous studies in the following three main subjects related to the proposed methodology: mapping of multiple dataflow applications, mapping of deep learning applications, and integrating deep learning applications into the model-based design.

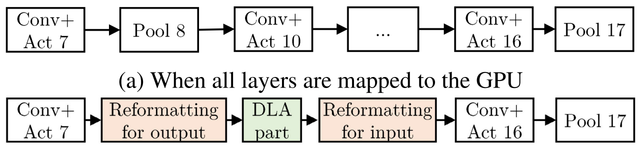
A. MAPPING OF MULTIPLE DATAFLOW APPLICATIONS

Since the mapping and scheduling of a dataflow graph onto a multiprocessor system is an NP-hard problem [15], many approximate methods for finding the optimal solution have been proposed such as heuristics based on list scheduling ([16], [17]) and [16] explored mappings by heuristics, meta-heuristics using a genetic algorithm (GA) [18]. While most of works focus on the mapping and scheduling of a single dataflow graph, only a few works deal with the mapping of multiple dataflow graphs onto multiple processing elements, to the best of our knowledge. In case there exist real-time constraints on dataflow applications, we need to check a mapping solution satisfies those constraints by schedulability analysis for throughput constraint or worst-case response time analysis for latency constraint, which is recognized as a very challenging problem in real-time community [19]. A simple solution to avoid this difficulty is to map each dataflow graph onto a disjoint set of processors, avoiding interference between applications on the same processor. Then, we can map each dataflow separately onto the assigned processors.

Schor et al. [20] proposed an evolutionary algorithm to find a mapping to minimize the maximum utilization. Kang et al. [21] proposed a two-step approach. In the first step, they find a set of Pareto-optimal parallel schedules of each individual dataflow graph using a multi-objective evolutionary algorithm. In the second step, another evolutionary algorithm is to used to find the best combination of Pareto-optimal solutions of all dataflow graphs, aiming to minimize resource usage. For each mapping candidate, worst-case response time analysis is performed to check if the deadline constraint is satisfied for each dataflow graph.

B. MAPPING OF MULTIPLE DEEP LEARNING APPLICATIONS

As deep learning applications are getting popular in embedded systems, extensive studies have been conducted recently to find the optimal mapping of DL applications specified by the associated SDK on a heterogeneous hardware platform. Xiang et el. [22] partition deep neural networks (DNNs) into pipeline stages, and map stages to processing elements by a heuristic. They focus on increasing the schedulability of multiple DNNs by balancing the utilization among PEs. While early works, including [22], considered CPU-GPU heterogeneous systems as the target hardware platform, recent works consider the DL hardware accelerators, also called NPU ([14], [23], [24]). Pujol et al. [23] consider an entire network as the mapping unit to a single PE without partitioning to leverage the associated SDK with each PE, assuming that the number of DL applications is more than the processing elements. Kang et al. [24] explore the per-layer mapping of DNNs with a GA, assuming that the layer-wise profiling information is given for each DL application. Even though they considered NPUs in the mapping step, no experiment with the NPU on a real hardware platform was made. Kim et al. [14] have recently proposed a two-stage mapping exploration methodology on heterogeneous processor systems including NPUs. In the first stage, they use a genetic algorithm to find a set of Pareto-optimal mappings of each DL network for pipelined execution, aiming to minimize the end-to-end latency and average power consumption of each processing element. In the second stage, they find a sub-optimal combination of mapping solutions for all applications. The



(a) When all layers are mapped to the GPU

(b) When layers #8-#14 are mapped to DLA, and others are to the GPU

FIGURE 4. An example kernel structure of Yolov2 network.

first step of our proposed technique, shown in Figure 1 (b), is inspired by this work. Note that all works ([14], [22], [23], [24]) consider the mapping of multiple DL applications only without considering other tasks running concurrently.

C. INTEGRATING DEEP LEARNING APPLICATIONS INTO THE MODEL-BASED DESIGN

Since supporting DL applications in the model-based design methodology is a recent demand, there exist only a few previous studies that tackle this problem in the model-based design methodology. The work of [12] introduces an extended SDF model, called the SDF/L model, that specifies two types of loop structures explicitly and shows how to specify a DL application with the SDF/L model. They leave it as future work on how to perform task-mapping of the SDF/L graph, exploiting the data-level parallelism. Minakova et al. [13] transform a CNN (convolutional neural network) to an SDF graph and find the mapping of SDF by using the genetic algorithm. After the mapping decision is made, they translate the CNN network to a CSDF graph that is partitioned into sub-graphs that are run on each processing element. In this work, a CNN network has to be translated into two different models, the SDF model for mapping, and the CSDF model for code generation and execution. Such translation requires considerable effort. Since the translated SDF model has a wide range of sample rates, finding an optimal mapping onto multiple processors itself is a challenging problem. Both works ([12], [13]) do not consider the mapping exploration of multiple applications.

Recently, Jeong et al. [25] proposed a methodology based on the genetic algorithm to explore the mappings for multiple dataflow graphs on CPU-GPU heterogeneous system with a deep learning application. In their work, they assume that the mapping for the deep learning application is fixed and find the mapping of dataflow graphs, aiming to reduce the worst-case response time for each dataflow application. To the best of our knowledge, the proposed method is the first approach that supports multiple deep learning applications running concurrently with other dataflow applications in a model-based design methodology on a heterogeneous hardware platform, including NPUs.

III. BACKGROUND AND SYSTEM MODEL

A. NVIDIA JETSON AGX XAVIER BOARD, TensorRT

In this work, we use an NVIDIA Jetson AGX Xavier (Xavier) board as a target hardware platform. It includes one Volta GPU and two NPUs, called DLAs(deep learning accelerators), in addition to an octa-core ARMv8 CPU. NVIDIA provides TensorRT as the SDK for fast inference on NVIDIA devices, including GPU and DLA. While a DLA is a power-efficient accelerator, its computation ability is weaker than the Volta GPU. In the Xavier board, there is a unified memory shared by the CPU and GPU.

TensorRT internally applies some optimization techniques such as layer fusion. Figure 4 (a) shows the kernel composition of the Yolov2tiny network [26] when all layers are mapped to the GPU. Note that a convolution layer is fused with the following activation layer to form a single kernel by TensorRT. Suppose we map some layers (layer #8 to layer #14) to a DLA. Then as shown in Fig. 4(b), two kernels are added automatically by TensorRT to the sub-network mapped to the GPU for interfacing with the DLA.

There are some restrictions imposed by the device and its SDK. While it is possible to profile the execution time of each layer on a GPU, per-layer profiling on a DLA is not supported. The number of sub-networks that can be mapped to two DLAs is also limited. And TensorRT does not support a CPU to execute the inference. We need to consider those restrictions when mapping the DL applications.

TABLE 1. Notations used in a system model.

Notation	Description
pe and $\mathcal{PE}$	A PE and a set of PEs ($pe \in \mathcal{PE}$)
PE_{proc}	A set of PEs for processor type $proc$ ($G_i \in \mathcal{G}$)
G_i and $\mathcal{G}$	A group and a set of groups ($G_i \in \mathcal{G}$)
A_j^i	A dataflow application in G_i
D_k^i	A DL application in G_i
V_x^i and E_x^i	A set of tasks and a set of edges in A_x^i or D_x^i
$\tau_y^{i,x}$	A task in A_x^i or D_x^i
$C(\tau_y^{i,x})$	The WCET of $\tau_y^{i,x}$
p^i and pr^i	A period and priority of G_i
$L(D_k^i)$	The average latency of D_k^i
$ET(pe, G_i)$	The execution time on pe when running G_i
$R(G_i)$	The WCRT of G_i
$Map(pe)$	A set of mapped tasks on pe
$U(pe)$	The utilization of pe
$Cand(D_k^i)$	The mapping candidates of D_k^i

B. SYSTEM MODEL

In this section, we describe the system model assumed in this work using the notations summarized in Table 1.

1) ARCHITECTURE SPECIFICATION

The target hardware platform consists of heterogeneous processing elements (PEs), $\mathcal{PE}$. In this work, three processor types are used in the Xavier board: CPU, GPU, and DLA. And a set of PEs for each processor type is represented as PE_{cpu}, PE_{gpu}, and PE_{dla}, respectively.

2) APPLICATION SPECIFICATION

Two types of applications are distinguished. DL applications are specified by a DL SDK, TensorRT in this work, while other applications are specified by a dataflow graph. We denote a given set of applications in each type as $\mathcal{D}$ and

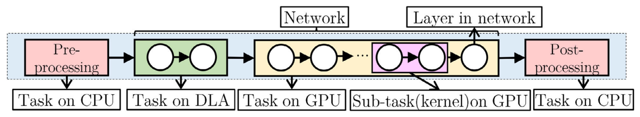


FIGURE 5. Task/Sub-task definition on the deep learning application specified by SDK.

$\mathcal{A}$, respectively. As shown in Figure 2, an edge between two applications denotes the data dependency between them. The connected applications form an application group, denoted by G_i. Group G_i is defined by a tuple $\langle A^i, D^i, E^i, p^i, pr^i \rangle$. $A^i_j \in \mathcal{A}$ and $D^i_k \in \mathcal{D}$ are applications that belong to group G_i. The last three elements indicate edges between applications, the invocation period, and the priority, respectively. There may exist multiple groups, and the set of groups is denoted as $\mathcal{G}$. A dataflow application A^i_j is characterized by a tuple $\langle V^i_j, E^i_j \rangle$, where V^i_j and E^i_j represent the set of tasks and the set of edges between tasks, respectively. For the dataflow application, A^i_j, task $\tau^{i,j}_m$ is naturally defined by the model. For instance, in Figure 2, the *Image processing* application consists of eight tasks inside.

For a DL application D^i_k, however, it is characterized by a tuple $\langle V^i_k, E^i_k \rangle$ only after pipelining of the application is made, where V^i_k is a set of tasks and E^i_k is a set of edges between tasks. How to pipeline a DL application is explained in the next section. After pipelining decision is made, the DL application consists of multiple pipeline stages, each of which is mapped to a PE. Figure 5 shows an example that has four pipeline stages colored differently. Each pipeline stage is defined as a task for DL application, $\tau^{i,k}_n \in V^i_k$. For the task mapped to the GPU colored yellow, the SDK forms a set of kernels after optimization. The purple box in the yellow GPU-mapped task represents a kernel that merges two layers after layer fusion. Since a kernel is a unit of profiling, we define each kernel as a sub-task in the GPU.

Since per-kernel profiling is not possible on a DLA, however, the entire set of layers becomes a single task on a DLA. A DL application has a pre-processing task that feeds input data to the inference body and a post-processing task that processes the output data. Those tasks should be mapped to the CPU core. Note that a pipelined DL application has a chain structure of tasks, which is also assumed in [22] and [14] since there is a dependency between pipeline stages and the execution order of kernels is set by the SDK [27]. In summary, a DL application consists of chain-structured tasks ($\tau^{i,k}_n \in V^i_k$).

3) SCHEDULING SPECIFICATION

We denote the worst-case execution time (WCET) of task $\tau^{i,x}_y$ as $C(\tau^{i,x}_y)$. The invocation period of task $\tau^{i,x}_y$ is denoted as $P(\tau^{i,x}_y)$, which is the same as p^i, the period of graph G_i. We assume that the graphs run periodically with the implicit deadline assumption that the period becomes the relative deadline. Thus, the deadline of group G_i is its period, p^i. The average latency of a group G_i is represented as $L(G_i)$, and the execution time on each processing element $pe \in \mathcal{PE}$ is denoted by $ET(pe, G_i)$. Similarly, the latency of a DL application D^i_k is denoted by $L(D^i_k)$. Meanwhile, $R(G_i)$ indicates the worst-case response time (WCRT) of a group G_i, and it is necessary to check whether the WCRT violates the deadline. The calculation of the WCRT is explained in Section IV-B3.

Even though a group is assigned a priority, pr^i, it is applicable only for the tasks mapped on the CPU. Since there is no way to set the priority of a task on a DLA, tasks mapped on a DLA are executed in the FIFO order. Even though there are two priority levels in GPU, they are usually not used. Similar to DLAs, the GPU executes the mapped tasks in the FIFO order.

We denote the set of mapped tasks on processing element $pe \in \mathcal{PE}$ as $Map(pe)$. Based on the mapped tasks on pe, we calculate the utilization of pe, $U(pe)$, as $\sum_{\tau^{i,x}_y \in Map(pe)} (C(\tau^{i,x}_y))/p^i$. In the proposed methodology, we choose Pareto-optimal mapping candidates, explained in the next section. The set of mapping candidates of D^i_k is represented by $Cand(D^i_k)$ where $D^i_k \in \mathcal{D}$, and the x-th candidate is denoted by $Cand(D^i_k, x)$.

IV. PROPOSED METHODOLOGY

In step 1, a set of Pareto-optimal mapping candidates for each DL application is obtained independently and applied to the extended model-based design framework as additional input information. In step 2, we try to find an optimal mapping of tasks by a meta-heuristic with objectives. For each DL application, we select a mapping candidate while we decide on the mapping of dataflow tasks simultaneously in this step. The last step is to generate the target code for each processing element, which will not be discussed in this work.

A. STEP 1: FINDING THE PARETO-OPTIMAL MAPPING SOLUTIONS OF EACH DL APPLICATION

The problem addressed in this step is summarized as follows.

- **Input:** All deep learning applications in $\mathcal{D}$ are provided.
- **Objective:** We define multiple objectives to minimize the latency of each deep learning application $D^i_k \in \mathcal{D}$ and the execution time on GPU and DLA, which can be described in Equation 1.

$$\begin{aligned} &Minimize : (\, L(D^i_k),\, ET(D^i_k, pe)\,) \\ &where\ D^i_k \in \mathcal{D},\ pe \in PE_{gpu}\ and\ PE_{dla} \end{aligned} \tag{1}$$

- **Problem:** For each deep learning application $D^i_k \in \mathcal{D}$, find all Pareto-optimal mapping candidates. i.e. mapping of D^i_k: $D^i_k \rightarrow Cand(D^i_k)$.
- **Output:** The set of Pareto-optimal mappings of each deep learning application, $Cand(D^i_k)$. A mapping candidate indicates how a DL application is partitioned into pipeline stages and to which PEs the pipeline stages are mapped.

To solve this problem, the genetic algorithm is used in our implementation of the proposed methodology, as there are publicly available GA optimizers that are well-maintained.

TABLE 2. Mapping options for pipelining of a DL application.

Options	# of stages	Composition of PEs
A	2	DLA0 - GPU
B	3	GPU - DLA0 - GPU
C	3	DLA0 - DLA1 - GPU
D	4	GPU - DLA0 - DLA1 - GPU
E	1	GPU

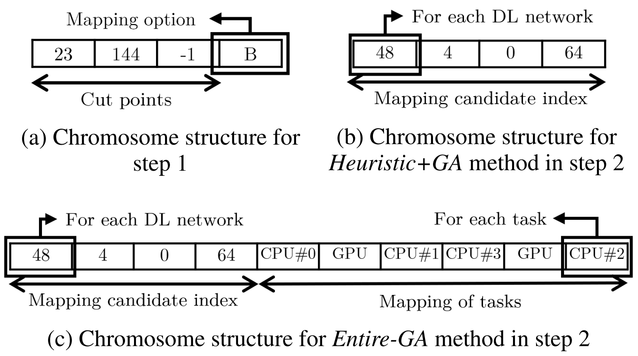


FIGURE 6. Chromosome structures: (a) for step1, (b) for *Heuristic+GA* method in step2, and (c) for *Entire−GA* method in step2.

Other meta-heuristics that support multiple objectives can also be used.

Similar to [14], we organize the mapping options as shown in Table 2 when partitioning the object detection application, *Det*, in the motivational example. For example, option *C* indicates the layers are partitioned into three stages. The pre-/post-processing tasks are mapped to the CPU while the inference body is partitioned into three pipeline stages that are mapped to two DLAs and GPU, respectively. Note that DLA0 and DLA1 are interchangeable without difference in terms of performance. The last pipeline stage is mapped onto the GPU in all mapping options since the last stage contains some layers that cannot be executed on a DLA. And no stage is mapped to the CPU. Those restrictions imposed by TensorRT need to be considered in the organization of the mapping options. It means that a set of mapping options may vary depending on the DL applications and the hardware platform.

After mapping options are identified, mapping candidates of the DL application are represented with the chromosome structure described in Figure 6 (a). The length of the chromosome is decided by the maximum number of pipeline stages. Since the maximum number of stages is four in Table 2, we set three genes for pipelining cut-points and one gene for the mapping option. In the example of Figure 6 (a), layers #0∼#23 and layers #24∼#144 are mapped to the GPU and DLA0, respectively, since the mapping option is option *B*. And the rest of the layers are also mapped to GPU. If there is no corresponding cut-point, the gene has a value of −1.

We define multiple objectives for GA fitness evaluation as described in Equation (1): end-to-end latency, the execution time on GPU, and the execution time on DLA and find Pareto-optimal solutions as mapping candidates for each deep learning application. We use the measured values as fitness values while running the network on a real board for each mapping candidate. Analytical performance estimation is not feasible because the layer-wise execution time could not be obtained for DLAs and the effect of optimization techniques of TensorRT on the performance cannot be estimated.

Suppose that a mapping is selected for a DL application, *Det*. Then the DL application can be transformed into a chain-structured task graph as shown in Figure 5. The dependency between the *Image processing* application and *Det* remains by grafting task *Pre-processing* to task *cvtBGRtoNV12-1*. Such graph transformation is necessary to determine the mapping dataflow tasks and check the schedulability in the next step.

B. STEP 2: MAPPING EXPLORATION

In this step, we determine the mapping of each DL application among Pareto-optimal mapping candidates and the mapping of dataflow tasks onto PEs. The problem is summarized as follows.

- **Input:** All dataflow applications in $\mathcal{A}$ and the set of Pareto-optimal mappings candidates for each deep learning application, $Cand(D_k^i)$ where $D_k^i \in \mathcal{D}$.
- **Constraint:** The WCRT of group G_i should be less than or equal to its deadline, period. i.e. $R(G_i) \leq p^i$ *where* $\forall G_i \in \mathcal{G}$.
- **Objective:** We aim to minimize the maximum utilization to balance the utilization of PEs, $Max_{pe \in \mathcal{PE}}(Util(pe))$.

$$Minimize : Max_{pe \in \mathcal{PE}}(U(pe)) \tag{2}$$

- **Problem:** Find the mapping of tasks $\tau_m^{i,j}$ to PEs, or $\tau_m^{i,j} \rightarrow \mathcal{PE}$, in each dataflow application A_j^i and select the mapping candidate of each deep learning applications. For the selected mapping candidate, we decide the mapping to PEs of each selected mapping candidate: $Cand(D_k^i, x) \rightarrow \mathcal{PE}$.
- **Output:** The mappings of dataflow applications and deep learning applications.

To solve this problem, we propose to use a GA algorithm. Since the execution time of GA increases as the problem size grows, two methods are devised. In the first method, denoted *Heuristic+GA*, we use a heuristic to determine the mapping of dataflow applications, while the GA decides the mapping of DL applications. A solution candidate is represented by a chromosome whose structure is depicted in Figure 6 (b). Each gene indicates the *index* of mapping candidates for each DL application. For example, 4 indicates the 4-th mapping candidate belonging to the second application. It contains as many genes as the number of DL applications. In the second method, denoted *Entire−GA*, we use the GA to select the mapping of dataflow applications simultaneously with the selection of the mapping of DL applications. The corresponding chromosome structure is shown in Figure 6 (c). The chromosome is divided into two parts. The left part indicates the index of mapping candidates for each DL application, same as the first method. The right part indicates the mapping of dataflow tasks. For example, in the figure, the gene of *GPU*

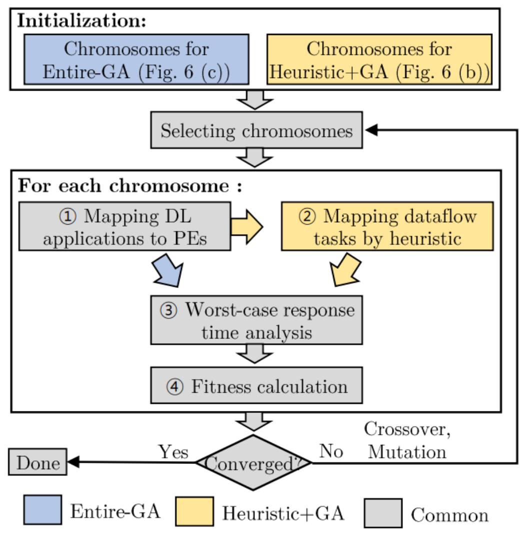

Initialization:
Chromosomes for Entire-GA (Fig. 6 (c))
Chromosomes for Heuristic+GA (Fig. 6 (b))
Selecting chromosomes
For each chromosome :
① Mapping DL applications to PEs
② Mapping dataflow tasks by heuristic
③ Worst-case response time analysis
④ Fitness calculation
Converged?
Yes
No
Done
Crossover, Mutation
Entire-GA
Heuristic+GA
Common

FIGURE 7. **Procedure of the proposed mapping exploration technique.**

Algorithm 1 Pseudo Code of Mapping Heuristic for Dataflow Tasks

```
1:  procedure mapModelTasks(chromosome)
2:      utils = calculateCurrentUtil()
3:      for each task in ordered task list do
4:          minValue = MaxInt
5:          for each mappable processor proc do
6:              minPEUtil, core = getMinUtil(proc, utils)
7:              value = getUtilIfMappedTo(task, proc, core)
8:              if value < minValue then
9:                  minValue, minProc, minCore = value, proc, core
10:         mappingInfo[task] = (minProc, minCore)
11:         updateUtil(minValue, minProc, minCore)
```

means that the corresponding tasks are mapped to the GPU while others are mapped to the specific core of the CPU.

Figure 7 shows the procedure of the proposed mapping exploration technique that consists of four main steps in the evolving process. In the initialization phase, we generate initial chromosomes randomly. Before entering into the evolving process, we first check if the combination of mapping candidates is executable on the target platform. As explained in Section III, the number of DLA-mapped tasks is limited by the SDK.[1] If it violates the constraint, then the chromosome is discarded by setting the fitness value to the maximum value.

In the first step of the iteration, we perform mapping of DL applications to PEs. In this step, we determine which DLA and which CPU core is used for each DL application. After the mapping of DL application is determined, we use a heuristic to determine the mapping of dataflow applications in the *Heuristic+GA* method. This step is skipped in the *Entire−GA* method since the chromosome includes the mapping information of dataflow applications.

After all mappings are decided, we check the schedulability of applications through the worst-case response time analysis and compute the fitness value to select the dominant species in the evolutionary process. With the selected dominant solutions, we perform GA operations such as crossover and mutation, to define the next generation chromosomes. Such an evolutionary process is repeated until no better solution, or chromosome, is found during a given number of iterations.

Since the *Entire−GA* method explores a wider design space than the *Heuristic+GA* method, it is likely to find a better solution, taking much longer time. We improve the convergence speed of the *Entire−GA* method by using the mapping solutions found by the *Heuristic+GA* method as initial chromosomes. Using good initial chromosomes is useful for better exploration [28]. If no mapping is found by the *Heuristic+GA* method due to tight real-time constraint, we loosen the constraint to find solutions to obtain initial chromosomes for the *Entire−GA* method.

[1]In the version we used, the number of DLA-mapped tasks is limited to four.

1) MAPPING DL APPLICATIONS TO PEs

The mapping option in Table 2 indicates the processor type, not a specific processing element. For instance, we may use any DLA between two DLAs in the system when mapping option A or B is taken. Similarly, the pre-/post-processing tasks of a DL application can be mapped to any core in a multi-core CPU. To evaluate the fitness value of a chromosome, we need to determine which PE to use for each DL application. Since our objective is to balance the processor utilization, we use a simple scheme as follows: First, we sort the mapped tasks on each processor in the decreasing order of profiled execution time. Next, we perform a greedy mapping of tasks to PEs in a round-robin fashion starting from the longest task. For example, if there are three tasks mapped onto DLA, they are mapped to DLA0, DLA1, and DLA0 in the order of the execution length if there are two DLAs.

2) MAPPING DATAFLOW TASKS BY A SIMPLE HEURISTIC

Algorithm 1 displays the pseudo-code of the proposed mapping heuristic for dataflow tasks in the *Heuristic+GA* scheme. This heuristic is called for each chromosome that indicates a candidate mapping combination of all DL applications. In other words, we determine the mapping of dataflow tasks after the mapping of pipeline stages of all DL applications onto processing elements is completed. Since mapping is performed for each chromosome during the evolution process, we use a simple greedy heuristic, sacrificing performance for faster execution speed. We first compute the PE utilization based on the mapping result of DL applications (line 2). Next, we sort the dataflow tasks in the decreasing order of the maximum WCET value among the mappable processors and determine the mapping in the sorted order (line 3). We find the PE with the minimum utilization for each mappable processor (lines 5-6). And we compute the maximum utilization among all PEs in the selected processor when the current task is mapped to the corresponding PE (*getUtilIfMappedTo* function in line 7). To minimize the maximum utilization, we select the PE with the minimum value and map the task to the PE (lines 8-10). Afterwards, the PE utilization is updated according to the mapping (line 11).

The time complexity of Algorithm 1 is $O(|V_j| \cdot |\mathcal{PE}|)$ where $|V_j|$ indicates the number of tasks for all dataflow applications and $|\mathcal{PE}|$ is the number of PEs. This is because it checks the utilization of all PEs to find the PE with the lowest utilization for each task to map. The space complexity is $O(|V_j| + |\mathcal{PE}|)$ since the space to store the mapping information of tasks and PE utilizations depends on the number of tasks and PEs.

3) WORST-CASE RESPONSE TIME ANALYSIS

For each group, the worst-case response time (WCRT) analysis is conducted and the schedulability is checked by comparing the deadline constraint and the estimated WCRT. We use a compositional performance analysis (CPA) to estimate the WCRT for each group ([29], [30]). In the CPA, the WCRT analysis is performed for each PE separately and the dependency between tasks mapped to different PEs is modeled as an event stream that is specified by a tuple (period, jitter, the minimum distance between two events). Starting from the PE that the source task in a group is mapped to, WCRT analysis is performed one PE at a time up to the PE where the last task in the group is mapped to, following the task dependency. In case there exist multiple dependency paths in a group, we choose the maximum WCRT of all paths as the WCRT of the group.

Since the scheduling policy of processors is not identical, we apply a different WCRT analysis method for each processor type. For CPU that uses a fixed-priority preemptive scheduling [31] scheme, we use a well-known response time analysis formulated as follows:

$$r^{m+1} = C(\tau_y^{i,x}) + \sum_{\tau_h \in hp(\tau_y^{i,x})} \left\lceil \frac{r^m + J_{\tau_h}}{P(\tau_h)} \right\rceil \cdot C(\tau_h)$$
$$where\ r^0 = C_{\tau_y^{i,x}} \tag{3}$$

Equation (3) estimates the WCRT of a task $\tau_y^{i,x}$ which is mapped to the CPU. Task set $hp(\tau_y^{i,x})$ is a set of higher or equal priority tasks that are mapped to the same PE with task $\tau_y^{i,x}$. J_{τ_h} is the jitter of task τ_h. The estimated WCRT of CPU task $\tau_y^{i,x}$ becomes the converged value of r^m.

In the GPU, the mapped tasks are executed in a FIFO order without preemption [32]. The WCRT of task $\tau_y^{i,x}$ mapped to the GPU can be computed by the following equation.

$$r = C(\tau_y^{i,x}) + \sum_{\tau_e \in ep(\tau_y^{i,x})} B_{\tau_e}^{\tau_y^{i,x}} \tag{4}$$

Task set $ep(\tau_y^{i,x})$ indicates a set of tasks that are mapped to the same PE with $\tau_y^{i,x}$. $B_{\tau_e}^{\tau_y^{i,x}}$ represents the maximum interference from task $\tau_e \in ep(\tau_y^{i,x})$ to the target task $\tau_y^{i,x}$. If τ_e is a dataflow task, the number of interference is bounded by $\lceil \frac{P(\tau_y^{i,x})}{P(\tau_e)} \rceil + 1$. If it is a pipeline stage of another DL application, we need to consider the number of sub-tasks in the task. Suppose task $\tau_y^{i,x}$ and τ_e are both pipeline stages that have five and two sub-tasks, respectively. Sub-tasks in $\tau_y^{i,x}$ can be interfered at most five times since sub-tasks are scheduled in the FIFO order. On the other hand, task τ_e can interfere with task $\tau_y^{i,x}$ at most $\lceil \frac{P(\tau_y^{i,x})}{P(\tau_e)} \rceil + 1$ times. If the task τ_e can interfere with at most twice, four sub-tasks ($min(5, 2 \cdot 2)$) in τ_e may interfere with task $\tau_y^{i,x}$.

TABLE 3. Benchmark networks and the number of mapping candidates obtained by step 1.

Subgraph	Network	# of layers	# of selected candidates
$Det1$	Yolov4 [34]	269	51
$Det2$	Yolov2tiny [26]	24	14
$Det3$	Yolvo3tiny [35]	35	12
$Det4$	Yolov4csp [36]	290	65

TABLE 4. Mappable processors of tasks in dataflow applications.

Tasks	Processor	Tasks	Processor
ReadImage	CPU	cvt_NV12_BGR-1	CPU,GPU
cvt_BGR_NV12	CPU,GPU	Rescale-2	GPU
Rescale-1	GPU	cvt_NV12_BGR-2	CPU,GPU
Bilateral	CPU,GPU	SaveImage	CPU

Equation (4) is also applied to DLA since the same non-preemptive scheduling policy is used as the GPU. We check whether the estimated WCRT violates the given deadline constraint. If the estimated WCRT is bigger than the deadline, the mapping does not satisfy the schedulability constraint.

4) FITNESS CALCULATION

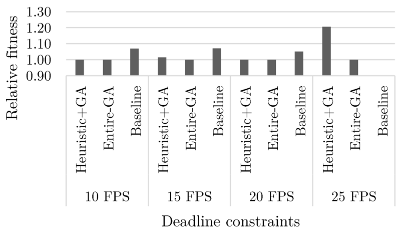
As the objective function of GA in the second step, we aim to minimize the maximum PE utilization in order to balance the utilization of PEs, which is also taken in the work of [20]. With this objective in mind, we define two types of fitness values as described in Equation (2). One is the maximum utilization of each processing element ($Max_{pe \in \mathcal{PE}}(U(pe))$)) and the other is the maximum WCRT value of each group ($Max_{G_i \in \mathcal{G}}(R(G_i))$). Since the implicit deadline constraint should be satisfied for each group, by setting the latter fitness value, dominant solutions at each generation are more likely to satisfy the deadline constraint. If the estimated WCRT does not satisfy the deadline constraint, the solution is not feasible. Based on the fitness values of each candidate solution, we select the solution with the minimum value of the maximum utilization.

$$Minimize : (\, Max_{pe \in \mathcal{PE}}(U(pe)), Max_{G_i \in \mathcal{G}}(R(G_i)) \,) \tag{5}$$

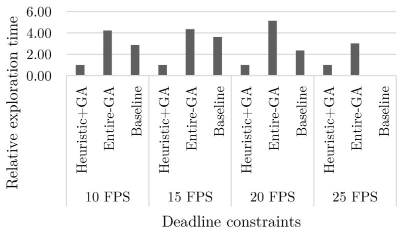
V. EXPERIMENTS

A. COMPARISON WITH A PREVIOUS WORK

As a preliminary experiment, we evaluate the approach taken by previous works, which is to translate a DL application to a dataflow model. The work of [7] provides an example in which a Resnet152 network [33] is specified by an extended SDF model, called SDF/L. We implement the same network



(a) Fitness



(b) Exploration time

FIGURE 8. Comparison of three methods for the motivational example: Heuristic+GA, Entire−GA, Baseline.

using TensorRT and compare both implementations on the same target board. The pre-/post-processing parts are mapped to the CPU, and other parts are mapped to GPU in both methods. It is observed that the previous approach could achieve only 18 FPS (frame per second) performance while the version of TensorRT achieves 81 FPS performance. It confirms that transforming a DL application to a dataflow may suffer from significant performance loss if the same degree of optimizations is not applied as TensorRT. In addition, model conversion takes a huge amount of effort. Since the previous works do not target the system with NPUs, they do not consider the challenges posed by the NPU and its SDK described in Section III-A.

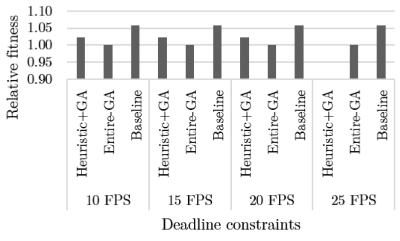
B. SET-UP

Since there is no previous work that tackles the same mapping problem, we devise a GA-based scheme by merging two recent previous works [14] and [25] and take it as a baseline technique to compare, denoted by *Baseline* in the experimental results. The former work is used to map multiple DL applications on the target board first, and the latter is used to map dataflow tasks after DL application mapping is completed. In contrast, two proposed methods, *Heuristic+GA* and *Entire−GA* perform the mapping of DL applications and dataflow applications simultaneously.

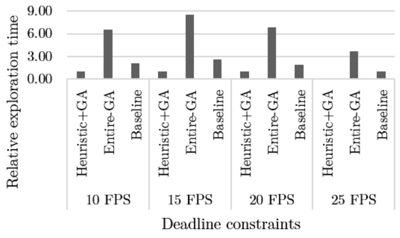
We implement the GA meta-heuristic with the DEAP library [37]. The GA runs on a host computer consisting of AMD Ryzen 9 3950X Processor. Experiments are made with the aforementioned motivational example in Figure 2 and randomly generated graphs. We conduct each experiment three times and get the average value to measure the mapping exploration time and the best fitness value. The target system is the Xavier board with Jetpack 4.6 and TensorRT 8.0.1. We reduce the available number of CPU cores to four to observe the effect of resource contention between tasks on the CPU.

C. EXPERIMENTAL RESULTS: MOTIVATIONAL EXAMPLE

We profile the tasks with TensorRT IProfiler and POSIX time library. We set the WCET of the task as the value obtained by adding six times the standard deviation to the mean of the profiled execution times. We deploy four different DL networks, which are provided in [38], as described in Table 3, each of which corresponds to a *Det* in Figure 2. For example, *Det4*, which is connected to *Image processing4*, is a *Yolov4csp* network in Figure 2. Also, we set the priorities of

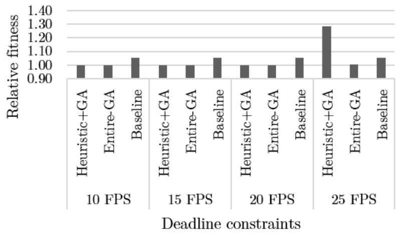


(a) Fitness

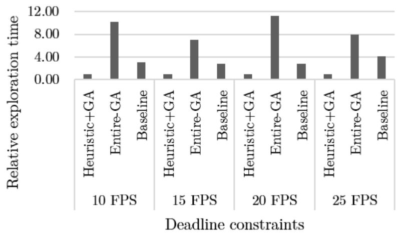


(b) Exploration time

FIGURE 9. Comparison of three methods with four randomly generated dataflow applications.

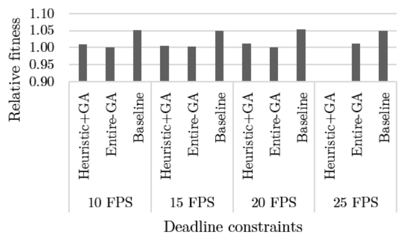


(a) Fitness

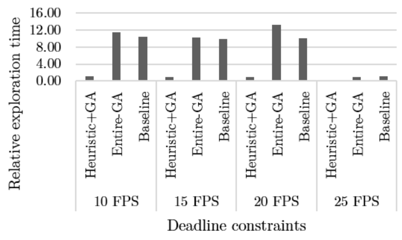


(b) Exploration time

FIGURE 10. Comparison of three methods with eight randomly generated dataflow applications.



(a) Fitness



(b) Exploration time

FIGURE 11. Comparison of three methods with sixteen randomly generated dataflow applications.

application groups in the following order: G_1(Image processing1 + Det1), G_4, G_2, G_3. The mappable processors of each task are described in Table 4.

1) FINDING MAPPING CANDIDATES

The number of Pareto-optimal mappings selected from step 1 for each DL application is shown in Table 3. In this step, we run the GA with 65 chromosomes for 1000 iterations. Multiplying those numbers for all DL applications defines the design space size for selecting the mapping candidates in step 2.

2) DESIGN SPACE EXPLORATION

We vary the deadline constraint of all groups from 10 FPS to 25 FPS, assuming implicit deadlines.[2] In this experiment, we run the GA at most 100 iterations with 2048 chromosomes by making the GA terminate if there is no better solution found in 10 iterations.

Figure 8 shows the relative fitness (maximum utilization) and exploration time over the minimum value in each bundle. The lower value is better in the figure. For all deadline constraints, the *Entire−GA* method shows the best results as shown in Figure 8 (a). Up to a deadline constraint of 20 FPS, the *Heuristic+GA* and *Entire−GA* methods give a similar result. But with the deadline constraint of 25 FPS, *Entire−GA* gives better results by a large margin. This is because the size of the design space for *Heuristic+GA* method is too narrow to find good solutions. The *Baseline* method could not find a feasible solution at a tight deadline constraint of 25 FPS. Also, at lower deadline constraints, it shows 5%∼7% worse fitness than *Entire−GA* method. As for exploration time, the *Heuristic+GA* method took the least amount of time to perform, as depicted in Figure 8 (b) since it explores a smaller design space than the other methods. The Entire−GA method takes the longest time as expected.

3) CODE GENERATION

To check the viability of the proposed methodology, we synthesize the target codes with a model-based design framework [7] and run them on the target hardware platform. The sample of the motivational example is available on a GitHub.[3]

D. EXPERIMENTAL RESULTS: RANDOMLY GENERATED DATAFLOW GRAPHS

In this experiment, we randomly generate dataflow graphs by the SDF3 [39] while using the same four DL applications as the motivational example. We assume that all applications are independent. Each dataflow graph consists of 10 nodes. We vary the number of dataflow graphs and the average WCET of the dataflow tasks. The average WCET of dataflow tasks is inversely proportional to the number of graphs to make the total workload of dataflow applications remain similar. It means that the average WCET of dataflow tasks is four times larger with 4 dataflow graphs than that with 16 dataflow graphs. In the case that the number of graphs is 16, the WCET of a dataflow task is randomly set in the range of 10∼100us on the GPU, and 100∼500us on the CPU. And we set a half of dataflow tasks can be run on GPU only. We make the average WCET of CPU-mapped tasks be about five times of that of GPU-mapped tasks.

Figures 9-11 show the results of the experiment when the number of graphs is 4, 8, and 16, respectively. As expected, the *Entire−GA* method shows the best fitness. The fitness gap between the *Entire−GA* method and the *Heuristic+GA* method tends to increase as the deadline constraint is tightened. In some cases, the *Heuristic+GA* method could not find a solution even when the *Baseline* method found a solution. But in most cases when the *Heuristic+GA* method finds a solution, it finds a better solution than the *Baseline* method. Hence these two methods are not dominating each other in terms of fitness. When no solution could be found by the *Heuristic+GA* method when the deadline constraint is 25 FPS, we loosened the constraint to 10 FPS and found the solution that is used as the initial chromosome in the *Entire−GA* method.

As for the exploration time, the *Heuristic+GA* method takes the least time by more than three times than the *Entire−GA* method. This experiment clearly demonstrates the trade-off between two methods in terms of fitness and exploration time. And it also shows that the proposed two methods are significantly better than the previous state-of-the-art method which is the *Baseline* method.

[2]Implicit deadline means the period is equal to the relative deadline. For example, 10 FPS means a deadline of 100 milliseconds.

[3]https://github.com/cap-lab/HOPES/tree/master/HOPES_UI/schematics/Test_Examples/ExternalTask/ImgProcessing.files

VI. CONCLUSION

In this paper, we propose a novel technique to support deep learning applications in a model-based embedded software design methodology leveraging the optimization capability of a deep learning SDK. We first find a set of Pareto-optimal mapping candidates for each deep learning application, independently of the model-based design flow. Adding the obtained mapping solution to the model-based design framework, we explore the mapping of dataflow applications and the mapping candidates of deep learning applications together with the meta-heuristic. The viability and efficacy of the technique are confirmed by experiments with a non-trivial real-life example and randomly generated graphs. We could reduce at least 5% of the maximum utilization over the previous state-of-the-art method that separates the mapping of DL applications and dataflow applications. More importantly, the proposed technique could find a solution when the previous method fails to find one. Even though the current implementation is made for a specific target hardware platform and its associated SDK, the proposed methodology can be applied to other hardware-software platforms, we believe.

REFERENCES

[1] D. Harel and M. Politi, *Modeling Reactive Systems With Statecharts: The STATEMATE Approach*. New York, NY, USA: McGraw-Hill, 1998.

[2] R. Devaraj, A. Sarkar, and S. Biswas, "Supervisory control approach and its symbolic computation for power-aware RT scheduling," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 787–799, Feb. 2019.

[3] J. Castrillon, R. Leupers, and G. Ascheid, "MAPS: Mapping concurrent dataflow applications to heterogeneous MPSoCs," *IEEE Trans. Ind. Informat.*, vol. 9, no. 1, pp. 527–545, Feb. 2013.

[4] M. Pelcat, K. Desnos, J. Heulot, C. Guy, J. Nezan, and S. Aridhi, "Preesm: A dataflow-based rapid prototyping framework for simplifying multicore DSP programming," in *Proc. 6th Eur. Embedded Design Educ. Res. Conf. (EDERC)*, Sep. 2014, pp. 36–40.

[5] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt, "Ptolemy: A framework for simulating and prototyping heterogeneous systems," in *Proc. Readings Hardw./Softw. Co-Design*, 2001, pp. 527–543.

[6] C. Ptolemaeus, *System Design, Modeling, and Simulation: Using Ptolemy II*, vol. 1. Berkeley, CA, USA: Ptolemy.org, 2014.

[7] E. Jeong, D. Jeong, and S. Ha, "Dataflow model–based software synthesis framework for parallel and distributed embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 5, pp. 1–38, Sep. 2021.

[8] R. Devaraj and A. Sarkar, "Resource-optimal fault-tolerant scheduler design for task graphs using supervisory control," *IEEE Trans. Ind. Informat.*, vol. 17, no. 11, pp. 7325–7337, Nov. 2021.

[9] S. Ha and H. Oh, "Decidable dataflow models for signal processing: Synchronous dataflow and its extensions," in *Handbook of Signal Processing Systems*, 2013, pp. 1083–1109.

[10] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.

[11] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete, "Cycle-static dataflow," *IEEE Trans. Signal Process.*, vol. 44, no. 2, pp. 397–408, Feb. 1996.

[12] H. Hong, H. Oh, and S. Ha, "Hierarchical dataflow modeling of iterative applications," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[13] S. Minakova, E. Tang, and T. Stefanov, "Combining task- and data-level parallelism for high-throughput CNN inference on embedded CPUs-GPUs MPSoCs," in *Proc. SAMOS*. Cham, Switzerland: Springer, 2020, pp. 18–35.

[14] J. Kim and S. Ha, "Energy-aware scenario-based mapping of deep learning applications onto heterogeneous processors under real-time constraints," *IEEE Trans. Comput.*, vol. 72, no. 6, pp. 1666–1680, Jun. 2023.

[15] M. R. Garey and D. S. Johnson, "A guide to the theory of NP-completeness," *Comput. Intractability*, pp. 37–79, Nov. 1990.

[16] H. Oh and S. Ha, "A static scheduling heuristic for heterogeneous processors," in *Proc. Euro-Par*. Berlin, Germany: Springer, 1996, pp. 573–577.

[17] 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.

[18] E. S. H. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 2, pp. 113–120, Feb. 1994.

[19] A. K. Singh, P. Dziurzanski, H. R. Mendis, and L. S. Indrusiak, "A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems," *ACM Comput. Surv.*, vol. 50, no. 2, pp. 1–40, Mar. 2018.

[20] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele, "Scenario-based design flow for mapping streaming applications onto on-chip many-core systems," in *Proc. Int. Conf. Compil., Archit. Synth. Embedded Syst.*, Oct. 2012, pp. 71–80.

[21] S. Kang, D. Kang, H. Yang, and S. Ha, "Real-time co-scheduling of multiple dataflow graphs on multi-processor systems," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[22] Y. Xiang and H. Kim, "Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2019, pp. 392–405.

[23] R. Pujol, H. Tabani, L. Kosmidis, E. Mezzetti, J. A. Ferrer, and F. J. Cazorla, "Generating and exploiting deep learning variants to increase heterogeneous resource utilization in the NVIDIA xavier," in *Proc. ECRTS*, 2019.

[24] D. Kang, J. Oh, J. Choi, Y. Yi, and S. Ha, "Scheduling of deep learning applications onto heterogeneous processors in an embedded device," *IEEE Access*, vol. 8, pp. 43980–43991, 2020.

[25] D. Jeong, J. Kim, M. Oldja, and S. Ha, "Parallel scheduling of multiple SDF graphs onto heterogeneous processors," *IEEE Access*, vol. 9, pp. 20493–20507, 2021.

[26] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 6517–6525.

[27] (2022). *NVIDIA TensorRT*. Accessed: Mar. 23, 2023. [Online]. Available: https://developer.nvidia.com/blog/tensorrt-3-faster-tensorflow-inference/

[28] B. Kazimipour, X. Li, and A. K. Qin, "A review of population initialization techniques for evolutionary algorithms," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2014, pp. 2585–2592.

[29] J. C. P. Gutierrez, J. J. G. Garcia, and M. G. Harbour, "On the schedulability analysis for distributed hard real-time systems," in *Proc. 9th Euromicro Workshop Real Time Syst.*, 1997, pp. 136–143.

[30] M. Jersak, "Compositional performance analysis for complex embedded applications," Ph.D. dissertation, Dept. Elect. Eng. Inf. Technol., Braunschweig, Techn. Univ., Brunswick, Germany, 2005.

[31] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proc. 11th Real-Time Syst. Symp.*, 1990, pp. 201–209.

[32] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "GPU scheduling on the NVIDIA TX2: Hidden details revealed," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2017, pp. 104–115.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[34] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, *arXiv:2004.10934*.

[35] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.

[36] C. Wang, A. Bochkovskiy, and H. M. Liao, "Scaled-YOLOv4: Scaling cross stage partial network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 13024–13033.

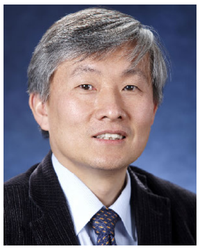
[37] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2171–2175, 2012.

[38] E. Jeong, J. Kim, and S. Ha, "TensorRT-based framework and optimization methodology for deep learning inference on Jetson boards," *ACM Trans. Embedded Comput. Syst.*, vol. 21, no. 5, pp. 1–26, Sep. 2022.

[39] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF for free," in *Proc. ACSD*, 2006, pp. 276–278.

JANGRYUL KIM (Graduate Student Member, IEEE) received the B.S. degree in computer science and engineering from Sogang University, Seoul, South Korea, in 2017. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Seoul National University, Seoul. His current research interests include system performance estimation and the design space exploration of embedded systems.

JAEWOO SON (Graduate Student Member, IEEE) received the B.S. degree in information systems from Hanyang University, Seoul, South Korea, in 2021. He is currently pursuing the Ph.D. degree in computer science and engineering with Seoul National University, Seoul. His current research interests include the HW/SW codesign of embedded systems and system simulation.



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 the HW/SW codesign of embedded systems, embedded machine learning, and the Internet of Things. He has actively participated in the premier international conferences in the EDA area, for instance serving ESWEEK, in 2018, as the General Chair. He is a member of ACM.