IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 13, 2021, accepted January 20, 2021, date of publication January 26, 2021, date of current version February 4, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3054725

Parallel Scheduling of Multiple SDF Graphs Onto Heterogeneous Processors

DOWHAN JEONG[1], JANGRYUL KIM[2], MARI-LIIS OLDJA[2], AND SOONHOI HA[2], (Fellow, IEEE)

[1]TmaxA&C, Seongnam 13613, South Korea
[2]Department of Computer Engineering, Seoul National University, Seoul 08826, South Korea

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

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] under Grant NRF-2019R1A2B5B02069406.

ABSTRACT Parallel scheduling of multiple real-time applications onto heterogeneous processors is needed in the emerging embedded systems such as self-driving cars, smart cameras, and smartphones. Assuming that an embedded application is specified as a synchronous dataflow (SDF) graph or its extension, we propose a novel parallel scheduling methodology based on an evolutionary algorithm where the mapping of tasks onto processors is evolved to optimize a given objective function in an iterative fashion. In each iteration, we use an existing worst-case response time (WCRT) analysis tool to check if all applications satisfy their real-time requirements by translating each SDF graph into a directed acyclic graph (DAG) that is assumed in the WCRT analysis tool. Since the WCRT analysis must be performed in each iteration of evolution, we propose a clustering technique to reduce drastically the analysis time that depends on the number of nodes and their dependency. We formally prove that the proposed clustering technique does not change the estimated WCRT of each application. The effectiveness of the proposed scheduling methodology with the clustering technique is verified with extensive experiments using real-life benchmarks, randomly generated graphs, and the comparison with the existing technique.

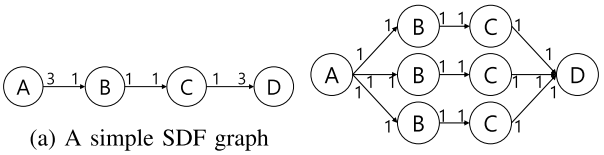
INDEX TERMS Mapping and scheduling, design space exploration, dataflow model, model transformation, performance analysis, worst-case response time.

I. INTRODUCTION

To cope with the increasing user demand for compute-intensive deep learning applications, embedded systems tend to equip heterogeneous processing elements (PEs) that include a multi-core CPU, a GPU, and/or a deep learning accelerator called a Neural Processing Unit (NPU). In such a system, we may need to run multiple applications concurrently sharing the PEs. Since an embedded application usually has a real-time performance requirement such as throughput and response time, parallel scheduling of multiple real-time applications onto shared heterogeneous PEs emerges as an essential but challenging problem.

We assume that an embedded application is specified as a synchronous dataflow (SDF) [2] graph or its extension. The SDF model is popularly used for signal processing or compute-intensive applications where an application is specified as a task graph: a node represents a computation task, and a directed edge represents the data dependency between two end nodes, or tasks, with a FIFO channel. Since the number of data samples consumed from each input port or produced to each output port per task execution is fixed in the SDF model, we can construct an execution schedule of tasks statically. The number of samples consumed or produced is called the *sample rate* of the associated port. Fig. 1 (a) shows a simple SDF graph, annotating the sample rate of each port on the edges. Since task A produces three samples per execution and task B consumes one sample per execution, task B should be executed three times more than task A on average in order to avoid buffer overflow of the channel between them if the graph is executed repeatedly. A single execution of an SDF graph consists of the minimum number of executions of tasks while satisfying the relative execution rates. The minimum number is called the *repetition count* of each task, denoted as a $rep(X)$ for task X. The repetition counts of four tasks in Fig. 1 (a) become as follows: $rep(A) = rep(D) = 1$,

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



(a) A simple SDF graph

(b) The equivalent HSDF graph to graph (a)

FIGURE 1. An example graph of SDF model.

$rep(B) = rep(C) = 3$. A single execution of the SDF graph includes three task instances, or jobs, of tasks B and C each.[1]

Since the SDF graph describes only the data dependency between tasks, it can be easily parallelized by mapping task instances to PEs. While the parallel scheduling of a single SDF graph onto heterogeneous PEs has been extensively researched ([3]–[6]), there exist only a few studies that consider the scheduling of multiple SDF graphs on the shared PEs ([7]–[9]). One approach to schedule multiple SDF graphs is to merge all SDF graphs into a single SDF graph after expanding each of them to the hyper-period and schedule the merged graph at once [7]. This approach is not applicable when the starting offsets of SDF graphs are not fixed. The authors in [8] proposed a two-step approach. After they find a set of static schedules onto PEs for each SDF graph, varying the number of processors, they use a meta-heuristic to find an optimal combination of per-graph schedules to minimize the resource requirement by processor sharing. They assume that the processors are homogeneous. This approach is extended in [9] for the scheduling methodology of deep learning applications. They extended the analysis technique to non-preemptive scheduling onto heterogeneous processors.

In this work, we propose a novel parallel scheduling methodology based on an evolutionary algorithm where the mapping of tasks onto processors is evolved to optimize a given objective function in an iterative fashion. In each iteration, it is necessary to check if all applications satisfy their real-time requirement and to evaluate the objective function for given mapping and scheduling information of tasks. When the repetition periods of SDF graphs are different and the starting offsets are not fixed, it is very challenging to analyze the real-time performance of each SDF graph considering all possible interference among applications. One solution is to decompose each SDF graph into a set of independent real-time tasks that have unique deadlines and offsets ([10], [11]). In this approach, the deadline and the starting offset of each real-time task should be determined carefully, keeping the dependency between tasks, which is likely to produce a too pessimistic estimation of the real-time performance. In the proposed technique, we use a worst-case response time (WCRT) analysis tool. There exist several WCRT analysis tools available ([12]–[15]), all assuming that each application is represented as a directed acyclic task graph. While the true WCRT cannot be found analytically, they aim to estimate the upper bound as tightly as possible by considering all possible interference between tasks conservatively.

To use an existing WCRT analysis tool, we convert each SDF graph into a homogeneous synchronous dataflow (HSDF) graph that becomes a directed acyclic graph (DAG) if a cycle is broken by removing the feedback edge with an initial sample. Fig.1 (b) shows the equivalent HSDF graph converted from the SDF graph of Fig.1 (a). Note that a node in the converted DAG corresponds to a job or a task instance of an associated task: an SDF task generates as many nodes as its repetition count. Since mapping decision is assumed to made at the task level, all jobs from a single SDF task are mapped to the same PE. In case the granularity of an SDF task is large, the task is likely to have internal states of being maintained consistently over jobs. If there is no internal state in an SDF task, the task is explicitly specified as *parallelizable*, and we can map the task onto multiple homogeneous PEs.

Since the WCRT analysis must be performed in each iteration of evolution, the analysis speed is an important factor to consider. The time complexity of a WCRT analysis tool depends on the number of nodes and their dependency. Table. 1 shows the analysis time of several WCRT analysis tools, depending on the number of nodes. In this preliminary experiment, we assume that five applications that have the same number of nodes are running on four processing elements, two preemptive and two non-preemptive. The dependency of nodes in each application is randomly constructed except for MAST (Modeling and Analysis Suite for Real-Time Applications) [16] that accepts the chained topology only. The execution time of each node is selected randomly between [500, 1000]. The period and deadline of applications are set to a large value so that the analysis tool produces the WCRT of each application, assuming that interference from the other applications occurs once from each application. We repeat each experiment 1000 times and take the average value. It is observed that the analysis time increases drastically as the number of nodes in a DAG increases.

TABLE 1. The average analysis times of three WCRT analysis tools with varying number of nodes(unit: ms).

WCRT Analysis Tool	Total Number of Nodes					
	10	50	100	200	500	1000
MAST [16]	15.8	75.8	358.6	3873.9	>10s	>10s
pyCPA [17]	6.5	73.6	389.3	3078.5	>10s	>10s
HPA [15]	4.8	22.3	51.5	138.9	554.8	2439.8

To reduce the WCRT analysis time, we propose a novel node clustering technique for the converted DAG. The key constraint for node clustering is not to change the real-time performance by considering all possible interference scenarios between applications for given mapping and scheduling information of applications. This constraint makes the proposed clustering technique distinguished from existent SDF clustering techniques ([18], [19]) that do not consider mapping and scheduling. If those clustering techniques are

[1] A task instance and a *job* are used interchangeably in this paper

applied beforehand at the task level, the design space for mapping will be reduced drastically. On the contrary, the proposed clustering is performed at the job level. We formally prove that the proposed technique does not change the real-time performance that is estimated by a chosen WCRT analysis tool.

As for applications, we consider three types of real-time characteristics: periodic, sporadic, and best-effort. We assume that a periodic application has an implicit deadline so that the worst-case response time may not be greater than the period. A sporadic application has a deadline constraint to be met, while a best-effort application has a lower priority than the other types of applications with the objective of throughput maximization. Note that such diverse real-time characteristics can be examined by WCRT analysis. Since we are likely to have different types of applications running concurrently in practice, supporting a heterogeneous combination of real-time requirements is another benefit of using the WCRT analysis tool.

The need and goodness of the proposed technique are confirmed with extensive experiments using real-life benchmarks and randomly generated graphs. We first illustrate how the proposed methodology explores the design space of parallel scheduling of multiple applications. Next, we demonstrate that the proposed clustering technique can reduce the graph size and the WCRT analysis time significantly.

Four main contributions of this work can be summarized as follows.

- A novel parallel scheduling technique based on an evolutionary algorithm is proposed to schedule multiple SDF graphs with diverse real-time characteristics onto heterogeneous PEs. For the performance evaluation of each mapping candidate, it uses an existing WCRT analysis tool.
- To reduce the execution time of WCRT analysis, a node clustering technique is devised. With five real-life benchmark applications running concurrently, the WCRT analysis time is reduced to 1.15% (87X speed-up).
- We formally prove that the proposed clustering technique does not change the real-time performance that is estimated by the WCRT analysis tool.
- The proposed scheduling technique is extended to support multi-mode SDF graphs in which each mode of operation is specified by an SDF graph.

The rest of the paper is organized as follows. In the next section, we review the related work on mapping and scheduling of SDF graphs, WCRT estimation, and the SDF graph reduction. After stating the system model and the mapping and scheduling model in a formal way in section III, we present the proposed scheduling technique based on a genetic algorithm in section IV. Section V explains the proposed clustering technique with formal proof that it does not change the performance estimation result. Section VI explains how the proposed performance estimation technique can be applied to a multi-mode SDF model. After experimental results are shown and discussed in section VII, concluding remarks are given in section VIII.

II. RELATED WORK

In this section, we review the related work in the following three techniques involved in the proposed methodology: mapping and scheduling of SDF graphs, the WCRT estimation, and SDF graph reduction.

A. SCHEDULING OF SDF GRAPHS

SDF graph scheduling problems can be divided into two categories: scheduling of a single SDF graph and scheduling of multiple SDF graphs. Since the parallel scheduling problem of a DAG on multiple processors is known to be NP-hard [20], numerous techniques have been proposed to obtain a sub-optimal solution. Since they are applicable for the scheduling of an SDF graph after the SDF graph is translated into a DAG, we first review the recent DAG scheduling techniques onto heterogeneous processor systems. And we review the scheduling approaches of multiple SDF graphs next.

1) SCHEDULING OF A SINGLE SDF GRAPH

Earlier work on parallel scheduling of a DAG onto a heterogeneous multi-processor system proposed a list scheduling heuristic [21], called Best-Imaginary-Level (BIL) scheduling, where a task is assigned a set of priorities considering the different mapping possibilities of the precedent tasks for each processor candidate. Even though it is a heuristic, it is proven that it produces an optimal scheduling result if the graph has a linear topology. A similar list-scheduling heuristic, called Heterogeneous Earliest-Finish-Time (HEFT), was proposed [22] where HEFT corresponds to BIL [21]. Even though they are fast and produce sub-optimal solutions for a given problem setting, it is not extensible to consider design constraints and architecture characteristics.

To overcome the drawbacks of heuristics, scheduling techniques based on a meta-heuristic algorithm have been proposed. An earlier work [23] uses a genetic algorithm to solve the scheduling problem onto multiple processors. The authors of [24] proposed to use a QEA (quantum-inspired evolutionary algorithm) to schedule an SDF graph considering pipelining as well as task-level parallelism with the objective of throughput maximization. Many subsequent studies used the meta-heuristic-based approaches since there exist solvers available to the public and the problem formulation is extensible to consider multiple objectives and various design constraints. In case the design space of mapping can be limited, ILP (Integer Linear Programming)-based approaches can be used to obtain the optimal scheduling solutions as proposed in [25] and [26]. The former aims to minimize the number of used processors, while the latter aims to minimize the execution latency while keeping the deadline constraints. The latter is extended to consider heterogeneous processors and shared bus contention in [27].

2) PARALLEL SCHEDULING OF MULTIPLE SDF GRAPHS

In spite of a long period of research, mapping and scheduling of multiple task graphs onto heterogeneous PEs remains a challenging problem. Most of the existent approaches assume that the task graphs are all DAGs and executed only once so that the scheduling objective is to minimize the schedule length of applications. As an example, a scheduling heuristic is proposed in [28] by extending the HEFT algorithm [22] to multiple DAGs.

There exist only a few techniques proposed to schedule periodic multiple SDF graphs onto multiple PEs, to the best of our knowledge. The difficulty lies in the schedulability analysis for each mapping candidate. To ease the schedulability analysis, we may decompose an SDF graph into a set of independent real-time tasks first ([10], [11]), and map those generated tasks onto PEs using any real-time scheduling technique of independent tasks. Since this decomposition may give a loose estimation of the real-time performance, the authors in [8] proposed a two-step approach. In the first step, they find a set of static schedules onto PEs for each SDF graph using a genetic algorithm. In the second step, they apply a genetic algorithm again to find a sub-optimal combination of per-graph schedules to minimize the resource requirement. For checking the schedulability constraints, they introduce the notion of *mobility* that indicates how much interference a schedule can tolerate, and they allow the merging of two schedules only when the amount of co-mapped tasks from the other task graph is smaller than the mobility on each processing element.

On the contrary to this two-stage technique, our proposed technique applies a meta-heuristic to determine the mapping of all SDF graphs at once and uses the WCRT analysis tool for real-time performance analysis. The authors of [9] have proposed a parallel scheduling technique of deep learning applications, adopting the worst-case response time analysis method used in [8]. They extended the analysis technique to support the non-preemptive scheduling policy and heterogeneous processing elements. Their proposed technique is based on a meta-heuristic to find a near-optimal mapping of layers in deep learning applications. However, it assumes that the applications are all HSDF while our proposed technique supports general SDF graphs.

B. WCRT ESTIMATION

The worst-case response time of an application that is specified by a directed task graph is defined as the longest time difference between the completion and the release time of the application. When multiple applications are running on a distributed embedded system and task execution times are varying, finding the WCRT of an application is extremely difficult so that various approaches have been proposed to make an estimation as tightly as possible by conservatively modeling the inter-task interference. They commonly assume that an application is specified by a DAG.

Since we make an implicit deadline assumption, the throughput performance of a periodic application is satisfied if the WCRT of the application is no greater than the period. The latency constraint is directly confirmed with the WCRT. The performance of a best-effort application is maximized when the WCRT is minimized in our model. Thus the real-time performance of all applications can be analyzed by WCRT analysis. In the proposed parallel scheduling methodology, we may use any WCRT analysis tool. Now we review the WCRT analysis tools available in this section.

Early researches extended the conventional response time analysis (RTA) to the system level. While RTA assumes independent tasks, those extensions model the task dependency with the starting offsets of tasks. In case the real system fits for the assumed model, such a holistic approach could make a tight estimation. A representative work is MAST that is an open-source set of tools that enables modeling real-time applications and performing timing analysis of those applications [16]. On the other hand, a compositional approach is proposed in [12], where the conventional RTA is carried out for each processing element (PE). The interface between PEs is defined by an event stream that is modeled as a 3-tuple (p, j, d) where p, j, and d represent the period, jitter, and the minimum distance between two events, respectively. The WCRT of a task graph can be estimated as a sum of WCRTs on each PE where the task graph is mapped onto. There is an open-source tool called pyCPA (Python Implementation of Compositional Performance Analysis) [17], based on this approach. There exists another compositional approach based on real-time calculus [29]. In this approach, an event stream is modeled with a pair of arrival curves that show how many events, minimum and maximum, may arrive during a given time window. The computation capability of a PE is modeled as a service curve. With a given load model of the input arrival curve and the service model of a PE, the real-time calculus computes the output arrival curve and the remaining service curve. While those compositional approaches are scalable and fast, the estimation accuracy is relatively poor since inter-PE dependency is not fully considered.

When the starting offset of each periodic application is known a priori and fixed, it is showed in [14] that the WCRT of each application could be tightly estimated by expanding all graphs up to the hyper-period of all applications (the least common multiple of periods) and analyzing the scheduling time bounds of each task. This method is called STBA (Scheduling Time Bound Analysis). Since the schedule pattern is repeated after the hyper-period, the possible interference between tasks can be narrowed significantly for conservative estimation. This method can be applied to both preemptive and non-preemptive scheduling policies. However, this approach is not generally applicable due to the assumption of fixed starting offsets. In order to overcome the limitation of this work, a hybrid analysis method between the schedule-bound analysis of [14] and the RTA method was proposed in [15]. They used the RTA method to compute the interference between applications instead of expanding graphs to the hyper-period. The scalability and general

applicability of this technique are claimed to be par with the compositional approach.

C. SDF GRAPH REDUCTION

For the real-time performance analysis of an SDF graph, it is necessary to convert it to a DAG. Since the graph size of the converted graph can be prohibitively large, a few techniques have been proposed to reduce the SDF graph size ([18], [19]). Two graph transformation techniques are introduced in [18] to reduce the complexity of timing analysis of large SDF graphs. One is to convert a large SDF graph with a regular structure into a smaller SDF graph by combining nodes with the same repetition count. The other is to convert a smaller SDF graph to a reduced HSDF that induces the same Max-Plus matrix formulation that is used to compute the throughput performance. Since these techniques aim to maintain the theoretical performance bound of the original SDF graph itself, the converted graphs cannot be used for parallel scheduling onto multiple processors.

The work in [19] is more relevant to the proposed clustering technique in that it reduces the number of nodes in the converted DAG by merging. To find the merging candidates, they consider the slack, which is the difference between the worst-case execution time of a node and its timing constraint. They merge nodes with positive slacks iteratively considering the deadlock possibility. While this approach respects the throughput and latency constraints of the original application graph, it is again performed before parallel scheduling, which affects the design space of mapping. Moreover, it does not consider the interference from the other SDF graphs.

III. SYSTEM MODEL

In this section, we preset the system model by introducing some terminologies and notations, as well as the assumptions made for the proposed methodology.

A. APPLICATION MODEL

We consider a set $\mathcal{G}$ of multiple applications (graphs). Each SDF graph $g \in \mathcal{G}$ is defined by a tuple $\langle N_g, E_g, T_g \rangle$. N_g, E_g, and T_g represent the set of nodes, the set of edges, and the period of the graph, respectively. Furthermore, we indicate a set of jobs as $\dot{N}_g$ and the set of edges between jobs as $\dot{E}_g$. The size of a set A is represented by $|A|$, so the number of nodes and jobs of graph g is denoted as $|N_g|$ and $|\dot{N}_g|$, respectively. Also, we express the graph containing node $n \in N_g$ as $G(n)$. Each node has a varying execution time represented by a tuple $[C^l(n), C^u(n)]$, each of which indicates the best and worst-case execution time (BCET and WCET) of the node on the mapped processing element. Also, we denote the average execution time of task τ as $\bar{C}(\tau)$. The average execution time is used for parallel schedule, while the execution time bound is considered in the WCRT analysis. We assume that all graphs are assigned distinct priorities, denoted by PR_g. It means that the priority of all nodes in a graph is lower or higher than all nodes in another graph. i.e., $\forall^{i,j} n_i \in g_i, n_j \in g_j$, if $PR_{g_i} > PR_{g_j}$ then $PR_{n_i} > PR_{n_j}$ where PR_n means the priority of node n. Between tasks in the same SDF graph, priorities are given to the jobs according to the scheduling order, which will be explained in the next section.

Each application can be run in a periodic, sporadic, or best-effort manner. The comparison between the three application types is shown in Table 2. A periodic application is invoked periodically with the period of T_g. There is no constraint on the inter-arrival time between applications. An application may start anytime during the execution of the other application: the starting offset is dynamic. Once it is started, a periodic application has a periodic arrival time, meaning that a task without any input data dependency has a periodic arrival time, and subsequent tasks with input dependency are executed in a data-driven manner: it becomes executable as soon as its input data are available. Also, we assume that the deadline is the same as the period of the graph (implicit deadline). If an application is invoked sporadically, T_g indicates the minimum distance between two invocations to consider the worst case interference among applications in the WCRT analysis. It is treated as a periodic application with a dynamic starting offset and its period becomes the minimum inter-arrival time [30]. While a periodic or sporadic application is assigned a user-defined priority, a best-effort application has the lowest priority without any deadline constraint, meaning that it can be executed only when the mapped PE is idle.

TABLE 2. Comparison of three application types.

Application Type	Priority	Period	Deadline
Periodic	User-defined	Static	Implicit deadline
Sporadic	User-defined	Minimum distance	Implicit deadline
Best-effort	Lowest	-	-

B. MAPPING AND SCHEDULING

The hardware platform consists of a set of heterogeneous PEs $\mathcal{PE} = \{P_0, P_1, \ldots, P_{m-1}\} = \mathcal{P} \cup \mathcal{NP}$ where m indicates the total number of PEs and $\mathcal{P}$ and $\mathcal{NP}$ represent the set of preemptive and non-preemptive PEs, respectively. Note that we allow a mixture of preemptive and non-preemptive PEs if the WCRT analysis tool supports it.

The mapping decision is made for each task, implying that all jobs of an SDF task within an iteration are mapped to the same processor. In case the granularity of an SDF task is as coarse as a thread that is the mapping and scheduling unit of an operating system, the task likely has internal states that need to be migrated if jobs are mapped onto different PEs. In order to avoid such state migration complexity, this assumption is made. A parallelizable SDF task without internal states is explicitly distinguished if jobs can be concurrently executed on multiple processors. Including parallelizable tasks, the mapping decision is denoted by a function $M : \mathcal{N} \rightarrow \mathbb{P}(\mathcal{PE}) - \varnothing$ where $\mathcal{N} = \cup_{g \in \mathcal{G}} N_g$ and $\mathbb{P}(\mathcal{PE})$ is a power set of $\mathcal{PE}$.

We assume that the self-timed scheduling is adopted for the scheduling of SDF tasks, where the scheduling order among

the tasks of the same SDF graph mapped on each processor is determined statically at compile time and does not vary at run time [31].

IV. PARALLEL SCHEDULING METHODOLOGY

The proposed methodology explores the design space of task mapping by iteratively evolving the mapping decision, as shown in Fig. 2. It consists of four main steps that are explained in this section. Fig. 3 (a) shows an example SDF graph that consists of 5 tasks that are mapped onto three PEs. In this example, we set the average execution time of tasks S, A, B, and D to 2 time units (tu), and C to 3 time units. The time unit tu is a symbolic unit that could be replaced with any actual unit of time such as $Kcycles$, ms, us, etc.

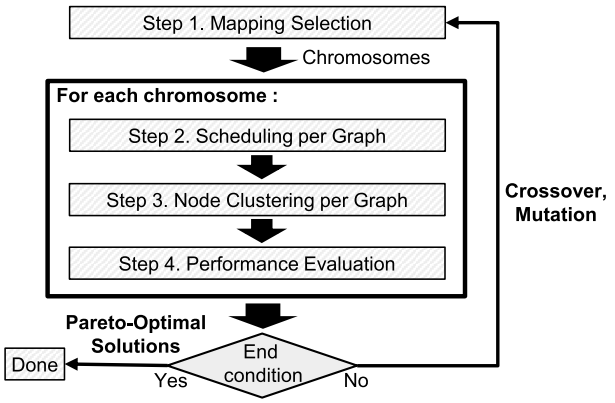


FIGURE 2. Proposed parallel scheduling methodology of multiple SDF applications based on a genetic algorithm.

A. MAPPING SELECTION

We use a genetic algorithm (GA) as a meta-heuristic for design space exploration of mapping. While any meta-heuristic can be applied, a GA is chosen since an acknowledged GA solver [32] is freely available. Mapping information of nodes is encoded in a chromosome in which each gene represents where the associated task is mapped to. In order to support a parallelizable task, a task can be mapped onto a subset of $\mathcal{PE}$. To notate a subset of $\mathcal{PE}$, we number each element of the power set of $\mathcal{PE}$, excluding the empty set. For instance, Fig. 4 shows an example chromosome for Fig. 3. If there is a node that is mapped on $P1$ and $P2$, the value of the gene will be 5. For multiple applications, the associated chromosomes will be concatenated to make a single long chromosome. Nevertheless, GA operations such as crossover and mutation are applied to each chromosome separately. Initially, we randomly generate as many chromosomes as a given parameter.

B. SCHEDULING PER GRAPH

In this step, we determine the execution order of tasks for each graph with a given mapping of each chromosome. For instance, we need to determine the execution order of jobs between tasks B and C on processor $P1$ when task mapping is assumed, as displayed in Fig. 3 (a). After executing the

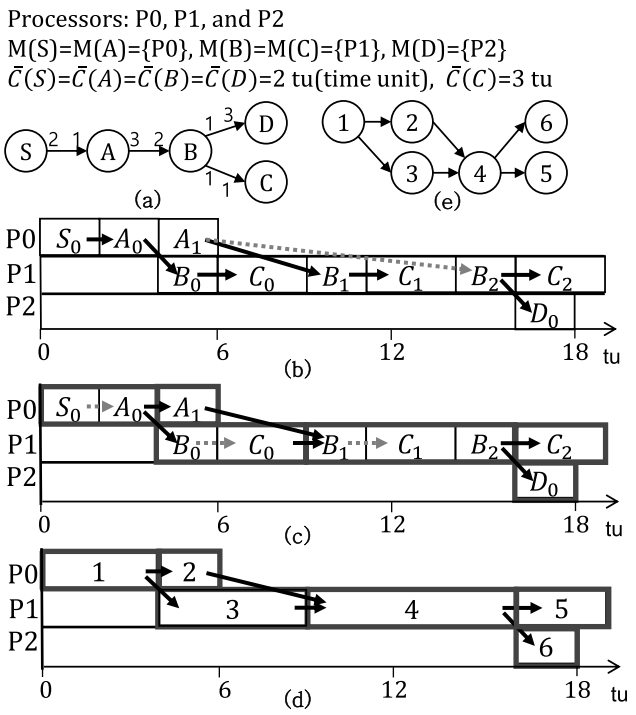


FIGURE 3. (a) An SDF graph example, (b) The schedule diagram for the SDF graph, (c) The schedule diagram after initial clustering, (d) The clustered schedule diagram, (e) The converted DAG.

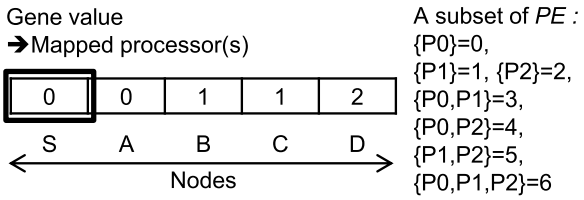


FIGURE 4. An example chromosome structure for Fig. 3.

first instance of task B, the second instance of task B and the first instance of task C are both executable. We use a list scheduling heuristic for the converted DAG of an SDF application, assuming that the priority of a job depends on the longest path from the job to the completion in order to minimize the schedule length of the SDF graph. The scheduling result is shown in Fig. 3 (b) with a schedule diagram where the vertical axis means the processing element, and the horizontal axis means the elapsed time. The schedule diagram is supplemented with dependency arcs between jobs. Note that the dependency arcs correspond to edges in the DAG, omitting the dependency between jobs of the same node. In case multiple jobs of the same node depend on a single source job, we represent only a single dependency arc from the source job to the first job; For example, dependency from A_1 to B_2 is omitted because there is a dependency arc from A_1 to B_1.

There are several notes worth mentioning in this step. First, the proposed methodology does not depend on a specific scheduling method even though we use a list scheduling heuristic in the current implementation: we can choose

a different execution order from Fig. 3 (b). Second, the priorities are given to the jobs, not tasks. Third, even though a task may have a varying execution time, we need to use a single representative value in the scheduling step; We use the average execution time $\bar{C}(\tau)$ of each task τ in the current implementation. The varying execution time will be considered in the WCRT estimation. Lastly, we may omit the scheduling step for an application whose task mapping is not changed since scheduling is performed on each graph separately.

C. NODE CLUSTERING PER GRAPH

With the scheduling result, node clustering is performed for each graph. The objective of node clustering is to reduce the number of nodes as much as possible without affecting the WCRT analysis. We propose a novel node clustering technique that will be explained in the next section in detail.

D. PERFORMANCE EVALUATION

With the clustered DAGs and the mapping and scheduling information, we evaluate the performance of each application. The basic tool for performance evaluation is the WCRT estimation of each application in the proposed Design-Space Exploration (DSE) framework. In the current implementation, the HPA (Hybrid Performance Analysis) [15] tool is used for WCRT estimation since it is freely available and easy to use, and its estimation accuracy is claimed to be better than the other WCRT estimation tools. The HPA tool supports an arbitrary mixture of preemptive scheduling and non-preemptive scheduling. Also, it supports dynamic starting offset of applicatons. The input information to the HPA tools includes a set of DAGs running concurrently with a given mapping of nodes onto PEs, priorities of nodes on each PE, and execution time range, [BCET, WCET], of each node.

A performance evaluation is performed for each chromosome. After the estimated WCRT of an application is obtained by HPA, it is compared with the deadline to check if it is violated. If any deadline violation is found, the fitness value for the corresponding chromosome (mapping) is set to a large number. Otherwise, the estimated WCRT becomes the fitness value of the chromosome. While the objective function can be modified, we assume that the objective is to minimize the WCRT of each selected application.

Among many chromosomes, we select some with smaller fitness values than the other and perform GA operations such as mutation and crossover to produce the offspring chromosomes. With the same number of chromosomes, we repeat these three steps until the fitness value for each application is converged, or the maximum number of iteration is reached.

V. NODE CLUSTERING TECHNIQUE

In this section, we explain the proposed node clustering technique after the mapping and scheduling of SDF tasks is performed. We denote the clustered task for jobs $n_1, n_2, \ldots, n_m$ as $Cluster(n_1, n_m)$, where n_1 and n_m are the first and the last job of the cluster, respectively. The size of $Cluster(n_1, n_m)$ is the number of jobs in the cluster, and the execution time of $Cluster(n_1, n_m)$ becomes the sum of the execution time of the jobs. In order to merge jobs without affecting the WCRT, clustering is performed to satisfy the following two conditions.

1) Clustering is performed within a processing element. It means that jobs n_1 to n_m are mapped to the same processor.
2) There should be no dependency between a job in the cluster and another job outside the cluster except for the start job, n_1, with its predecessor and the last job, n_m, with its successor. For example, in Fig. 3 (c), job C_0 cannot be merged(clustered) with B_1 since there is a dependency from A_1 to B_1.

In the proposed clustering method, we start with the initial clusters that merge all consecutive jobs on each processor. For instance, three initial clusters ($Cluster(S_0, A_1)$, $Cluster(B_0, C_2)$, $Cluster(D_0, D_0)$) are formed for the example of Fig. 3 (b). While the initial clusters satisfy the first condition, they may violate the second condition. We perform partitioning of each cluster recursively until all clusters satisfy the second condition. The pseudo-code of such a cluster partitioning algorithm is shown in Algorithm 1.

Algorithm 1 Cluster Partitioning Algorithm

```
1:  function partition((Cluster(n_1, n_m)))
2:      for i = 1; i ≤ m; i = i + 1 do
3:          if i ≠ 1 and ∃n_k ∉ Cluster(n_1, n_m) s.t.
4:  hasDependency(n_k, n_i) then
5:              partition(Cluster(n_1, n_{i-1}));
6:              partition(Cluster(n_i, n_m));
7:              return;
8:          else if i ≠ m and ∃n_k ∉ Cluster(n_1, n_m) s.t.
9:  hasDependency(n_i, n_k) then
10:             partition(Cluster(n_1, n_i));
11:             partition(Cluster(n_{i+1}, n_m));
12:             return;
13:         end if
14:     end for
15: end function
```

The function $hasDependency(n_k, n_i)$ returns whether dependency exists from job n_k to n_i. Lines 3-4 and 8-9 check if an intermediate job n_i in the cluster has input or output dependency relation with a job outside the cluster n_k. If such a dependency is found, the cluster is partitioned into two clusters based on the intermediate job. Then the cluster partitioning algorithm is recursively applied for the partitioned clusters (line 5-7, line 10-12). The final clustering result is obtained from this recursive algorithm with six clusters, as shown in Fig. 3 (c) for the previous example. Note that additional dependency arcs are added between the clusters on each PE to enforce the executive order. After clustering is completed, a reduced DAG can be created, as shown in Fig. 3 (e) for the previous example.

A. SUPPORTING NON-PREEMPTIVE PROCESSING ELEMENTS

In case a processing element adopts a non-preemptive scheduling policy, another condition is required for safe clustering. Under a non-preemptive scheduling policy, a high priority job τ_h of another application cannot preempt a low priority job while running. Suppose the execution time of the cluster of lower priority jobs is larger than the period of the high priority job. Then the high priority job may preempt the cluster at most once even though more than one preemption is possible if clustering is not performed. It means that clustering affects the real-time performance, which should be avoided. Thus we need to limit the size of a cluster on a non-preemptive PE.

To this end, we compute the maximum possible execution time ($MPET$) of each cluster that is mapped on $P \in \mathcal{NP}$. The $MPET$ of a cluster is the sum of the cluster execution time and all possible interference amount from high priority applications. The possible interference from a high priority application can be conservatively estimated by a demand bound function (DBF) $dbf_{g,p}(\Delta)$ [30] of the application, which indicates the maximum amount of resource demand on a processor p in any time interval Δ by graph g.

In case the execution time of a task is varying within the best-case execution time (BCET) and the worst-case execution time (WCET), care should be taken in the computation of the demand bound function of a high priority application. To conservatively find a maximum demand bound function on a given PE, BCETs need to be used for the jobs mapped onto the other PEs, while WCETs are used for the jobs mapped onto the given PE. Fig. 5 shows an example SDF graph and the demand bound function of the graph on a non-preemptive PE, $P1$. The demand bound function is computed assuming that the execution time of nodes E and G is 3 tu while that of node F is 2 tu.

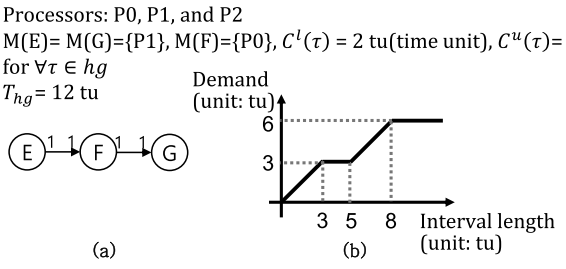


FIGURE 5. (a) A DAG *hg* associated with an SDF graph with the highest priority, and (b) The demand bound function of DAG *hg* on processor *P*1.

The $MPET$ of a cluster mapped on processor p can be determined as follows:

$$MPET = \sum_{i=1}^{m} C^u(n_i) + \sum_{hp \in hG} (dbf_{hp,p}(MPET)) \quad (1)$$

where hG is a set of task graphs that have higher priority than the cluster, and m is the number of jobs inside the cluster. The calculation is repeated until the $MPET$ value converges. Then the following inequality should be satisfied:

$$MPET \leq \min_{hp \in hG} (T_{hp}) \quad (2)$$

In case inequality (2) is not satisfied, we partition the cluster further until all partitioned clusters satisfy the inequality.

Example: Suppose that two SDF graphs of Fig. 3 (a) and Fig. 5 (a) share the resource, and the former has a lower priority than the latter. The left column of Table. 3 indicates the $MPET$ calculation on processor $P1$ for the graph of Fig. 3 (a). Through iterative computation, the $MPET$ of $Cluster(B_1, B_2)$ becomes 13 tu, but it violates the inequality (2) since the period of the graph hg is 12 tu. Therefore, the cluster should be partitioned. We split the last job of the cluster until the inequality satisfies. In this example, the $Cluster(B_1, B_2)$ is partitioned to two clusters, $Cluster(B_1, C_1)$ and $Cluster(B_2, B_2)$. After partitioning, we calculate the $MPET$ of the $Cluster(B_1, C_1)$, as displayed in the right column of Table 3. Inequality (2) is satisfied for $Cluster(B_1, C_1)$ and obviously for $Cluster(B_2, B_2)$. In summary, if the processor is non-preemptive, we need to check if the cluster execution time is no larger than the period of any higher priority application. It may incur additional partitioning.

TABLE 3. *MPET* computation on processor *P*1 for a cluster in Fig. 3 (c), considering the higher priority graph *hg* of Fig. 5 (unit: tu).

$Cluster(B_1, B_2)$	$Cluster(B_1, C_1)$
$MPET = 7 + dbf_{hg,P1}(7)$ $= 7 + 5 = 12$	$MPET = 5 + dbf_{hg,P1}(5)$ $= 7 + 3 = 10$
$MPET = 7 + dbf_{hg,P1}(12)$ $= 7 + 6 = 13$	$MPET = 5 + dbf_{hg,P1}(10)$ $= 5 + 6 = 11$
$MPET = 7 + dbf_{hg,P1}(13)$ $= 7 + 6 = 13$	$MPET = 5 + dbf_{hg,P1}(11)$ $= 5 + 6 = 11$

B. PROOF OF CORRECTNESS

We define a clustering algorithm as *correct* if it does not change the worst-case response time. We can prove that the proposed clustering algorithm is correct. Table 4 defines several notations to be used in the proof.

TABLE 4. Notations for proof.

Notation	Description
$RT(n)$	release time of node n
$ST(n)$	start time of node n
$FT(n)$	finish time of node n
$C(n)$	execution time of node n
$Delay_{n_s}(n_t)$	worst-case execution delay of node n_t due to blocking or preemption by n_s

Let jobs n_1 and n_m be the start and the end job of a cluster N, and τ be a job mapped to the same processor such that $G(\tau) \neq G(N)$. The release time and the finish time of cluster N can be defined as follows: $RT(N) = RT(n_1)$ and $FT(N) = FT(n_m)$. Clustering does not affect the worst-case response time if equations 3 and 4 are satisfied, meaning that it does

not change the worst-case interference amount from/to other tasks, respectively.

$$\sum_{i=1}^{m} Delay_{\tau}(n_i) = Delay_{\tau}(N) \tag{3}$$

$$\sum_{i=1}^{m} Delay_{n_i}(\tau) = Delay_{N}(\tau) \tag{4}$$

Note that there is no intra-graph interference since the task execution order is fixed.

Proof: Equation (3) holds.

Case 1: $PR_{\tau} > PR_{n_i}$

Interference by τ will occur when it is released during the time interval, as described in eq.(5) for non-preemptive scheduling and eq.(6) for preemptive scheduling.

$$RT(n_1) - C(\tau) < RT(\tau) \leq FT(n_{m-1}) \tag{5}$$

$$RT(n_1) - C(\tau) < RT(\tau) \leq FT(n_m) \tag{6}$$

The left sides of both equations indicate the blocking delay caused by the earlier release of job τ. Since the last job n_m cannot be preempted once started in case of non-preemptive scheduling, the right sides of the two equations are different. Then, the delay caused by τ before clustering can be described as eq.(7).

$$\sum_{i=1}^{m} Delay_{\tau}(n_i) = r \times C(\tau) \tag{7}$$

where r is the number of times τ occurs between the time interval. Since $RT(n_1) = RT(N)$, $FT(n_m) = FT(N)$ and $FT(n_{m-1}) = FT(N) - C(n_m)$, the intervals described in eq.(5) and eq.(6) are not changed after clustering. Since the worst-case interference by τ depends only on the interval size where r is not greater than one for non-preemptive scheduling because of MPET constraint at section V-A, eq.(3) is satisfied for both preemptive and non-preemptive scheduling cases.

Case 2: $PR_{\tau} < PR_{n_i}$

In the case of preemptive scheduling, the blocking delay caused by a lower priority job τ is zero so that eq.(3) holds trivially. Under a non-preemptive scheduling policy, τ may block at most once before clustering since all jobs in a cluster are consecutively scheduled. After clustering, the worst-case blocking delay by τ is also $C(\tau)$. Thus eq.(3) holds. □

Proof: Equation (4) holds.

Case 1: $PR_{\tau} > PR_{n_i}$

Since the priority of job τ is higher than all jobs $\{n_i\}$ in the cluster, there is no preemption delay caused by n_i to τ for both non-preemptive and preemptive scheduling cases. Thus, we only need to consider the blocking delay that may occur when there is a job n_i executing at the release time of τ under a non-preemptive scheduling policy. Since only one job in a cluster can block the execution of τ, the worst-case blocking delay can be computed as follows:

$$\sum_{i=1}^{m} Delay_{n_i}(\tau) = max(C(n_i)) \tag{8}$$

when jobs are clustered, however, the blocking delay caused by N will be the sum of all execution times of jobs. So, we need to redefine $Delay_N(\tau)$ as eq.(9) and apply it for the performance estimation tool.[2]

$$Delay_N(\tau) := max(C(n_i)) \tag{9}$$

Then, $\sum_{i=1}^{m} Delay_{n_i}(\tau) = Delay_N(\tau)$ holds.

Case 2: $PR_{\tau} < PR_{n_i}$

In the case of non-preemptive scheduling, job τ may experience only blocking delay. On the other hand, τ can be either delayed by blocking or preemption under a preemptive scheduling policy. Consider the blocking delay for both scheduling policies. If τ starts after $ST(n_1)$, the blocking lasts until $FT(n_m)$ since successive tasks continuously delay the task τ before clustering. This is the same after clustering because cluster N will also delay τ up to $FT(N) = FT(n_m)$. When a preemption occurs during the execution of job τ under a preemptive scheduling policy, τ will be resumed after $FT(n_m)$, which is the same after clustering. □

C. TIME COMPLEXITY

The time complexity of Algorithm 1 depends on the maximum number of recursive calls and the time complexity of the function *hasDependency*. For a given partition of size m, the complexity of function *hasDependency* is $O(|\dot{E}_g|)$ where g is the graph the cluster belongs to since it can be done by examining all edges of the DAG. Since a cluster can be partitioned into two clusters with size $m-l$ and $l(1 \leq l \leq m-1)$, the time complexity of partitioning a cluster with size m, a_m, can be expressed by the following recursive formula:

$$a_m = a_{m-l} + a_l + |\dot{E}_g| \tag{10}$$

The worst-case scenario of partitioning occurs when a cluster of size m is eventually divided into m clusters that have a single job. Then the time complexity of eq.(10) will satisfy the following inequality:

$$a_m \leq \sum_{k=1}^{m} a_1 + m \cdot |\dot{E}_g| \leq m + m \cdot |\dot{E}_g| \tag{11}$$

Therefore the time complexity for partitioning a cluster of size m becomes $O(m \cdot |\dot{E}_g|)$.

Let the number of jobs assigned to k-th processor be $|\dot{N}_g|_k$ $(1 \leq k \leq |\mathcal{PE}|)$. i.e., $\sum_{k=1}^{|\mathcal{PE}|} |\dot{N}_g|_k = \dot{N}_g$. The partitioning time for the k-th processor is bounded by $|\dot{N}_g|_k \cdot |\dot{E}_g|$ from eq.(11) with $m = |\dot{N}_g|_k$. Thus the maximum partitioning time is bounded by the following eq.(12):

$$\sum_{k=1}^{|\mathcal{PE}|} (|\dot{N}_g|_k) \cdot |\dot{E}_g| = |\dot{N}_g| \cdot |\dot{E}_g| \tag{12}$$

Therefore, the time complexity of Algorithm 1 is $O(|\dot{N}_g| \cdot |\dot{E}_g|)$. In the case of non-preemptive scheduling, we need to perform extra partitioning to satisfy inequality (2). Once

[2]The HPA tool allows us to change this definition. If such change is not supported, the correctness of clustering will not be guaranteed for non-preemptive PEs. Then we apply the proposed clustering technique to preemptive PEs only

the demand bound function is computed for each processing element, the complexity of this extra partitioning is $O(|\dot{N}_g|)$.

D. DEPENDENCY RELAXATION OPTIMIZATION

Since dependency between jobs is the main factor in clustering decisions, the fewer the dependencies, the more the clustering opportunities. Therefore we remove the inessential dependencies considering the priority of jobs and job execution times on each processor. Suppose that the graph in Fig. 3 (b) has the highest priority among all application graphs that are mapped on $P0$, then job A_1 will be executed immediately after A_0 without any delay. If $C^l(B_0)+C^l(C_0) \geq C^u(A_1)$, the dependency between A_1 and B_1 can be safely removed since B_1 is guaranteed to start after A_1 finishes. If we apply the clustering algorithm without this dependency arc, $Cluster(B_0, C_0)$ and $Cluster(B_1, B_2)$ will not be partitioned since there is no dependency between A_1 and B_1.

Suppose that there exists dependency from job X to job Y where job X belongs to the highest priority graph among all application graphs that are mapped on $M(X)$, and there exists another job $\{\tau_k\}$ such that $M(X) = M(\tau_k)$, $G(X) = G(Y) \neq G(\tau_k)$ where $PR_{G(X)} > PR_{G(\tau_k)}$. If the following condition is satisfied, the dependency from X to Y can be safely removed.

$$ST^l(Y) \geq FT^u(X) + I(M(X)) \tag{13}$$

$$I(P) = \begin{cases} max(C^u(\tau_k)) & \text{if } P \in \mathcal{NP} \text{ and } M(X) \neq M(Y) \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

where $ST^l(Y)$ and $FT^u(X)$ denotes the minimum start time of job Y and the maximum finish time of job X in the schedule, and $I(M(X))$ denotes the maximum blocking delay by lower priority jobs mapped onto the same PE with X, i.e., maximum blocking delay by τ_k. Note that if $M(X) = M(Y)$, then no blocking delay is needed to be considered since Y becomes executable after X completes its execution.

E. COMPARISON WITH EXISTING TECHNIQUES

In section II, two relevant techniques to SDF graph transformation are reviewed. To clarify the difference between those techniques and the proposed technique, we apply them to a simple SDF graph shown in Fig. 6 (a).

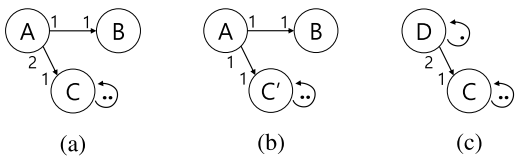


FIGURE 6. (a) An example SDF graph, (b) The best possible transformed graph by [19], (c) The transformed intermediate graph by [18].

Fig. 6 (b) and (c) show the reduced graphs obtained by applying the slack-based merging technique proposed in [19] and the two-stage method proposed in [18], respectively. Since graph reduction is performed before mapping and scheduling is performed, those techniques reduce the design space of mapping. In the former technique, two jobs of task C are merged, and the merged jobs will be mapped to the same PE. In addition, since no interference from other applications between the merged two jobs is allowed, the WCRT estimation is affected after merging. In the latter technique, tasks A and B are abstracted to task D whose execution time is set to the larger execution time between A and B for conservative real-time analysis. Similar to the former case, it reduces the design space of mapping, and the WCRT estimation is changed. On the other hand, in the proposed technique, the design space of mapping is unaffected since clustering is performed *after* a mapping decision is made.

VI. SUPPORTING MULTI-MODE SDF MODEL

Since the pure SDF model is not able to express dynamic behavior, several extensions have been proposed to expand the expression capability. One approach is to support multiple modes of operation in which each mode of operation is specified by an SDF graph. Finite State Machine (FSM)-based scenario-aware dataflow (SADF) [33], Mode Transition Machine (MTM) SADF [34], and mode-aware dataflow (MADF) [35] are some examples of this approach. As a common subset of those approaches, we assume an extended SDF model, denoted as multi-mode SDF (MMSDF), and explain how the proposed clustering technique can be applied.

Figure 7 (a) shows a simple MMSDF graph that has two modes of operation. Note that each port is annotated with two sample rates separated by a dash, meaning the sample rate may vary depending on the operation mode. In Fig. 7 (b) and (c), a separate SDF graph is drawn in each mode of operation. To distinguish two different modes of operation, we use uppercase letters in mode 1 and lowercase letters in mode 2 for the same node in the original MMSDF graph that uses Greek alphabets to denote tasks. For example, task A in Fig. 7 (b) and task a in Fig. 7 (c) indicate task α in Fig. 7 (a).

We assume that the task mapping is fixed while a task may have different execution times and dependency arcs in two modes. The execution times of nodes are all assumed to be 10 time units for mode 1, and 20 time units for mode 2. The iteration period of the application is the same in all modes. We assume that the mode change may occur dynamically at run-time at the iteration boundary.

How can we support a multi-mode SDF graph in the proposed parallel scheduling methodology? A naive but inefficient solution is to assume that all modes of operation are executed simultaneously, even though only a single operating mode is executed per iteration in real. While it over-estimates the WCRT of all applications significantly, it will give us a conservative estimation. To reduce the over-estimation amount, we propose to merge modes of operation after applying the proposed clustering algorithm of section V to each mode, naming the technique as mode clustering. For the example of Fig. 7, Fig. 7 (d) and (e) display the schedule diagrams after the node clustering algorithm is applied. Based on the node clustered schedule diagrams, an MMSDF graph

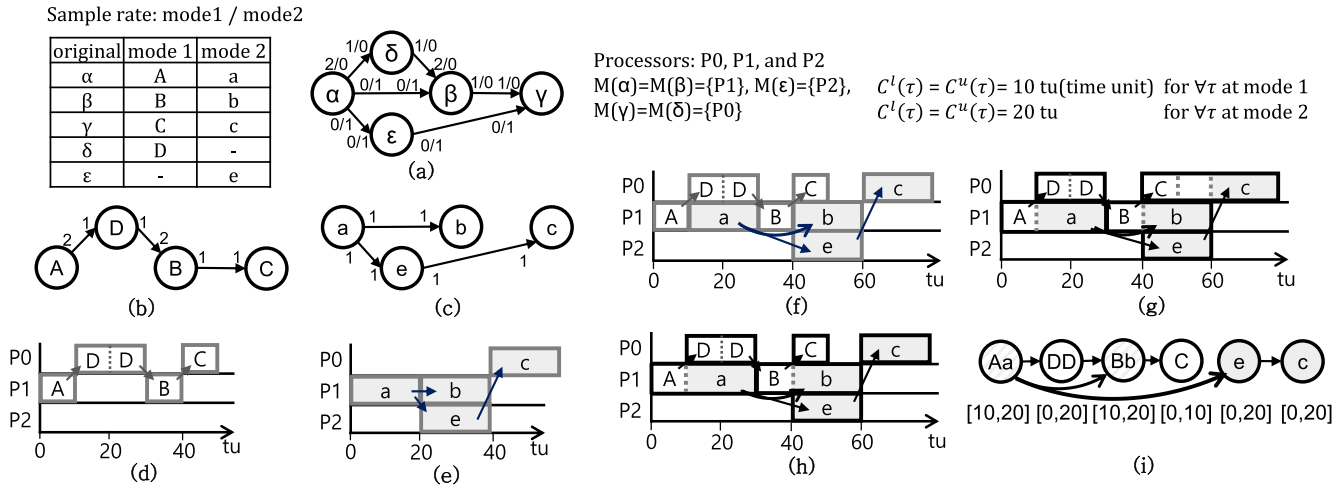


FIGURE 7. (a) An example MMSDF graph, (b) SDF graph for mode 1, (c) SDF graph for mode 2, (d) Schedule diagram of (b) After node clustering, (e) Schedule diagram of (c) After node clustering, (f) Interleaved schedule diagram of (d) And (e), (g) Interleaved schedule diagram after initial clustering, (h) Schedule diagram after mode clustering, and (i) The converted DAG after mode clustering.

consisting of multiple SDF graphs is converted into a single DAG after the mode clustering technique is applied. We prove the correctness of our mode merging technique by showing that the converted graph is a superset graph of all modes at the end of this section.

To merge two modes, we combine two schedule diagrams in an interleaved fashion, as illustrated in Fig. 7 (f), trying to make two adjacent clusters (jobs) belong to different modes on each processing element as much as possible.[3] The dependency arcs are preserved in the interleaved schedule.

Based on the interleaved schedule, we first construct initial mode clusters that merge consecutive jobs that belong to different modes, as shown in Fig. 7 (g); There should be *at most* one job for each mode in an initial cluster. Here, we denote the mode cluster of $n_1, n_2, \ldots, n_m$ as *Cluster*$(n_1, n_2, \ldots, n_m)$ where m is the total number of modes. An empty job ϕ represents a non-existing job in a certain mode. The best and worst execution time of a mode cluster is set to the minimum and maximum execution time of jobs in all modes, respectively. The minimum execution time is set to zero if the mode cluster has no job in a certain mode; in Fig. 7 (g), the best execution times of a *Cluster(DD,*ϕ*)* and *Cluster(*ϕ*,e)* are set to zero.

After initial mode clustering, we examine the mode clusters which does not contain ϕ one by one in the topological order and partition the cluster if necessary. We check the input dependency with a preceding cluster that is already examined. In case there is no input dependency or a preceding cluster has dependency in all modes, the cluster is not partitioned. Otherwise, the cluster is partitioned since unnecessary dependencies are added for a certain mode by mode clustering. When partitioned, the mode cluster is divided into original jobs, one per mode. In Fig. 7 (g) and (h), for example, *Cluster(B,b)* is not partitioned since its predecessor cluster *Cluster(A,a)* has a dependency path from job *A* to job *B* via job *D*, and direct dependency from job *a* to job *b*. On the other hand, *Cluster(C,c)* is partitioned, since job *b* in the precedent cluster *Cluster(B,b)* does not have dependency to job *c* while there exists a dependency from job B to job *C*.

Note that no dependencies are newly created during mode clustering. After partitioning is completed for all mode clusters, they can be converted into a DAG in a similar way to Section V. Fig. 7 (i) shows the converted DAG for the given example.

The correctness of the proposed mode clustering technique can be proved by showing that the schedule diagram of the converted DAG subsumes the schedule diagram of all modes. Let $f(\tau)$ be a mode cluster containing job τ. Since the maximum execution time of $f(\tau)$ is no smaller than $C^u(\tau)$ and the minimum execution time of $f(\tau)$ is no greater than $C^l(\tau)$, the mode cluster covers job τ. Also, there always exists a dependency from cluster $f(\alpha)$ to cluster $f(\beta)$ when there is a dependency between jobs α and β in a certain mode since the dependency arcs are preserved. Therefore, the schedule diagram after mode clustering subsumes the schedule diagram of each mode.

VII. EXPERIMENTS

A. BENCHMARKS AND SET-UP

For experiments, we select five real-life applications specified with an SDF graph or a multi-mode SDF graph, as displayed in Table 5. The hardware platform is Jetson TX2 [36], which is a heterogeneous system consisting of two Denver CPUs, four ARM A57 CPUs, and one NVIDIA Pascal GPU: the total number of PEs is seven. The *LaneDet* application consists of a set of filters to detect the lane from an input image. *ResNet* is an image classification application known

[3] In this section, to avoid confusion, we call the cluster which is the result of the previous clustering technique as a job, and the cluster of jobs in different modes as a *mode cluster*, or simply cluster.

TABLE 5. The benchmark applications.

App	PR_g	Models	$\|N_g\|$	Type	T_g(ms)
LaneDet	5	SDF	14	Time-driven	200
ResNet	4	SDF	28	Time-driven	500
H264Dec	3	MMSDF	17	Event-driven	33
Kmeans	2	SDF	5	Time-driven	500
Crypto	1	SDF	4	Best-effort	-

as ResNet-152. We assume that the mapping of *ResNet* is given a priori: four tasks for I/O interface are mapped on CPU while the other tasks are all mapped on GPU. The *Kmeans* application performs K-means clustering algorithm. It contains a parallelizable task that can be mapped onto multiple processors, except GPU. *H264Dec* has two modes of operation for video decoding. *Crpyto* represents a cryptography algorithm based on RSA. The last three applications can be run only on CPU cores.

Table 5 shows the real-time requirement of applications as well as the priority and the number of nodes in the SDF graph representation. Note that the *Crypto* application is a best-effort application without any deadline constraint. We measured the minimum, average, and maximum execution time of each node by running each application multiple times on each processing element. In case the execution time varies too much, we set the maximum time to 75% of the profiled maximum to make all applications schedulable on Jetson TX2. Since all benchmarks are soft real-time applications, such trimming will be acceptable, we believe. In case a task cannot run on a GPU, the execution time is set to infinite on the GPU. In addition, we measured the communication overhead between the CPU and GPU as a function of the size of the data being transmitted. Refer to [37] for detailed information on the benchmarks, including SDF specifications and the profiled execution time. In addition, extensive experiments are conducted with synthesized SDF graphs by using SDF3 (SDF For Free) [38]. All experiments have been performed on a system with Ubuntu 18.04.2 LTS, Intel Core i9-9900KF CPU @ 3.60GHz, and 64GB RAM. All proposed techniques are written in Java.

B. EXPERIMENT: SDF MODEL TRANSFORMATION

1) BENCHMARK APPLICATIONS

The first set of experiments is conducted to examine the effectiveness of the proposed node clustering technique for real-life benchmark applications. By randomly changing the mapping, the proposed technique is applied 10,000 times in total to obtain the average reduction of the number of nodes and the execution time. The reduction rate of nodes and the execution time of conversion and analysis are shown in Table 6.

The *ResNet* and *Kmeans* applications get the most benefit from the proposed clustering technique since they have some tasks that have many jobs to be scheduled sequentially on the mapped PE. Since the *LaneDet* and *Crypto* applications are specified by HSDF graphs, a little gain is achieved by clustering. *H264Dec* is a multi-mode SDF graph that is composed of two HSDF graphs. The node reduction ratio is slightly less than a half, which indicates that the proposed mode clustering reduces the number of nodes effectively.

TABLE 6. The number of nodes and the WCRT analysis time before and after node clustering.

Apps	Num. of Nodes		Execution Time (ms)		
			/w Clus.		w/o Clus.
	/w Clus.	w/o Clus.	Conv.	Analysis	Analysis
LaneDet	12.3	14	0.2	2.78	242.45
ResNet	3.0	258	1.7		
H264Dec	8.3	17	0.4		
Kmeans	47.2	1022	5.3		
Crypto	3.5	4	0.1		

Note that the WCRT analysis time by HPA is reduced down to 1.15% (87X speed-up) when the proposed node clustering is applied. Since the WCRT analysis is performed for all candidate mapping, such improvement is significant for exploring the design space of mapping. The table reveals that the clustering overhead in time is negligible, compared to the gain of reduced analysis time.

2) RANDOMLY GENERATED SDF GRAPHS

The second set of experiments is conducted with randomly generated SDF graphs with the SDF3 random graph generator that allows the user to change the number of nodes, the average repetition count of nodes, and the range of the node execution time. Table 7 shows how we vary the number of nodes and the average repetition counts. Note that the total number of nodes in the DAG is the product of the number of nodes and the average repetition count of nodes. We also vary the number of PEs onto which nodes are randomly mapped. We assume 20% of nodes are parallelizable.

TABLE 7. Node reduction ratio(%) for random SDF graphs.

Num. of Nodes	Num. of Processors	Repetition count				
		1	2	5	10	50
5	2	33.9	53.7	67.9	73.5	78.7
	4	15.2	38.9	55.8	62.1	69.5
	8	8.0	34.6	48.6	55.1	62.7
10	2	30.0	55.7	73.8	81.6	82.5
	4	11.4	42.6	63.2	72.6	74.0
	8	5.0	36.0	56.1	69.3	68.6
50	2	28.4	59.6	80.0	87.8	92.6
	4	9.4	47.0	72.4	81.9	89.3
	8	3.8	41.7	69.2	79.9	87.3

We make the execution time statistics of the randomly generated graph be similar to that of real-life benchmarks. We make 50% of the nodes have fixed execution times, and the rest have variable execution times. To model heterogeneous PEs, a node has different execution times on each PE. Since the performance difference between PEs is large in Jetson TX2, the execution time of nodes has a large variation from 1us to 1s, similarly to the case of real-life benchmarks. Since the minimum and maximum execution times of the nodes are 0.57 times and 2.1 times, respectively, of the average execution time in real-life benchmarks, we set

the minimum and maximum execution time of a node to be 0.5 and 3 times the average execution time if it has a variable execution time.

For each setting, we generate 60 random graphs and conduct each experiment 100 times to get the average impact of the proposed technique. The result is shown in Table 7 in terms of the node reduction ratio by clustering. As expected, the reduction ratio increases as the average repetition count increases. When the repetition count is 1, all SDF graphs are HSDF graphs so that only a little chance of clustering exists. Nonetheless, up to about 30% of reduction could be achieved on a two-processor system. While the difference is not significant, the reduction ratio decreases as the number of PEs increases since more jobs tend to be sequentially scheduled on a PE in case the number of PE is small.

Next, we compare the WCRT analysis time before and after the proposed SDF model transformation technique is applied when multiple applications are running concurrently. As shown in the first column of Table 8, we consider three different numbers of randomly generated graphs: 2, 5, and 7. The number of nodes, the number of PEs, and the average repetition count for each graph are set to 10, 4, and 10, respectively. We consider two types of systems in which all PEs are preemptible ($\mathcal{P}$) or non-preemptible ($\mathcal{NP}$), even though mixed scheduling systems can be supported. The period of each graph is set as the product of the number of graphs and the sum of each job's maximum execution time, which makes all applications schedulable for some mappings.

TABLE 8. Comparison of the reduction ratio and execution times for multiple graphs.

$\|\mathcal{G}\|$	Reduction Ratio (%)		Execution Time (ms)			
	/w Clustering		/w Clustering [Conversion/Analysis]		w/o Clustering	
	P.	NP.	P.	NP.	P.	NP.
2	73.8	73.8	1.62/1.78	1.83/1.59	51.57	41.92
5	73.5	73.3	3.42/2.76	4.99/2.66	152.02	114.90
7	73.5	73.3	5.30/4.05	7.23/3.86	204.10	129.60

For each experiment, we use 60 randomly generated graphs with random mapping and repeat the experiment 100 times to obtain the average result. The experimental result is summarized in Table 8. Similarly to Table 7, the number of nodes is decreased significantly by the clustering technique, and the WCRT analysis time is reduced drastically when the clustering technique is applied. Also, as the number of graphs is increased, the overall execution time is increased. It confirms the necessity and benefit of the proposed technique for fast design space exploration of multiple SDF graphs. Note that the node reduction ratio for non-preemptive systems is slightly lower than that for preemptive systems. It is because there exist some clusters that are further partitioned to satisfy the *MPET* constraint of inequality (2). Since the difference is not significant, the WCRT analysis times show no noticeable difference.

C. EXPERIMENT: PARALLEL SCHEDULING

The proposed parallel scheduling methodology is applied to find optimal mappings of real-life benchmarks onto seven PEs in Jetson TX2. As explained above, the mapping of *ResNet* is assumed to be given and fixed. We aim to minimize the number of used PEs and the WCRT of *ResNet*, *Crypto*, and *Kmeans*. The *LaneDet* and *H264Dec* applications have throughput constraints that are equal to their periods. We set the number of chromosomes to 100 and generate 25 offspring per iteration. We perform uniform crossover and 5% mutation with a rate of 95% and 70%, respectively, based on NSGA2 [39] selector algorithm. The genetic algorithm terminates after 15, 000 generations. For fast convergence, all tasks that can be performed on the GPU are mapped to the GPU, and the remaining tasks are randomized in the initial chromosomes.

Fig. 8 shows three Pareto-optimal solutions found: minimum WCRT for *Kmeans* (*Kmeans_best*) and *Crypto* (*Crypto_best*), and the minimum number of used PEs. All the solutions satisfy the throughput constraints of applications. The *Relative* WCRT is computed as the ratio of the obtained WCRT over the best WCRT for each application. Since the *ResNet* is compute-intensive, the mapping freedom of the *ResNet* while satisfying timing constraints is limited, which explains why the relative WCRT of *ResNet* does not vary in three different Pareto-optimal solutions. On the other hand, the WCRT of *Crypto* varies most with a maximum 8.6 times larger WCRT between *Kmeans_best* and *Crypto_best*. This is because *Crypto* is a best-effort application with the lowest priority among all. The best-effort application is likely to have a very large WCRT value because of the lowest priority and no deadline constraint. On the contrary, the variation of the WCRT value of *Kmeans* is smaller since it has a higher priority than the Crypto, and the WCRT value does not exceed the given deadline (i.e., the WCRT variation of a periodic or sporadic application is relatively small since it should satisfy the throughput constraint). Also, through this experiment, we confirm that the number of used PEs can be reduced while satisfying the constraints. It is possible to run the benchmark applications simultaneously onto only four PEs without the constraint violation. Since a single generation of evolution takes about 250ms, the total evolution time is about 1 hour. Without clustering, it would take two orders of magnitude longer time.

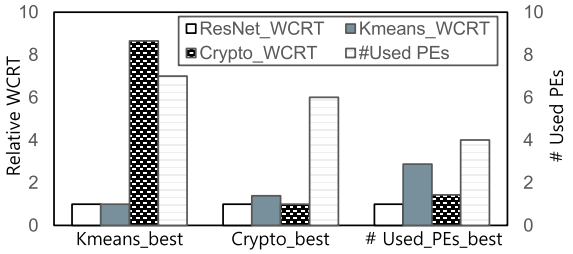


FIGURE 8. Three Pareto-optimal solutions with four objectives.

D. COMPARISON WITH THE SOTA PARALLEL SCHEDULING TECHNIQUE

In this section, we compare our proposed technique with the state-of-the-art (SOTA) parallel scheduling technique [9] for multiple graphs. Based on the schedulability analysis used in [8], it produces the sub-optimal scheduling results of multiple deep learning applications in terms of the response time of each application under the given response time constraints with a genetic algorithm, similar to our proposed technique. Since it assumes that deep learning applications are represented as DAGs, we used multiple DAGs as input even though our proposed technique can accept general SDF graphs. We used the same benchmark applications as inputs, *Squeezenet* [40] and *MobileNet v2* [41], with the profiled task and communication overhead used in [9]. For a fair comparison, we set the iteration number of genetic algorithms of each methodology equally to 5, 000 generations.

Figure 9 shows the Pareto-optimal solutions for two applications, in which the x and y axes represent the WCRT values of *Squeezenet* and *MobileNet v2*, respectively. In the figure, each dotted line represents a set of Pareto-optimal solutions obtained by the proposed and the reference technique. Since a lower WCRT value means a less pessimistic result, it is better when a solution is placed in the lower left than the other one. As can be seen from the figure, the solutions from the proposed methodology dominate the solutions from the reference technique. For the same *Squeezenet* WCRT value, our solution gets 10% to 32% smaller WCRT value for *MobileNet v2* than the reference technique. Similarly, for the same *MobileNet v2* WCRT value, our solution obtains 55% to 58% smaller WCRT value for *Squeezenet* than the reference technique. It is notable that our proposed methodology can be applied to SDF and multi-mode SDF graphs by node clustering technique described above in addition to the better performance, compared with the SOTA technique that assumes DAGs for application specification.

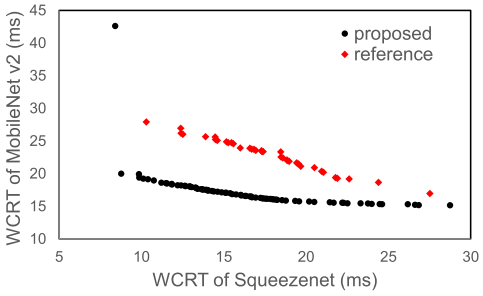


FIGURE 9. Pareto-optimal graphs of the proposed and reference technique.

VIII. CONCLUSION AND FUTURE WORK

In this paper, a novel parallel scheduling methodology is proposed to schedule multiple SDF graphs with diverse real-time constraints onto heterogeneous processors. It explores the design space of task mapping with a genetic algorithm. For each mapping candidate, scheduling of jobs is conducted with a list scheduling heuristic. For a given mapping and scheduling result, the worst-case performance of each application is analyzed with an existing WCRT analysis tool. To reduce the WCRT analysis time, we propose a node clustering technique that does not affect the design space and the estimated WCRT. The correctness of the proposed clustering technique is formally proved. The node clustering technique is extended to support the multi-mode SDF model by mode clustering. The effectiveness of the proposed method is verified with extensive experiments. Our clustering technique achieves faster the WCRT analysis time by 87 times for a given set of real-life benchmark applications and at least 3.8% to 92.6% node reduction ratio for randomly generated graphs with various configurations. Our proposed methodology could find better solutions compared to the SOTA methodology up to 58% smaller WCRT value of an application.

To overcome the limitation of the SDF model in the expression capability, several extensions of the SDF model have been proposed besides the MMSDF model. For example, a library task model [42] is introduced to support resource sharing, and the SDF/L model [43] is proposed to express loop structures explicitly. It remains a future work to extend the clustering technique to support these extended models. It is likely that there are other tasks that are running concurrently with SDF applications on the mapped processors. It is necessary to consider the interference between those tasks with the SDF applications in the WCRT analysis, which is left as another future work.

ACKNOWLEDGMENT

This paper is a revised version of the first author's master dissertation [1].

REFERENCES

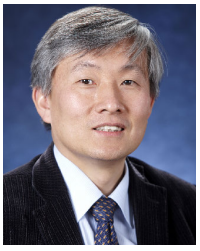
[1] D. Jeong, "Parallel scheduling of multiple SDF graphs onto heterogeneous processors," M.S. thesis, Seoul Nat. Univ., Seoul, South Korea, 2020.
[2] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, no. 9, pp. 1235–1245, Sep. 1987.
[3] A. H. Ghamarian, S. Stuijk, T. Basten, M. C. W. Geilen, and B. D. Theelen, "Latency minimization for synchronous data flow graphs," in *Proc. 10th Euromicro Conf. Digit. Syst. Design Architectures, Methods Tools (DSD)*, Aug. 2007, pp. 189–196.
[4] J. Choi, H. Oh, S. Kim, and S. Ha, "Executing synchronous dataflow graphs on a SPM-based multicore architecture," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, 2012, pp. 664–671.
[5] W. Che and K. S. Chatha, "Unrolling and retiming of stream applications onto embedded multicore processors," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, 2012, pp. 1272–1277.
[6] H. Jung, H. Oh, and S. Ha, "Multiprocessor scheduling of an SDF graph with library tasks considering the worst case contention delay," in *Proc. 14th ACM/IEEE Symp. Embedded Syst. Real-Time Multimedia*, Oct. 2016, pp. 1–10.
[7] O. Kermia and Y. Sorel, "A rapid heuristic for scheduling non-preemptive dependent periodic tasks onto multiprocessor," in *Proc. ISCA 20th Int. Conf. Parallel Distrib. Comput. Syst. (PDCS)*, 2007.
[8] 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 Autom. Conf.*, Jun. 2016, pp. 1–6.
[9] 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.
[10] H. I. Ali, B. Akesson, and L. M. Pinho, "Generalized extraction of real-time parameters for homogeneous synchronous dataflow graphs," in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process.*, Mar. 2015, pp. 701–710.

[11] S. Niknam, P. Wang, and T. Stefanov, "Hard real-time scheduling of streaming applications modeled as cyclic CSDF graphs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 1549–1554.
[12] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis—The SymTA/S approach," *IEE Proc.-Comput. Digit. Techn.*, vol. 152, no. 2, pp. 148–166, 2005.
[13] P. S. Kurtin, J. P. H. M. Hausmans, and M. J. G. Bekooij, "Combining offsets with precedence constraints to improve temporal analysis of cyclic real-time streaming applications," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2016, pp. 1–12.
[14] J. Kim, H. Oh, J. Choi, H. Ha, and S. Ha, "A novel analytical method for worst case response time estimation of distributed embedded systems," in *Proc. 50th Annu. Design Autom. Conf. (DAC)*, 2013, pp. 1–10.
[15] J. Choi, H. Oh, and S. Ha, "A hybrid performance analysis technique for distributed real-time embedded systems," *Real-Time Syst.*, vol. 54, no. 3, pp. 562–604, Jul. 2018.
[16] M. Gonzalez Harbour, J. G. García, J. P. Gutiérrez, and J. M. D. Moyano, "MAST: Modeling and analysis suite for real time applications," in *Proc. 13th Euromicro Conf. Real-Time Syst.*, 2001, pp. 125–134.
[17] J. Diemer, P. Axer, and R. Ernst, "Compositional performance analysis in python with pycpa," in *Proc. 3rd Int. Workshop Anal. Tools Methodologies Embedded Real-Time Syst. (WATERS)*, 2012, p. 178.
[18] M. Geilen, "Reduction techniques for synchronous dataflow graphs," in *Proc. DAC*, 2009, pp. 911–916.
[19] H. I. Ali, S. Stuijk, B. Akesson, and L. M. Pinho, "Reducing the complexity of dataflow graphs using slack-based merging," *ACM Trans. Design Autom. Electron. Syst.*, vol. 22, no. 2, pp. 1–22, Mar. 2017.
[20] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1990.
[21] H. Oh and S. Ha, "A static scheduling heuristic for heterogeneous processors," in *Proc. Eur. Conf. Parallel Process.* Springer, 1996, pp. 573–577.
[22] 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.
[23] E. S. 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.
[24] H. Yang and S. Ha, "Pipelined data parallel task mapping/scheduling technique for MPSOC," in *Proc. DATE*, 2009, pp. 69–74.
[25] H. Yang and S. Ha, "ILP based data parallel multi-task mapping/scheduling technique for MPSOC," in *Proc. ISOCC*, 2008, pp. I-134–I-137.
[26] S. K. Roy, R. Devaraj, and A. Sarkar, "Optimal scheduling of PTGs with multiple service levels on heterogeneous distributed systems," in *Proc. Amer. Control Conf. (ACC)*, Jul. 2019, pp. 157–162.
[27] S. K. Roy, R. Devaraj, A. Sarkar, K. Maji, and S. Sinha, "Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems," *J. Syst. Archit.*, vol. 105, May 2020, Art. no. 101706. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1383762119305132
[28] X. Liu, R. Xu, Y. Cheng, and P. Zhang, "A novel hefts-l algorithm for scheduling large number of DAG tasks," in *Proc. 3rd Workshop Adv. Res. Technol. Ind. (WARTIA)*. Amsterdam The Netherlands: Atlantis Press, 2017.
[29] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Proc. DATE*, vol. 3, 2003, p. 10190.
[30] 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.
[31] E. A. Lee and S. Ha, "Scheduling strategies for multiprocessor real-time DSP," in *Proc. IEEE Global Telecommun. Conf. Exhib. Commun. Technol. 1990s Beyond*, Nov. 1989, pp. 1279–1283.
[32] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J: A modular framework for meta-heuristic optimization," in *Proc. 13th Annu. Conf. Genetic Evol. Comput.*, 2011, pp. 1723–1730.
[33] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *Proc. Int. Conf. Embedded Comput. Syst. Archit., Modeling Simulation*, Jul. 2011, pp. 404–411.
[34] H. Jung, C. Lee, S.-H. Kang, S. Kim, H. Oh, and S. Ha, "Dynamic behavior specification and dynamic mapping for real-time embedded systems: Hopes approach," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4s, pp. 1–26, 2014.
[35] J. T. Zhai, "Adaptive streaming applications: Analysis and implementation models," Ph.D. dissertation, Dept. Sci., Leiden Inst. Adv. Comput. Sci. (LIACS), Leiden Univ., Leiden, The Netherlands, 2015.
[36] Nvidia. *Jetson TX2*. Accessed: Jan. 20, 2021. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/
[37] *SDF Benchmarks*. Accessed: Jan. 20, 2021. [Online]. Available: https://github.com/Dukejung/sdfBenchmarks
[38] S. Stuijk, M. Geilen, and T. Basten, "SDF^3: SDF for free," in *Proc. 6th Int. Conf. Appl. Concurrency Syst. Design (ACSD)*, 2006, pp. 276–278.
[39] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[40] 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
[41] 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.
[42] H.-w. Park, H. Jung, H. Oh, and S. Ha, "Library support in an actor-based parallel programming platform," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 340–353, May 2011.
[43] H. Hong, H. Oh, and S. Ha, "Hierarchical dataflow modeling of iterative applications," in *Proc. 54th Annu. Design Autom. Conf.*, Jun. 2017, pp. 1–6.

DOWHAN JEONG received the B.S. and M.S. degrees in computer science and engineering from Seoul National University, Seoul, South Korea, in 2018 and 2020, respectively. His current research interests include HW/SW codesign of embedded systems, design automation, and system performance estimation.

JANGRYUL KIM 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 design space exploration of embedded systems.

MARI-LIIS OLDJA received the B.S. degree in computer science from the University of Tartu, Estonia, in 2014. She is currently pursuing the M.S. degree in computer science and engineering with Seoul National University, Seoul, South Korea. Her current research interests include design automation, data parallel scheduling, and performance estimation of embedded systems.



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 Internet of Things. He has actively participated in the premier international conferences in the EDA area, for instance serving CODES+ISSS, in 2006, ASPDAC, in 2008, as the Program Co-Chair, and ESWEEK, in 2018, as the General Chair. He is a member of ACM.