Adaptive Abstraction-Level Conversion Framework for Accelerated Discrete-Event Simulation in Smart Semiconductor Manufacturing

MOON GI SEOK\textsuperscript{1}, (Member, IEEE), WENTONG CAI\textsuperscript{1}, (Member, IEEE), HESSAM S. SARJOUGHIAN\textsuperscript{2}, AND DAEJIN PARK\textsuperscript{3}, (Member, IEEE)

\textsuperscript{1}School of Computation Science and Engineering (SCSE), Nanyang Technological University, Singapore 639798
\textsuperscript{2}Arizona Center for Integrative Modeling and Simulation (ACIMS), Arizona State University, Tempe, AZ 85281, USA
\textsuperscript{3}School of Electronics Engineering, Kyungpook National University, Daegu 41566, South Korea

Corresponding author: Wentong Cai (aswtcai@ntu.edu.sg)

This research is supported by Agency for Science, Technology and Research (A*STAR) Singapore, under its Research Innovation Enterprise (RIE) 2020 Advanced Manufacturing and Engineering (AME) Industry Alignment Fund-Pre-Positioning (IAF-PP) Program Grant No. A19C1a0018, and partially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF2019R1A2C2005099, NRF2018R1A6A1A030251109).

\iffalse
\textsuperscript{1}School of Computation Science and Engineering (SCSE), Nanyang Technological University, Singapore 639798
\textsuperscript{2}Arizona Center for Integrative Modeling and Simulation (ACIMS), Arizona State University, Tempe, AZ 85281, USA
\textsuperscript{3}School of Electronics Engineering, Kyungpook National University, Daegu 41566, South Korea

Corresponding author: Wentong Cai (aswtcai@ntu.edu.sg)

This research is supported by Agency for Science, Technology and Research (A*STAR) Singapore, under its Research Innovation Enterprise (RIE) 2020 Advanced Manufacturing and Engineering (AME) Industry Alignment Fund-Pre-Positioning (IAF-PP) Program Grant No. A19C1a0018, and partially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF2019R1A2C2005099, NRF2018R1A6A1A030251109).
\fi

\begin{abstract}
Speeding up the simulation of discrete-event wafer-fabrication models is essential for fast decision-making to handle unexpected events in smart semiconductor manufacturing because decision-parameter optimization requires repeated simulation execution based on the current manufacturing situation. In this paper, we present a runtime abstraction-level conversion approach for discrete-event fab models to gain simulation speedup. During the simulation, if the fab’s machine group model reaches a steady state, then the proposed method attempts to substitute this group model with a mean-delay model (MDM) as a high abstraction level model. The MDM abstracts detailed event-driven operations of subcomponents in the group into an average delay based on the queuing modeling, which can guarantee acceptable accuracy in predicting the performance of steady-state queuing systems. To detect the steadiness, the proposed abstraction-level converter (ALC) observes the queuing parameters of low-level groups to identify the statistical convergence of each group’s work-in-progress (WIP) level. When a group’s WIP level is converged, the output-to-input couplings between the models are revised to change a wafer-lot process flow from the low-level group to a MDM. When the ALC detects lot-arrival changes or any wafer processing status change (e.g., a machine-down), the high-level model is switched back to its corresponding low-level group model. During high-to-low level conversion, the ALC generates dummy wafer-lot events to re-initialize the machine states. The proposed method was applied to various case studies of wafer-fab systems and achieved simulation speedups up to about 4 times with 0.6 to 8.3% accuracy degradations.
\end{abstract}

\begin{IEEEkeywords}
Abstraction-level conversion, wafer fabrication, discrete-event modeling, smart manufacturing.
\end{IEEEkeywords}

\section{I. INTRODUCTION}

For smart manufacturing in wafer-fabrication (wafer-fab) systems, physical manufacturing-execution systems (MESs) collaborate with monitoring and controlling facilities (MCFs) to respond to rapid changes in production plans or machines’ statuses. In the smart fab, the MCF collects real-time operation data from monitoring MESs and manages highly accurate fab models through the model-parameter calibration based on the collected data. The fab models’ high accuracy guarantees reliable performance expectations, considering the decision candidates of various scenarios when an unexpected circumstance is met. After evaluating the short- or middle-term performance for decision candidates, process engineers in the MCF can efficiently influence the MEMs by revising the production plans or dispatching rules based on optimal decision candidate(s).

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

\vfill
There are several studies on simulation-based optimization (SBO) in fab-production planning and scheduling problems [1]–[5]. For the production-plan decision, the previous studies formulate the planning problem using linear programming (LP, or a mixed integer programming) approaches to optimize the release quantities of wafer-lot in a period. Specific parameter values (e.g., cycle time and work-in-progress (WIP) inventory level), which form an LP’s objective function and constraints, are obtained from wafer-fab simulation results under a given production scenario; the scenario determines the rest parameter values of the LP’s objective function.

For the scheduling optimization, many studies have utilized wafer-fab models to minimize the cycle time (or variance of cycle time, and so on) considering various dispatching rules and lot-release policies. During the optimization, various optimization algorithms are employed, such as genetic algorithm, simulation annealing, ordinal algorithm, and so on. However, the wafer-fab model is designed conventionally by discrete-event (DE) modeling, which enables a process engineer to build a hierarchical fab system using subcomponents whose dynamic and interactional operations are modeled in an event-driven manner [9]–[12].

Some researches have introduced queuing models for the fast evaluation of wafer-fab system performance [6]–[8]. However, queuing models have an accuracy problem because they abstract the production dynamics into static (time-independent) equilibrium equations. Thus, queuing models have a certain level of limitation in considering the dynamic effects: unexpected events (e.g., machine down or wafer-lot rework), lot-release changes (caused by new demand, production completion, or release policy) in high-mix fabs, and lot-priority changes according to target dispatching rules.

For the simulation speedup of DE fab models, some studies utilize the parallel simulation approach using multiple CPU cores [13], [14]. The parallel simulation of DE fab models can lead the speedup of a single simulation instance for a design-candidate evaluation. During the SBO, however, when executing multiple independent simulation instances to consider the DE models’ stochastic properties or to comply parallel-optimization algorithms, the parallel simulation can be inefficient because of the data and time synchronization overhead among running models on different CPU cores, compared to running multiple independent simulation instances on multiple cores [15], [16].

For the SBO’s fast response in the smart fab, this paper focuses on accelerating single simulation instances of DE fab models without utilizing multiple CPU cores. For the speedup, some studies have developed approximated analytical models of detailed DE fab models [17]–[20]. The abstracted models have evolved in various forms: a constant delay, probability distributions, or exponential/quantile functions. They are trained using simulation results according to a design of experiments (DOE). Similar to queuing modeling, since the abstracted models do not include dynamic parameters, we cannot evaluate changing factors in the middle of simulation; the changes can be caused by unexpected machine breakdown, production plan changes, etc. Moreover, when we handle unexpected situations that are out of the DOE coverages of the trained model, the model should be retrained using simulation results based on a new scenario-dependent DOE. The low flexibility caused by the static abstraction, which requires the model training before SBO, makes it challenging to utilize the models in managing smart fabs’ various situations.

To resolve the static abstraction problem, we propose an adaptive abstraction-level change approach, which chooses active models among differently abstracted models representing equivalent target subsystems in runtime. The approach has been mainly applied to agent-based simulation, which adaptively reduces the number of agents by abstracting groups of spatial agents into meta-agents that subsume individual behaviors when the clustered agents act together for a long time [22]–[24]. The proposed approach is a first attempt to apply the conversion approach to manufacturing-system simulation. The abstraction-conversion condition is strictly relevant to steady states of detailed models.

This paper proposes a framework that abstracts detailed event-driven operations of the fab’s machine-group DE models into statistical mean-delay models (MDMs) when the group models turn into a steady state. When a divergence condition is encountered in a steady-state group, then a DE group model becomes active instead of a MDM to simulate group’s transient state. The MDMs are designed based on the queuing models, which guarantee high accuracy in steady-state queuing systems. We denote the DE fab model (which serves the microscopic dynamics of lot-process flow) as a low-abstraction-level (or low-level) model and the MDM as a high abstraction-level (high-level) model. We extended our previous method to cover various lot-release and scheduling policies and to provide reliable detection methods of convergence to or divergence from a steady state [21].

In the proposed approach, a machine group’s steady state means that the number of waiting jobs (wafer lots) in the group’s queue converges. A key component, abstraction-level converter (ALC), collects the queuing-model parameters of a machine group; the parameters include the inter-arrival and queue-waiting times of a probing DE machine-group model. To prevent a hasty decision of local convergence (or divergence) based on queuing-parameter values in a short time period, the averaged queueing-parameter values within a specific time period is treated as single sample observation. The hasty decision can increase the number of abstraction-level conversions even in unsteady states. When the number of observations reaches a defined sample size, the ALC invokes a Kolmogorov–Smirnov (KS) test to determine whether two consecutive samples’ queuing-parameter distributions become statistically indistinguishable. If the time invariances of inter-arrival and waiting times are detected, the ALC concludes that the group’s WIP level is converged in a steady state based on Little’s law.

If a convergence condition of a steady state is confirmed, the ALC informs its MDM of the observed average delays.
Then, the ALC requests the simulation engine to change the flow of incoming jobs (wafer-lot events) from the low- to the high-level model by adjusting the output-to-input port connections between related models, as shown in Fig. 1. To detect a divergence condition from the WIP steady state, the ALC monitors (1) a statistical arrival-rate change from the converged distribution or (2) any event arrival that influences the machine’s process time. When a divergence condition is met, the ALC restores the low-level model by re-directing the incoming wafer-lot flow. For maintaining consistency between two-level models during the high-to-low level conversion, the ALC generates the dummy events to the low-level model to mimic the lot-processing workload of the high-level model.

Overall, the contributions of our proposed method can be summarized as follows.

- The proposed dynamic switching abstracts the detailed internal operations and event exchanges of steady-state group models into an observed mean delay during the runtime, which results in several speedups with relatively small accuracy reductions.

- We propose a steady-state convergence detection method based on a KS-test and an event-group concept. In the iterative divergence test, the ALC compares two inter-arrival time distributions: a fixed distribution previously observed at the steady-state detection and current time-varying distribution. Considering the inter-arrival time distribution of a steady state is fixed, we introduce an effective KS-test method utilizing a proposed data management method to reduce the computation overhead.

- To maintain consistency at the high-to-low level conversion, we present the dummy-event concept to mimic the machine and queue busyness of a steady state.

The rest of the paper is organized as follows. Section II describes the proposed multi-level modeling approach. Section III details the WIP-convergence detection and abstraction-level transition. Section IV shows the detailed procedures of MDM and coupling changes. Section V shows the experimental results of the proposed approach, and Section VI concludes the paper.

II. ABSTRACTION-LEVEL CONVERSION APPROACH

In this section, we introduce an overview of the DE fab models. Then, we present a perspective of the model-abstraction conversion based on the WIP-level steadiness and the structure of a framework for the adaptive abstraction-level change in simulation runtime.

A. BACKGROUND OF DISCRETE-EVENT MODELING OF WAFER-FAB MANUFACTURING SYSTEMS

Many modelers have attempted to represent the fab system as a set of event-driven model components whose state changes are triggered by events. The primary model components are inventory models and machine-group models (also called workstation models). Still, other models (e.g., the automatic guided vehicle (AGV) models for wafer delivery) can be included in the fab, depending on the modeler’s objectives.

The inventory models release blank wafers into machine groups based on the required demands. There are various release policies, which can be categorized into push-based (open-loop) and pull-based (closed-loop) release policies. The push-based policy has a uniform release rate regardless of any current fab status, so it does not make an adaptive adjustment for a certain disturbance (e.g., machine down). The pull-based policy generates the wafer lots based on the WIP- or workload-related feedback, so that the release rates vary dynamically in response to the disturbances. The examples of pull-based policies are constant work in progress (CONWIP) and workload regulation (WR).

The subcomponents of machine-group models are equipment (or machine) models, operator models, and wafer-lot queues. An equipment model performs a specific process according to a recipe of incoming wafer lots that decides the operational parameters of subprocess steps, the setup time, and so on. The process steps of each machine are modeled considering the sequence of subprocess steps, the number of chambers, and the number of processing wafers (or lots).

During the simulation, wafers are processed through a series of machines in different workstations. For a particular process step in a machine group, waiting wafer lots in a group-queue model can be dispatched to numerous machines. There are multiple dispatching rules to decide the priority between waiting lots. The typical examples of dispatching rules are first in first out (FIFO), shortest processing time (SPT), shortest remaining processing time (SRPT), and fluctuation smoothing policies for mean cycle time (FSMCT), and so on.

In high-mix fabs, the machine groups can be defined in detail for each product, and a machine model can be a member of multiple groups. After finishing a process, wafer lots are loaded to a cassette and transported to another equipment group for a subsequent process. The operator models are reliant upon the human resources to load and unload wafers into machines, to run the machines, and sometimes to perform inspection steps. The modeler can assign different types of operator models, which supervise overall work areas and change the production plans.

Depending on the modeling details of the composed models, various types of events can be employed. A wafer-lot event (\( \ell \)), among various events, is a key event representing
a processing flow and is utilized to exchange wafer lots among machine-group models. The \( e_i \) events can contain the information of the lot name, due date, the number of wafers, a current and required next processing step and the steps’ recipes, and so on.

### B. PROPOSED ABSTRACTION-LEVEL CONVERSION CONCEPT

During the simulation of DE fab models, wafer-lot events (\( e_i \)) flow across multiple machine groups according to the required process sequence in their machines. Concerning the level of detail, process engineers can variously design a machine-group model using subcomponent models, such as a group queue, operators, tools, and so on. The subcomponent models can exchange various types of events, including \( e_i \), for the lot dispatching to available machines, lot loading/unloading process, subchamber operations, and so on.

In real fab, the machine groups of target wafer-fab systems typically start with a transient state. Then, they gradually become a stable state by adjusting the lot-release intervals to prevent the escalation of WIP levels in each group’s queue. The stable groups can also turn into a transient state when the lot arrival-rate changes are caused by the start or completion of a product manufacturing in a high-mix fab, the machines or operators’ scheduled or unexpected status changes, and so on.

When a group operates in a transient state, the queuing performance, such as the time spent in the group, the average number of wafer lots in the group (called a group’s WIP), and the machine utilization, vary abruptly over time; whereas, in a steady state, the average queuing performance of a group reaches stable equilibrium values under the converged rates of input \( e_i \) arrivals.

DE modeling is known as an effective method to reproduce the behaviors of transient and steady-state machine groups. The proposed approach adopts the dynamic partial replacement of an active steady-state group model in runtime, by changing the DE machine-group model to an abstracted MDM, as shown in Fig. 2. The conversion abstracts the detailed operations and event interactions of the group’s subcomponents into a constant delay model. We denote DE machine-group models as low abstraction level models and MDMs as high abstraction level models.

For the detection of a group’s steady state, the queuing parameters of the inter-arrival times (1/\( \lambda \)) and waiting times (\( t_w \)) of the wafer lots are observed. If the observed 1/\( \lambda \) and \( t_w \) distributions of two consecutive samples becomes statistically time-invariant, then we can confirm that the queue length also converges to \( \hat{w} \) based on Little’s law: \( \hat{w} = \lambda \cdot t_w \), where \( \lambda \) and \( t_w \) are the means of the \( \lambda \) and \( t_w \) observations. The detailed detection method of the steady-state condition (which is the convergence of \( t_w \) and 1/\( \lambda \)) is discussed in Sec. III-A.

Following the queuing modeling, the proposed approach considers the overall time spent (or delay, \( t_d \)) in the group as the sum of \( t_w \) and service time (\( t_s \)); a lot’s \( t_w \) is the difference between a group-arrival time (\( t_a \)) at the queue model and a queue-departure time, and the \( t_s \) is a remaining time before leaving the machine group (as \( t_d - t_w \)). The main portion of a lot’s \( t_s \) is a recipe-dependent wafer processing time in a machine. The processing times between wafer lots following the same recipe do not show significant differences. Based on the \( t_s \) property having a small amount of variance, if a group’s \( t_w \) distributions of consecutive samples converge, the overall \( t_d \) is assumed to converge.

The dispatching rules determine the priorities of waiting for lots based on their specific parameters. For example, the FIFO-dispatching rule concerns the arrival time, and the SPT rule deals with the machine’s process time. Generally, the parameters can be classified into (1) constant parameters (which are assigned before simulation) and (2) variable parameters (which change in runtime), as shown in Fig. 3(a). We denote that the lots having the same composite values of parameters related to the production plan and fabrication process are called the same type of lots.

The sub-parameters of the lot type may be different depending on the target dispatching rule; nevertheless, a recipe is an essential sub-parameter of the lot types. In a steady-state group, the \( t_w \) (\( t_w = t_s + t_d \)) distributions can significantly vary depending on the lot type because of the priority difference (affecting \( t_s \)) and recipe difference (involved in \( t_d \)). Thus, a mean delay is derived by the lot type and is differently applied to input lots based on their lot types.

In a steady-state machine group, the \( t_w \) of a new wafer lot is highly related to a relative priority rank and the proportion
of each type in the group queue. To maintain the steady state of $t_w$, the input wafer lots should be time invariant, which requires incoming lots’ $\lambda$ by lot type to vary within some statistical margins of the observed $\lambda$ distribution at the steady-state detection. For the calculation of $1/\lambda$ (or $\lambda$) by the lot type, a lot’s $1/\lambda$ is defined as the difference between the lot’s arrival time and a same-type lot’s previous arrival time. The detailed divergence detection method from a steady-state $1/\lambda$ distribution is described in Sec. III-A. The lot proportions by types in a steady-state machine group are represented as shown in Fig. 3(b).

An event that changes a machine or an operator’s mode (such as scheduled machine maintenance or operator’s rest) influences the group’s lot process capability; in this case, a mean value of $t_d$ at a steady state needs a re-observation. We define the mode-change events (which influence the process capability) as $e_m$. Thus, the overall divergence conditions in a steady-state group are (1) the $1/\lambda$ deviation from a statistical margin of steady-state distribution, (2) the $e_m$ arrival, and (3) an unobserved type of wafer-lot arrival.

C. OVERVIEW OF PROPOSED FRAMEWORK

For the abstraction-level transition, we propose a framework whose structure is illustrated in Fig. 4. In the framework, each machine group consists of a DE machine-group model and an additional pair of an abstraction-level converter (ALC) and a mean-delay model (MDM).

At the low level, the ALC monitors the outgoing $e_\ell$ of its low-level DE group model to extract the queuing parameters for a WIP convergence test. When a steady-state convergence test is passed, the ALC informs its MDM about the observed average $t_d$ of each lot type. At the high level, the ALC monitors the incoming events of $e_\ell$ from the previous machine groups to detect a steady-state divergence condition and relays $e_\ell$ to the MDM. When an abstraction level is converted to the low-level, the ALC generates dummy events of $e_\ell$ to make workload consistent between the two-level models.

For the framework, we assume that the target DE simulator supports output-to-input port connections between models, which represent event-flowing paths. The model connection via port interfaces ensures the modularity, which helps the modeler to construct various target fab systems using modular models. The abstraction level of a group determines $e_\ell$ destination from the previous machine between the low-level group model and the ALC. If the input $e_\ell$ port of a low-level group (or an ALC) is disconnected, the model becomes deactivated because of the absence of future $e_\ell$ events. When the target DE fab model adopts a pull-based lot release policy, if a DE machine group produces feedbacks (as a bottleneck group) to an inventory model, the group’s ALC generates the feedback (as an event, $e_{fb}$) on behalf of the DE group at the high level.

To enable the legacy DE machine-group models to operate with ALCs and MDMs, the DE group models need to be revised to support the following requirements.

- For every $e_\ell$ departing a current machine group, the following information must be generated: $1/\lambda$, $t_w$, and $t_d$ by lot type (see Sec. II-B).
- The ALC and DE machine-group model should share the latest group-arrival time ($t_a$) by type to derive the $1/\lambda$ of next-arriving lot after the abstraction level is changed.
- The DE machine-group model needs to support the dummy-$e_\ell$ processing. The dummy $e_\ell$ has no difference from typical $e_\ell$, except that they are discarded after being processed by machines without being propagated to any other group.
- Some mode-change events ($e_m$) that influence the lot-processing time (such as machine-down/up event) are handled globally based on a certain probability or scheduled plan.
- When a DE machine-group model sends a processing-ended $e_\ell$ to the next machine group, the component sends $e_\ell$ through the event’s output port, without indicating event’s destination explicitly. The simulation engine determines the $e_\ell$’s destination based on the output-to-input port connection information. The ALC can request the engine to revise the connection information in runtime.
- When a DE machine-group model generates the feedback event of $e_{fb}$ to inventory models, the DE group model should provide a query interface for its ALC to observe a state associated with a lot pulling condition (e.g., WIP or workload). At the low-to-high level conversion, the lot-pulling ALC initializes and updates the state to check whether the pulling condition is satisfied and generate the $e_{fb}$ events instead of the low-level model.

When exporting $e_\ell$ to the next machine groups, the MDM should revise $e_\ell$ for the next groups to receive equivalent data of $e_\ell$ (e.g., required processing step) regardless of the sender’s abstraction level. When a bottleneck group is required to generate $e_{fb}$, the group’s ALC checks a lot-pulling condition based on the current working status of both high- and low-level model at the high level. The ALC obtains the low-level model’s current working status using the extended query interface.
When some working (or active) machine groups, which are currently processing lots, share a machine, the abstraction levels of those groups need to be synchronized due to the following example: a shared machine receives the lots from two groups’ queues. If the two working groups’ abstraction levels are different, another low-level group can fully occupy the shared low-level machine because the high-level group no longer provides the lots to the machine.

Considering the dynamic $\ell_1$-flow change and other purposes, such as the abstraction-level synchronization among machine-sharing groups and the queuing-parameter sampling for the convergence and divergence test, we defined the system events for the ALC as follows.

Definition 1: The system events of ALC consist of $e_{pc}$, $e_{as}$, and $e_s$ where

- $e_{pc}$ is an event for the simulation engine to change the output-to-input connection between models,
- $e_{as}$ is an event for the abstraction-level synchronization among machine groups that share machine(s), and
- $e_s$ is an event to schedule the queuing-parameter sampling and steady-state convergence (or divergence) test of machine groups.

The detailed detection mechanisms of steady-state convergence or divergence are described in the following section.

III. CONVERGENCE AND DIVERGENCE CONDITIONS FOR STEADY STATE OF MACHINE GROUPS

A. CONVERGENCE DETECTION TOWARD STEADY STATE AT LOW ABSTRACTION LEVEL

At the low level, ALCs receive outgoing events of $\ell_1$ from their current groups. They extract the information of $1/\lambda$, $\tau_w$, $\tau_d$, and lot-type values from the event. ALCs can collect other information, such as recipe, for the dummy-$\ell_1$ generation. ALCs attempt to detect whether the WIP levels in their machine groups reach a steady state based on the queuing-parameter observations using two consecutive samples.

Typical DE fab models use random variables that follow certain distributions to define model’s operation parameters, such as a wafer-processing time of tool models or wafer-lifting (or break) time of operator models. Thus, each queuing parameter of lots can vary abruptly during simulation, even in a steady state. To avoid the hasty decision of local convergences (or divergences) using the observed queuing-parameter values of individual lots for a short-time period, which can result in unnecessary abstraction-level transitions, we introduce a concept of the $\ell_1$ group, as shown in Fig. 5. ALCs consider arrived events of $\ell_1$ within a defined period as the members of an $\ell_1$ group. Each $\ell_1$ group’s queuing parameters are the averages of individual member lots’ parameters and derived using the following procedures.

The queuing parameters of arrived $\ell_1$ events are stored using a local list ($L_g$). We denote the $1/\lambda$, $\tau_w$, and $\tau_d$ of an $i$-type $\ell_1$ ($e_{\ell_1,i}$) as $1/\lambda_{\ell_1,i}$, $\tau_{w,i}$, and $\tau_{d,i}$, respectively. When an $\ell_1$ arrives at the empty $L_g$, the ALC schedules a sampling task to observe the current $\ell_1$ group’s queuing parameters by sending a delayed event of $e_s$ to itself. When receives $e_s$ at the defined time bound of $\ell_1$ group, the ALC calculates averaged values of $1/\lambda$, $\tau_w$, and $\tau_d$, as well as other parameters: the number of $\ell_1$ arrivals by each lot type, the $\ell_1$-group order, and recipe. The other parameters are utilized to derive the mean delays, generate the dummy events of $e_{\ell_1}$, and prepare for the divergence test at the high-level conversion. After deriving the current $\ell_1$ group’s observations by lot types, the ALC stores the observations in the sample list.

When the number of observed $\ell_1$ groups reaches a defined size, the ALC starts a comparison of the current sample’s $1/\lambda$ and $\tau_w$ distributions with those of the previous sample. In the proposed approach, we employed a two-sample KS test to determine the samples’ statistical similarity in terms of means and variances.

The KS test is nonparametric, making no assumptions about distribution types of $1/\lambda$ and $\tau_w$. For the KS test, the ALC should prepare $1/\lambda$ and $\tau_w$ cumulative distribution functions (CDFs), using the previous and current samples’ observations. For formation into $1/\lambda$ or $\tau_w$ CDFs, each sample’s observations having different lot types should be sorted. To sort the values of $1/\lambda$ (or $\tau_w$) of different lot types, we presume that there is precedence between lot types, and the lot type has higher priority than the queuing parameter.

Given the previous and current samples’ distributions of $1/\lambda_i$ (or $\tau_w_i$), whose sizes are denoted as $n$ and $m$, the KS statistic ($D_{n,m}$) is calculated using the following equation.

$$D_{n,m} = \max_x |F_1(x) - F_2(x)|,$$

where $F_1$ and $F_2$ are the CDFs of two samples. The $D_{n,m}$ means that the maximum difference between two CDFs and has a negative relationship with the $p$-value; if $D_{n,m}$ becomes zero, the $p$-value becomes one, and a $p$-value closing to one means that the two distributions statistically match. In the proposed framework, we need to set the required $p$-value as a guidance of an allowable similarity (e.g., 0.8) of the two consecutive samples.

Based on a given $p$-value and an approximated sample size, which is $[n-m/(n+m)]$, the derivation methods of a maximum bound of $D_{n,m}$ have been studies in [25], [26]. The allowable maximum $D_{n,m}$ is denoted as $d_{max}$. The derived values of $D_{n,m}$ less than $d_{max}$ guarantee the required statistical similarity. For example, when the sizes of consecutive samples of
1/λ are 10 and 11, and the defined p-value is 0.7/0.8/0.9, then the d_\text{max} is calculated as 0.2645/0.2415/0.2115.

To reduce the overhead of the d_\text{max} calculation, the proposed method preloads sufficient candidates of d_\text{max} before the simulation, considering a defined number of monitoring e_\ell groups. For example, if the number of observed e_\ell groups for each sample is 30 and the number of possible lot types is 5, then the maximum n (or m) is 150; it requires 61 values of d_\text{max} for the approximated sample sizes from 15([30 \cdot 30]/(30 + 30)) to 75([150 \cdot 150]/(150 + 150))). The ALC selects a pertinent value among preloaded d_\text{max} values using the approximated size of two consecutive samples in runtime.

After comparing two D_n,s of λ and t_w consecutive distributions with d_\text{max}, the ALC confirms that its machine group runs in steady state. If confirmed, the ALC derives a high-level state (s_h) for the MDM’s operation and dummy-e_\ell generation at the subsequent high-to-low level conversion.

The detailed substates of s_h are defined as follows.

**Definition 2**: The high-level state, s_h, is represented as \{[\hat{t}_d,i], [\hat{t}_w,i], \hat{t}_w, [\hat{\lambda}_i], [\hat{n}_e,i]\}, where
- \{\hat{t}_d,i\} is the set of the average delays by lot type,
- \{\hat{t}_w,i\} is the set of the average waiting times by lot type,
- \hat{t}_w is the high-level conversion (or steady-state detection) time,
- [\hat{\lambda}_i] is the set of average \lambda_i in the steady-state group,
- [\hat{n}_e,i] is the set of the average number of wafer lots in the steady-state group, and
- \{\hat{s}_d,i\} is the set of wafer-processing parameters for the dummy events of \epsilon_{\ell,i}.

At the low-to-high level conversion, the ALC calculates the total averages of \hat{t}_d, \hat{t}_w, and 1/\lambda of a steady \{(\hat{t}_d,i), (\hat{t}_w,i), (\hat{\lambda}_i)\}, using locally averaged values of \hat{t}_d,i, \hat{t}_w,i, \hat{\lambda}_i, and the number of \epsilon_{\ell,i} (as a weighting factor) in each \epsilon_{\ell,i}-group observation. Each \hat{n}_e,i is calculated as \hat{\lambda}_i \times \hat{t}_d,i, based on Little’s law, and can be a floating-point value. Among the s_h, the \{\hat{t}_d,i\} is relayed to the MDM, and the other states are utilized for the dummy-e_\ell generation. The detailed dummy-e_\ell generation method is described in Sec. IV-C.

**B. DIVERGENCE DETECTION AT HIGH ABSTRACTION LEVEL**

This section describes the detailed test method for detecting a 1/λ deviation from a statistical margin, which is one of the divergence conditions (as discussed in Sec. II-B). To maintain a group’s high level, inter-arrival times of e_\ell should remain within the statistical margin of a steady state’s 1/λ distribution observed at the low-to-high level transition.

To check the time-invariant of 1/λ, whenever an e_\ell arrives, the ALC saves the e_\ell’s 1/λ in the local list (L_h) to derive the averaged 1/\lambda_i of a current e_\ell group, as shown in Fig. 6. After probing the 1/\lambda_i of the input e_\ell, the ALC relays the event to its MDM. When receives a sampling event of e_\ell at the time bound of the current e_\ell group, the ALC inserts the current e_\ell-group observations to the current-sample list. Then, the observations of the oldest e_\ell group leave to maintain the defined number of e_\ell groups, as shown in Fig. 6. The defined e_\ell-group number is the same as the number of e_\ell-group sampling for the convergence test at the low level. After updates the current sample, the ALC performs a divergence test by examining the current 1/\lambda_i distribution.

For the divergence test, the ALC manages an additional list (L_s) in runtime, as shown in Fig. 6. This list of L_s is proposed to efficiently measure the CDF distance without redundant comparison between each probability of two consecutive samples’ CDFs in iterative KS tests. The nodes in the L_s are sorted by their 1/\lambda_i values.

During iterative divergence tests, the previous sample, which is the 1/\lambda_i distribution of a steady state as a criterion for determining the current sample, is fixed after the high-level conversion. Whereas, the current sample’s 1/\lambda_i CDF is repeatedly changing whenever new observations occur. We denote the 1/\lambda_i CDF of a steady-state sample as F_\lambda and the other time-varying current sample as F'_\lambda. The proposed method focuses on partial CDF distances ([F_\lambda(1/\lambda_i) - F'_\lambda(1/\lambda_i)]) caused by changed observations instead of comparing all CDF distances.

To trace the CDF-distance change, we define the nodes in L_s as follows.

**Definition 3**: Each node in L_s is represented as (flag, 1/\lambda_i, F_\lambda(1/\lambda_i), c_\ell), where
- flag is the removable condition, flag \in \{true, false\}, and
- c_\ell is the cumulative number of 1/\lambda_i observations in the recent sample.

The L_s’s node is divided into two types: a fixed node containing any 1/\lambda_i of the previous steady-state sample and a removable node including an exclusive \lambda_i value in the current
sample. The current sample’s size \( (m) \) can change when the sizes of the leaving and adding observations are mismatched. Using a node’s \( F_s(1/\lambda_i) \) and \( c_{\ell} \) in \( L_c \), the CDF distance can be calculated as \( |F_s(1/\lambda_i) - c_{\ell}/m| \). The initial list of \( L_c \) at the high-level transition and a node-insertion example are illustrated in Fig. 7(a).

Before testing a divergence, the ALC should prepare a temporal list of \( (\lambda_i, n_t) \), where \( n_t \) means the observation-number change. The tuple list is sorted by \( \lambda_i \) and denoted as \( L_e \). If a \( \lambda_i \) value leaves as an oldest \( e_{\ell} \)-group observation, \( n_t \) is set to \(-1\). If a \( \lambda_i \) value is included in the oldest and newest \( e_{\ell} \) groups, \( n_t \) is zero. However, tuple elements whose \( n_t \) is zero are discarded without being added to the \( L_c \), as shown in Fig. 7(b).

After initializing \( m, d_{\text{max}}, L_c \), and \( L_s \), the detailed procedures to detect a steady-state divergence are described in Alg. 1.

During the divergence test, each node of \( (\lambda_i, n_t) \) in the \( L_c \) is processed one by one. If \( n_t \) of a current node of the \( L_c \) is \(-1\), then the \( c_{\ell} \) values of all the \( L_c \)’s influenced nodes, whose \( 1/\lambda_i \) is equal to or greater than the current \( L_c \) node’s \( 1/\lambda_i \), decrease by one because of the property of cumulative sum. Conversely, when the current \( L_c \) node’s \( n_t \) is \( 1 \), then the \( c_{\ell} \) values of \( L_c \) influenced nodes increase by one. The proposed test method traverses each node in \( L_c \) to revise its \( c_{\ell} \) based on the assembled \( n_t \) influences (\( \Delta c_{\ell} \)). The ALC traverses the \( L_s \)’s nodes to update their \( c_{\ell} \) amounts using the \( \Delta c_{\ell} \).

After updating the \( c_{\ell} \) of a current \( L_s \) node (\( node^0 \)), if the \( node^0 \)'s CDF distance is greater than \( d_{\text{max}} \), the ALC confirms that the machine group turns into a unsteady state. If the \( node^0 \)'s \( c_{\ell} \) (\( c_{\ell}^0 \)) becomes zero and its \( \text{flag} \) is \( \text{true} \), then the node is removed for fast-node traversal at next divergence tests. When any node for a new \( 1/\lambda_i \) value is not found, a new node is added based on the previous node’s \( F_x(1/\lambda_i) \) and \( \lambda_{\ell} \) \( (F_x(1/\lambda_i) \) and \( c_{\ell}^0 \)). If the previous node is not found (when \( node^0 \) is the head node of \( L_s \)), then the \( F_x(1/\lambda_i) \) and \( c_{\ell}^0 \) are zero.

**IV. ABSTRACTION-LEVEL CONVERTER AND DYNAMIC PORT-COUPLING CHANGE**

This section introduces the detailed operation procedures of ALC (including the dummy-\( e_{\ell} \) generation), based on the described steady-state convergence and divergence detection method. We also describe the technique of output-to-input port-connection change between models.

**A. ABSTRACTION-LEVEL CONVERTER**

The ALC, a key component for the dynamic change in the abstraction levels, is defined as follows.

**Definition 4**: The ALC model has primary states of \( s_a \), \( s_b \), \( \{t_{i,\ell}\} \), \( s_{fb} \), initial parameters of \( \{m_{\text{ale}}, t_{\text{idle}}\} \), and \( \psi_{h2l}, \psi_{l2h} \), and two abstraction-level conversion functions of \( \{\psi_{h2l}, \psi_{l2h}\} \), and two system-specific feedback conversion functions of \( \{h, g_b\} \), where

\[ s_a \] is the current abstraction level, \( s_a \in S_a, S_a=\{\text{low, high, ready}\};

\[ s_b \] is the high-level state (defined in Def. 2);

\[ \{t_{i,\ell}\} \] is the set of latest arrival times of \( i \)-type \( e_{\ell} \);

\[ b_i \] is a boolean state for the \( e_{\ell} \)-group sampling, \( b_i \in B_i \), \( B_i=\{\text{true, false}\};

\[ s_{fb} \] is a user-custom state to support a target lot-release policy at the high level;

\[ \{m_{\text{ale}}\} \] is the set of other equipment groups’ ALCs that share a machine with the current group;

\( t_{\text{idle}} \) is a user-defined threshold symbolizing a long-time span;

\( \psi_{l2h}: X \times S_a \rightarrow S_b \times B_i \times S_h \times Y^h_b \) is the low-to-high-level conversion function, \( X \) is the input event set.
of \{e_t, e_m, e_{as}, e_s\}, and \mathcal{S}_b is the set of \mathcal{S}_b; Y_b^p is zero, one, or multiple events of \epsilon_e, \epsilon_{pc}, and \epsilon_{as}.

- \psi_{\text{H2L}}: \mathcal{X} \times \mathcal{S}_d \times \mathcal{S}_b \rightarrow \mathcal{S}_d \times Y_b^p is the high-to-low-level conversion function; \mathcal{Y}_b^p is zero, one, or multiple events of \epsilon_{pc}, \epsilon_{as}, and dummy \epsilon_t;
- \eta_t/\eta_i is a user-custom function to initialize/terminate a lot-pulling operation of bottleneck machine group; and
- \eta_b: \{e_t \times \mathcal{S}_b \rightarrow \mathcal{S}_b \times Y_b^p\} is a user-custom lot-release feedback generator, and \mathcal{Y}_b is zero or one event of \epsilon_b.

At the low level, an ALC’s \mathcal{S}_b is one of low and ready, and the initial \mathcal{S}_b is low. The ready state means that the current group runs in a steady state, while an active ALC in \{m_{alc}\} operates in a low state. The active ALC means that a wafer lot recently arrived in its equipment group and can be determined based on the difference between the current time (t_c) and the latest wafer-lot arrival time (\text{sup} (t_{\text{l}}, i)); the difference is less than \text{tidle}. We denote that an ALC is idle when the difference between \text{t}_c and \text{sup} (t_{\text{l}}, i) is larger than or equal to \text{tidle}. For an ALC to be the high state, its equipment group and others in \{m_{alc}\} should operate in steady states or be idle.

The \mathcal{S}_b is updated when detecting a steady state (as discussed in Sec. III-A). As mentioned in Sec. II-C, low-level machine-group models and their ALCs are required to share \{t_{\text{l}}, i\} for the 1/\lambda derivation, after the abstraction-level change. The \mathcal{B}_b is a state to check whether an \epsilon_e event for the current \epsilon_e-group sampling was previously generated in runtime. The \mathcal{S}_b is a model-specific state to calculate a lot-release condition when the target fab model follows a pull-based lot-release policy. According to the target wafer-fab system and production scenario, the \{m_{alc}\} is initialized as \emptyset or some ACLs of other machine groups sharing any machine with the current group.

When an event arrives at the ALC at the low level, the ALC calls \psi_{\text{H2L}} to handle the input event, which is \epsilon_e, \epsilon_m, \epsilon_{as}, or \epsilon_s. At the high level, the ALC invokes \psi_{\text{H2L}} to serve incoming events of \epsilon_t. At the high-level conversion, when a machine group generates feedback to an inventory model, the machine group’s ALC initializes \mathcal{S}_b by \eta_t to obtain a current working state from the ALC’s low-level model. If a bottleneck machine group follows the CONWIP policy, \eta_t can update the \mathcal{S}_b based on the current WIP level of the machine group. Otherwise, If a bottleneck group adopts the WP policy, \eta_t can initialize the \mathcal{S}_b based on a steady-state workload (that is the sum of remaining processing times of staying lots) of the machine group.

The high-level ALC can produce the feedback events of \epsilon_b to an inventory model through the \eta_b execution, instead of its low-level bottleneck group model. For the CONWIP policy, a bottleneck machine-group model generates an \epsilon_b event when a lot is dispatched to a machine. Likewise, based on an arrived \epsilon_{t,i}, the group’s \eta_b can be designed to generate an \epsilon_b event having the \text{t}_{\text{w}, i} delay (that is the waiting time to be dispatched). For the WR policy, a bottleneck machine-group model generates feedback when the current workload becomes less than a defined threshold. Similarly, the \eta_b for the WR policy can be implemented to predict a future threshold-crossing (FTC) time under the assumption that no further lot events arrive. Then, the \eta_b schedules a delayed event of \epsilon_b for the delivery to a target inventory model at the predicted FTC time. When a lot arrives within the FTC time, the \eta_b nullifies the previously scheduled \epsilon_b and generates a new \epsilon_b. The \mathcal{S}_b should include the information of the predicted TC time and the scheduled \epsilon_b.

At the high-to-low level conversion, the ALC executes \eta_t to cancel any previously scheduled events of \epsilon_b. In the following sections, we explain the detailed procedures of \psi_{\text{H2L}} and \psi_{\text{L2H}}.

B. EVENT-HANDLING PROCEDURES AT LOW-LEVEL

Based on the proposed convergence test, the overall event-handling procedures of \psi_{\text{H2L}} for high-level conversion are described in Alg. 2.

The ALC receives each event through its associated input port. When \mathcal{S}_b is low, if an \epsilon_t arrives, the ALC saves \epsilon_t’s queuing and other related parameters using the local list (L_g). Then, the ALC schedules the operation of \epsilon_t-group sampling at the group’s time bound through sending \epsilon_s to itself, if the sampling is not scheduled. The sampling reservation sets the \mathcal{S}_b as true.

When receives \epsilon_s, the ALC derive the \epsilon_t-group observations and adds the observations to the current sample list. After the insertion at the low state, if the previous and current sample become full (that means that the number of observed \epsilon_t groups in the current sample reaches a user-defined threshold), the ALC performs the KS convergence tests using the 1/\lambda and \text{t}_{\text{w}} CDFs of the two samples.

If a steady state is confirmed, the ALC initializes the \mathcal{S}_b for the mean-delay update and dummy-\epsilon_e generation and \text{L}_g for the next divergence test. The \mathcal{S}_b is updated as high or ready depending other ALCs’ states in \{m_{alc}\}. If all ALCs in \{m_{alc}\} are ready or idle (\text{t}_c - \{t_{\text{l}}, i\} \geq \text{t}_{\text{idle}}), \mathcal{S}_b is updated as high. The ALC sends \epsilon_{as} to the other ALCs in \{m_{alc}\} to synchronize the abstraction level and informs the MDM of \{d_{alc}\}. When the ALC’s machine group is a lot-pulling bottleneck group, the ALC calls its \eta_t to initialize the \mathcal{S}_b.

For the \epsilon_t-path changes for the high-level model activation, it exports \epsilon_{pc} for the simulation engine to revise the destination of \epsilon_t departing from previous machine groups. The detailed coupling-change mechanism is discussed in Sec. 9. For the next divergence test, the ALC maintains the current sample data but clears the previous sample data. If the convergence test is failed, the ALC empties the current sample after copying the current sample to the previous one.

When \mathcal{S}_b is ready, the ALC waits for all other ALCs in \{m_{alc}\} to operate in steady or idle states. The ready ALC executes the 1/\lambda divergence test using the 1/\lambda of the current \epsilon_t group. If the 1/\lambda is diverged, the ALC sets \mathcal{S}_b to low.

When receives \epsilon_{as}, if \mathcal{S}_b is ready, then the ALC changes its abstraction level to high. However, if the ALC’s \mathcal{S}_b is low, the ALC updates the \mathcal{S}_b to high so that the abstraction levels of the ALC and its machine-sharing ACLs are synchronized to
Algorithm 2 The Overall $\psi_{12a}$ Procedures of ALC

1. $L_g$: a local lists to observe the current $e_l$-group queuing parameters
2. if receive $e_l$ then
3.   store the involved parameters of $e_l$ to $L_g$
4.   if $b_s$ is false then
5.     $b_s \leftarrow$ true, then schedule $e_s$ to be delivered to itself at the current $e_l$ group’s time bound
6. else if receive $e_s$ then
7.   add the current $e_l$-group observations (derived using the $L_g$) to the current sample
8. if $s_a$ is low then
9.   if two consecutive samples are full then
10.     execute the KS convergence test using the $1/\lambda$ and $t_w$ CDFs
11.     if converged then
12.       initialize $s_b$ and $L_a$ using the two samples, then clear the prev. sample
13.       if ALCs in $\{m_{alc}\}$ are idle or ready then
14.         $s_a \leftarrow$ high, and send $e_{al}$ to $\{m_{alc}\}$
15.         inform $(\ell_{d,i},i)$ of $s_b$ to MDM
16.       if is a lot-pulling group then
17.         initialize $s_{fb}$ using the $\eta_i$
18.       send $e_{pc}$ to the simulation engine
19. else
20.     $s_a \leftarrow$ ready
21. else
22.     copy the cur. sample to the prev. sample, then clear the cur. sample
23. else // $s_a = \text{idle}$
24.   refresh $L_a$ using the cur. $e_l$ group’s $1/\lambda$
25. test the $1/\lambda$, divergence, if diverged then
26.   $s_a \leftarrow$ low
27. $b_s \leftarrow$ false, then clear $L_g$ // for next sampling
28. else if receive $e_{as}$ then
29.   if $s_a$ is low then // $s_a$ is idle
30.     $s_a \leftarrow$ high, and send $e_{pc}$ to engine
31. else // $s_a = \text{ready}$
32.   turn abstraction level to high as Line No. 14 to 16
33. if $b_s$ is true then cancel $e_s$, and clear $L_g$
34. else if receive $e_m$ then
35.   if $s_a$ is ready then
36.     $s_a \leftarrow$ low
37. clear the two samples and do procedures in Line No. 32

high. If it receives $e_m$, the ALC cleans up the two consecutive samples and clears the scheduled task and data of $e_l$-group sampling after setting the $s_a$ as low.

\[
\begin{array}{c}
\text{Algorithm 2} \\
\text{The Overall $\psi_{12a}$ Procedures of ALC} \\
\end{array}
\]

C. EVENT-HANDLING PROCEDURES AT HIGH ABSTRACTION LEVEL

At the high level, the ALC attempts to evaluate the WIP-level divergence conditions, which are (1) observing the $1/\lambda$ divergence from its steady-state distributions, (2) receiving a new-type $e_t$, (3) receiving $e_m$, and (4) receiving $e_{as}$. If one of the conditions is met, the ALC generates the dummy events of $e_t$ to create a similar workload that corresponds to the group’s steady state before the low-level transition.

At the current simulation time ($t_c$), when a divergence condition is detected, the number of $i$-th priority lots ($n^h_{\ell,i}$) in the group is predicted based on the elapsed time ($t_{ac}$) between $t_c$ and $(t_{i,j})$, as follows.

\[
n^h_{\ell,i} = \max(0, \hat{n}_{\ell,i} - t_{c} - t_{ac} \cdot \hat{\lambda}_i).
\]

In a steady state, to maintain the $n_{\ell,i}$ as the observed $\hat{n}_{\ell,i}$ (in $s_b$), the departure rate should remain statistically similar to $\hat{\lambda}_i$, as shown in Fig. 8(a). We also expect that the $1/\lambda$ values of incoming $e_{\ell,i}$ are similar to the $1/\hat{\lambda}_i$ in the steady state. When a divergence condition is detected at $t_c$, if $t_{e,i}$ (that is the newest $1/\lambda$) is larger than $1/\hat{\lambda}_i$, the $n^h_{\ell,i}$ decreases from $\hat{n}_{\ell,i}$ by the number of times that $1/\hat{\lambda}_i$ has elapsed, as shown in Fig. 8(b). If the $t_{e,i}$ reaches zero, the $n^h_{\ell,i}$ becomes $\hat{n}_{\ell,i}$.

After the high-level transition, an ALC’s DE group model can still have waiting and processing lots. Based on the interval between $t_{ac}$ (in $s_b$) and $t_c$, the approximate number of remaining wafer lots in the low-level machine-group model ($n^l_{\ell,i}$) can be represented as follows.

\[
n^l_{\ell,i} = \max(0, \hat{n}_{\ell,i} - (t_c - t_{ac}) \cdot \hat{\lambda}_i).
\]

Considering the left lots in the low-level model, the overall number of dummy $i$-type $e_t$ ($n_{\ell,i}^{l,d}$) can be represented as $\lfloor n^l_{\ell,i} - n^l_{\ell,i} + 0.5 \rfloor$ (for the integer number), and the total number of dummy $e_t$ is $\sum_i n_{\ell,i}^{l,d}$.

Based on the proposed dummy-event generation method, the overall procedures for $\psi_{12a}$ are described in Alg. 3.

When a low-level idle ALC receives $e_{as}$ from another ALC in $\{m_{alc}\}$, the $e_{as}$-received ALC changes its abstraction level to high to synchronize with other machine-sharing ALCs (see Sec. IV-B). If the high-level idle group receives an $e_t$, the abstraction level of the group becomes low after delaying the $e_t$ to the low-level machine-group model.

When a high-level active ALC receives an $e_t$, if the type of $e_t$ is new, the ALC changes its level through the low-level conversion operations (as Line No. from 8 to 13 in Alg. 3). Otherwise, the ALC save $e_t$’s $1/\lambda$ value in the list of $L_g$ to
Algorithm 3 The Overall $\psi_{h\bar{2}h}$ Procedures of ALC

1. $L_g$: a local lists to observe the current $\ell$-group queuing parameters

2. if receive $e_{\ell}$ then // $\eta$-type $e_{\ell}$

3. if $t_e - \sup(t_{i_l}) > t_{idle}$ then // is idle?

4. $s_a \leftarrow$ low, and send $e_{\ell}$ to DE machine-group model

5. send $e_{as}$ to ALC in $(m_{alc})$ and $e_{pc}$.

6. else if $e_{\ell}$'s $\eta$-type is not found then

7. $s_a \leftarrow$ low, and send dummy $e_{\ell}$ events

8. send $e_{as}$ to ALCs in $(m_{alc})$ and $e_{pc}$

9. clear the current sample

10. if is a lot-pulling group then call the $\eta_l$

11. if $b_s$ is true then

12. $b_s \leftarrow$ false, and clear $e_s$ and $L_g$

else

13. $t_{i_l} \leftarrow t_e$, and add the $1/\lambda_l$ value of $e_{\ell,i}$ to $L_g$

14. if $b_s$ is false then

15. $b_s \leftarrow$ true, then schedule $e_{\ell}$ to be delivered to itself at the $e_{\ell}$ group’s time bound

16. if is a lot-pulling group then

17. execute $\eta_{lb}$ using $e_{\ell,i}$ and $s_{lb}$

18. relay $e_{\ell}$ to the MDM

else if receive $e_s$ then

19. update the $L_s$ and current sample using $e_{\ell}$-group $1/\lambda_l$

20. execute the $1/\lambda_l$ divergence test

if diverged then

21. turn abstraction level to low as Line No. 8 to 11

else if receive $e_{as}/e_{pc}$ then

22. turn abstraction level to low as Line No. 8 to 13

derive the current $\ell$-group’s $1/\lambda$ observations. Then, the ALC schedules the $e_s$ delivery to itself at the time bound of the current $\ell$ group for $1/\lambda$ sampling and a divergence test, and relays the arrived $e_{\ell}$ to its MDM.

When the ALC’s machine group is a lot-pulling bottleneck group, the ALC executes user-custom $\eta_{lb}$ function to control the lot release. During the $\eta_{lb}$ execution, the ALC references and updates the current working amount ($s_{lb}$) based on the input $e_{\ell,i}$ and can schedules the $e_{lb}$ event for the delivery to inventory models.

When the ALC receives an $e_s$, the ALC updates the $L_s$ (see Sec. III-B) and current sample using the current $\ell$ group’s $1/\lambda$ observations, then executes the proposed divergence test.

When the ALC confirms the $1/\lambda$ divergence or receives an $e_{m}/e_{as}$, the ALC performs the low-level conversion operations. If there exists a future event of $e_s$, then the ALC nullifies the event to be deactivated and clears the list of $L_g$.

D. MEAN-DELAY MODEL AND DYNAMIC WAFER-LOT FLOW CHANGE

At the low-to-high level conversion, the ALC informs the MDM of $(t_{ld,i})$. Instead of a detailed simulation of wafer-fabrication processing, the ALC relays input $e_{\ell}$ events to the MDM. The MDM imitate the data of relayed $e_{\ell}$ events to be indistinguishable from output $e_{\ell}$ events of corresponding low-level model. Then, the MDM exports the $e_{\ell}$ to the next group after manipulating the event’s arrival time so that the next group will receive the event after the mean delay.

In Sec. II-C, we assume that the target DE simulator supports connection (or coupling) between the model’s output and input ports. When a model exports an event through an output, the event is scheduled in the engine’s event-list and waits for its turn to be processed, as shown in Fig. 9. The simulation engine delivers the first event in the event list to its destination models to influence the models. For the delivery, the engine characterizes the event’s destinations based on the output-port terminals of the event’s source model, and the output port’s terminals are managed by a map consisting of terminal lists for each output port. From the implementation perspective, when an output-to-input coupling changes, the terminal lists of the output port must be changed accordingly.

In the proposed framework, we employ a coupling handler, which is a loadable engine module, before the start of simulation and can access the terminal lists of the event-source model during runtime. When an $e_{pc}$ is ready to be processed, the engine delivers the event to the coupling handler to achieve the coupling-change requirements of the $e_{pc}$. The $e_{pc}$ has a higher priority than other types of events, which leads it to be processed preferentially, even in other events exist in the event list.

The low-to-high level conversion of an ALC enables the $e_{\ell}$ and $e_{pc}$ output of its low-level machine-group model to be disconnected. In contrast, the $e_{\ell}$ destinations of the previous machine groups (that handle the last processing step of the current group) are switched from the low-level group model to the ALC. The destinations of ALC’s $e_{lb}$ output are reset to inventory models.

To change the $e_{\ell}$- and $e_{pc}$-path at the high-level conversion, when the coupling handler receives an $e_{pc}$ event from an ALC, the handler updates the terminal lists of related models’ $e_{\ell}$ output ports by

- removing the ALC’s low-level machine-group model from the terminal lists of the previous groups’ $e_{\ell}$ outputs,
- adding the ALC to the terminal lists of the previous groups’ $e_{\ell}$ outputs,
- removing the ALC from the terminal list of its current group’s $e_{\ell}$ output,
- removing the low-level machine-group model from the terminal lists of the previous groups’ $e_{\ell}$ outputs, and
receives an event related to models' maps.

FIGURE 10. Event delivery through the output port based on the terminal map.

For the low-level conversion, when the coupling handler receives an event, the handler restores the terminal lists of related models' outputs to the previous lists before the high-level transition.

V. EXPERIMENTATION

We designed a test model (denoted as the case-1 model) to manifest the relationship between the WIP convergence and the KS-test results. The structure of the fab model is illustrated in Fig. 10. There are five machine groups, each having a wafer-plot queue, a machine, and an operator. The inventory model supplies wafer lots to the first machine group, EG_A, for the production of two different products. Each product requires a different routing path: the order of EG_A, EG_B, EG_C, and EG_D, or the sequence of EG_A, EG_D, EG_C, and EG_E. Each group has a single machine. The machine of EG_C is a batch-processing unit representing a furnace and has a relatively long processing time. We applied two lot-release policies to the case-1 model: uniform job release and CONWIP.

Under the defined processing time of each machine, when supplying the lot at the uniform rate, a high-rate release from an inventory enables the EG_A and EG_C groups to operate in an unsteady state, in which their group queue lengths increase monotonically (see Fig. 11(a)). In contrast, the EG_B, EG_D, and EG_E groups operate in a steady state. For the KS-test-based convergence check, the number of each sample’s probing event is defined as 30, and the time period of each event group is 20 minutes. The defined p-value for the convergence test results than the values of other steady groups, as shown in Fig. 11(c). The other groups are approved as steady-state groups at an early simulation stage, and each model of the groups mainly operates at the high abstraction level. After this approval, the KS test of EG_B is no longer executed until a low-level transition occurs caused by scheduled machine maintenance (for one hour). During the machine maintenance, due to a number of pending lots in EG_C’s queue, EG_C supplies the lots to EG_D and EG_E in the same pattern as before, and EG_C’ queue length is reduced. The 1/λ of each group’s input events follows a Poisson distribution having an average interval, so the values of D_n,m of the 1/λ-convergence-test results are relatively smaller than those of t_w-test results, as shown in Fig. 11(b)(c).

When releasing wafer lots based on the CONWIP policy, the inventory model generates lots based on the required WIP level (defined as 12) of the bottleneck group of EG_C. During the simulation, the overall queue lengths vary within convergent regions, as shown in Fig. 12(a). After ten simulation days, all machine group models except EG_C operate at the high level until the scheduled tool down occurs, as displayed in Fig. 12(b) and (c). The EG_C gathers sample observations after the first test failure (around 19 days), and its high-level model is active after 44 days.

We designed another case (denoted as the case-2 model) employing a majority of machines to evaluate the simulation speedup and accuracy change of the proposed approach, as shown in Fig. 13. The model has four types of machine groups for general process steps: deposition, patterning, etching, and chemical-mechanical polishing. The detailed sub-processes in each machine can differ depending on the lot’s product recipe. The machines have different values of processing times according to the recipe and their stochastic properties. We referenced the timing parameters described in James’s book [27]. There are five types of products, each requiring a different number of mask layers (such as 4, 8, 12, 16, or 20). As the required layer number increases, the number of times that the previous group must be revisited increases (e.g., deposition -> patterning -> etching -> deposition ...).

In these experiments, we consider different types of lot-release policies and dispatching rules, as follows.

- Lot-release policies: uniform release, CONWIP, and WR.
- Dispatching rules: FIFO, shortest processing time (SPT), shortest remaining processing time (SRPT), and earliest due date (EDD).

The inventory models following the uniform lot-release policy produce blank wafer lots at defined periods with stochastic variations. In the given fab model, the first
deposition machine group is the critical bottleneck group, so the first group monitors the WIP levels of each product for the CONWIP or the workload for the WR policy. During the CONWIP simulation, considering the re-entrance, the bottleneck group requests the lot release when a dispatched lot no longer visits the group. In the WR-policy simulation, the bottleneck group measures the current workload considering the remaining processing times at the future reentrances to prevent the WIP-level divergence.

The machine-group queue model has specific comparator functions for target dispatching rules; the priority group queues contain sorted wafer lots using the comparator function corresponding to an employing dispatching rule. Depending on the recipe, the lot experiences a different processing time in the machines, and each product has a different due date.

To evaluate the accuracy of the proposed method, we defined the accuracy index as the average of each lot’s cycle-time difference between low-level-only and multi-level simulations. A lot’s cycle time is the difference between the generation time and the completion time of all steps of wafer processing. During the experiments, we utilized the same sampling parameters and $p$-value as the case-1 model experiments. We measured the simulation execution times using an experimental machine with an Intel® Xeon® E5-1650 3.6GHz CPU and 32 GB of memory.

The overall simulation results of the uniform release policy at various injection rates are displayed in Fig. 14. The simulation results include the execution time, high-abstraction-level duration ratio of groups (over simulation time), ALC execution time at the low and high abstraction level, number of the abstraction level changes, number of generated dummy $e_l$ events and overall speedup of multi-level simulation (compared to the low-level only simulation). The main overhead of the ALC execution at the low level is the lot sampling and convergence test. The high-level ALC computation overhead is the sum of processing wafer lots based on observed mean delays, divergence test, and additional lot-pulling executions for the CONWIP and WR policy.

Fig. 14(a) shows the simulation results of the uniform lot release at different rates, which produces five wafer lots with exponential distributions having average intervals of 150, 200, 250, and 300 minutes. As the release rate decreases (or the lot-release interval increases), WIP levels of all machine groups are more likely to vary in the converged regions. This helps the machine groups to be confirmed as steady-state groups more easily. The easier steady-state confirmation increases the high-level duration ratio and the execution time of high-level ALC. Conversely, the execution time of low-level ALCs decreases. The overhead of the low-level ALC’s sampling and test tasks does not significantly degrade the simulation performance due to its small amounts (that distributed from 7.5 to 32 milliseconds).
When the five-lot release intervals are over 200 for the 4000/5000/6000-lot productions, the overall speedups are distributed from 2.47 to 3.04. When the uniform lot-release rate is 5/130 (whose unit is the lot number per minute [1/min]) for the 5000- and 6000-lot productions, the proposed multi-level simulation can be slower than the low-level-only simulation because of an increased overhead of priority-queue models in highly congested groups.

Even though a particular group’s WIP increases or decreases macroscopically, a local WIP convergence can make the group operate at the high level for a short time. When switching back to the low level, a low-level queue model can execute the comparisons between newly-received dummy $e_\ell$ events and remaining lots (that can be more than 500 at the 5/130 rate). The additional queue overhead results in worse simulation performance of the mixed-level simulation than low-level only simulation. However, the high-injection cases are rare because the machine-group queues of general fabs are managed not to be diverged, so corresponding fab models do not encounter the rare cases having extremely large queues.

At the high-release rates from 5/130 to 5/190, the simulation results according to the $p$-value change are illustrated in Fig. 14(b). The smaller $p$-value makes machine groups to be confirmed as steady-state groups more easily. Thus, the overall speedup increases, while the accuracy becomes degraded. The larger $p$-value makes it difficult for unsteady machine groups to operate at the high level. Still, it quickly makes steady-state groups run at the low abstraction level when a small amount of $1/\lambda$ changes occur because of the stochastic property of the target DE model.
The simulation results for various lot-release and dispatching policies are displayed in Fig. 15. The overall speedups vary between 1.76 and 4.08, with an accuracy expense from 5% to 0.6%. Compared to other lot-release policies, the inventory models following the CONWIP policy generate lots in more irregular patterns having larger variation, which results in lower ratios of high-level duration and speedups. In the CONWIP simulation, parts of adaptive abstraction-change examples of the first machine group (EG_DEP) under the irregular lot release are illustrated in Fig. 16. The intervals between the last convergence tests (of $d_{\text{max}}$ of $t_r$ and $1/\lambda$) and dummy-$e_1$ generations are the durations of the high abstraction level.

VI. CONCLUSION

To speed up a discrete-event wafer-fab model, we proposed a two-level modeling and simulation approach that adjusts the abstraction level of machine groups during runtime. The proposed method attempts to simulate the wafer-lot processing in a steady-state machine group based on observed mean-delay information (as a high-level operation) instead of the detailed computation of the low-level DE group model. To detect a steady state of a machine group, the ALC collects the queuing-model parameter values from its low-level machine group’s outgoing wafer-lot events. When the number of parameter sampling reaches a defined count, the ALC checks whether the arrival-rate and waiting-time distributions are statistically indistinguishable. If the queuing-parameter distributions of consecutive samples become converged, then the high-level model handles input wafer lots based on observed mean spent time. At the high level, the ALC analyzes the inter-arrival time of input wafer-lot event deviates from the observed distribution or monitors the arrival of an operation-status-change event (such as machine down). If a WIP divergence condition is satisfied, the ALC reactivates the low-level model after generating the dummy events to ensure a workload consistency between low-level and high-level models. The proposed approach shows speedup up to 4.08 times when the fab is managed to be stable, with 0.6 to 8.3% accuracy reduction under given scenarios.

REFERENCES

[1] D. Kopp, L. Mönch, D. Pabst, and M. Stehli, “Qualification management in wafer fabs: Optimization approach and simulation-based performance assessment,” IEEE Trans. Autom. Sci. Eng., vol. 17, no. 1, pp. 475–489, Jan. 2020.

[2] B. W. Hsieh, C. H. Chen, and S. C. Chang, “Efficient simulation-based composition of scheduling policies by integrating ordinal optimization with design of experiment,” IEEE Trans. Autom. Sci. Eng., vol. 4, no. 4, pp. 533–568, Oct. 2007.

[3] L. Reinhardt and L. Monch, “Simulation-based optimization to design equipment-health-aware dispatching rules,” in Proc. WSC, Las Vegas, NV, USA, Dec. 2017, pp. 3565–3575.

[4] B. Liu, C. Li, F. Yang, H. Wan, and R. Uzsoy, “Production planning for semiconductor manufacturing via simulation optimization,” in Proc. WSC, Phoenix, AZ, USA, Dec. 2011, pp. 3612–3622.

[5] D. P. Connors, G. E. Feigin, and D. D. Yao, “A queueing network model for semiconductor manufacturing,” IEEE Trans. Semicond. Manuf., vol. 9, no. 3, pp. 412–427, Aug. 1996.

[6] D. Grosbard, A. Kalir, I. Tirkel, and G. Rabinowitz, “A queueing network model for wafer fabrication using decomposition without aggregation,” in Proc. CASE, Madison, WI, USA, Aug. 2003, pp. 717–722.

[7] M. A. Chik, A. B. Rahim, A. Z. M. Rejab, K. Ibrahim, and U. Hashim, “Discrete event simulation modeling for semiconductor fabrication operation,” in Proc. ICSE, Kuala Lumpur, Malaysia, Aug. 2014, pp. 325–328.

[8] Y.-H. Low, B.-P. Gan, S. Jain, W. Cai, W.-J. Hsu, S.-Y. Huang, and S. J. Turner, “Parallel discrete-event simulation of a supply-chain in semiconductor industry,” in Proc. 4th HPC, Beijing, China, 2000, pp. 1154–1157.

[9] M. Seok and T. G. Kim, “Parallel discrete event simulation for DEVS cellular models using a GPU,” in Proc. SpringSim, Orlando, FL, USA, 2012, pp. 26–30.

[10] T. Hildebrandt, D. Gowsami, and M. Freitag, “Large-scale simulation-based optimization of semiconductor dispatching rules,” in Proc. WSC, Savannah, GA, USA, Dec. 2014, pp. 2580–2590.

[11] Y. Ma, F. Qiao, W. Yu, and J. Lu, “Parallel simulation-based optimization on scheduling of a semiconductor manufacturing system,” in Proc. WSC, Savannah, GA, USA, Dec. 2014, pp. 2571–2579.

[12] R. T. Johnson, J. W. Fowler, and G. T. Mackulak, “A discrete event simulation model simplification technique,” in Proc. WSC, Orlando, FL, USA, 2005, p. 5.

[13] O. Rose, “Improved simple simulation models for semiconductor wafer factories,” in Proc. WSC, Washington, DC, USA, Dec. 2007, pp. 1708–1712.

[14] C. P. Veeger, L. F. Etman, E. Lefeber, I. J. B. F. Adan, J. van Herk, and J. E. Rooda, “Predicting cycle time distributions for integrated processing workstations: An aggregate modeling approach,” IEEE Trans. Semicond. Manuf., vol. 24, no. 2, pp. 223–236, May 2011.

[15] J. M. Bekki, J. W. Fowler, G. T. Mackulak, and B. L. Nelson, “Indirect cycle time quantile estimation using the Cornish–Fisher expansion,” IIE Trans., vol. 42, no. 1, pp. 31–44, Oct. 2010.

[16] M. Seok, C. W. Chan, W. Cai, H. S. Sarjoughian, and D. Park, “Runtime abstraction-level conversion of discrete-event wafer-fabrication models for simulation acceleration,” in Proc. SIGSIM-PADS, Miami, FL, USA, Jun. 2020, pp. 83–92.

[17] A. Sarraf Shirazi, T. Davison, S. von Mannen, J. Denzinger, and C. Jacob, “Adaptive agent abstractions to speed up spatial agent-based simulations,” Simul. Model. Pract. Theory, vol. 40, pp. 144–160, Jan. 2014.

[18] R. Franceschini, M. Challenger, A. Ciccetti, J. Denil, and H. Vangheluwe, “Challenges for automation in adaptive abstraction,” in Proc. MODELS-C, Munich, Germany, Sep. 2019, pp. 443–448.

[19] R. Franceschini, S. V. Mierlo, and H. Vangheluwe, “Towards adaptive abstraction in agent based simulation,” in Proc. WSC, National Harbor, MD, USA, Dec. 2019, pp. 2725–2736.

[20] G. Marsaglia, W. W. Tsang, and J. Wang, “Evaluating Kolmogorov’s distribution,” J. Stat. Softw., vol. 8, pp. 1–8, Nov. 2005.

[21] L. Carvalho, “An improved evaluation of Kolmogorov’s distribution,” J. Stat. Softw., vol. 65, pp. 1–4, Jun. 2003.

[22] J. P. Ignizio, Optimizing Factory Performance: Cost-Effective Ways to Achieve Significant and Sustainable Improvement. New York, NY, USA: McGraw-Hill, 2009.

[23] L. Monch, J. W. Fowler, and S. J. Mason, Production Planning and Control for Semiconductor Wafer Fabrication Facilities: Modeling, Analysis, and Systems. New York, NY, USA: Springer, 2012.
MOON GI SEOK (Member, IEEE) received the B.S. degree in electronics engineering from Korea University, Seoul, South Korea, in 2009, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2011 and 2017, respectively. He was a Postdoctoral Researcher at the KAIST and Arizona State University (ASU), Tempe, AZ, USA, from 2018 to 2019. He is currently a Research Fellow of Nanyang Technological University, Singapore. His research interests include multi-level modeling, simulation, and the verification of system-on-chip (SoC) designs, and high-performance simulation methodology in various domains.

HESSAM S. SARJOUGHIAN is an Associate Professor of computer science and computer engineering with the School of Computing, Informatics, and Decision Systems Engineering (CIDSE), Arizona State University (ASU), Tempe, AZ, USA, and the Co-Director of the Arizona Center for Integrative Modeling and Simulation (ACIMS). His research interests include model theory, poly-formalism modeling, collaborative modeling, simulation for complexity science, and modeling and simulation frameworks/tools. He is the Director of the ASU Online Master’s of Engineering in Modeling and Simulation Program.

WENTONG CAI (Member, IEEE) is a Professor with the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU), Singapore. He is a member of the ACM. His research interests include modeling and simulation, and parallel and distributed computing. He is an Associate Editor of the ACM Transactions on Modeling and Computer Simulation (TOMACS), an Editor of the Future Generation Computer Systems (FGCS), and an Editorial Board Member of the Journal of Simulation (JOS).

DAEJIN PARK (Member, IEEE) received the B.S. degree in electronics engineering from Kyungpook National University, Daegu, South Korea, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea, in 2003 and 2014, respectively. He was a Research Engineer at major semiconductor companies, such as SK Hynix Semiconductor, Samsung Electronics, and ABOV Semiconductor from 2003 to 2014, over 12 years, and has worked on processor architecture design and low-power ASIC implementation with custom-designed software algorithm optimization. He is a full-time Professor as a Presidential Research Fellow of the School of Electronics Engineering, Kyungpook National University.

***