Out of Order and Causally Correct: Ready Event Discovery through Data-Dependence Analysis

Erik J. Jensen
ejens005@odu.edu
Old Dominion University
Norfolk, Virginia, USA
Virginia Modeling, Analysis, and Simulation Center
Suffolk, Virginia, USA

James F. Leathrum
jleathru@odu.edu
Old Dominion University
Electrical and Computer Engineering
Norfolk, Virginia, USA

Christopher J. Lynch
cjlynch@odu.edu
Old Dominion University
Norfolk, Virginia, USA
Virginia Modeling, Analysis, and Simulation Center
Suffolk, Virginia, USA

Katherine Smith
k3smith@odu.edu
Old Dominion University
Norfolk, Virginia, USA
Virginia Modeling, Analysis, and Simulation Center
Suffolk, Virginia, USA

Ross Gore
rgore@odu.edu
Old Dominion University
Norfolk, Virginia, USA
Virginia Modeling, Analysis, and Simulation Center
Suffolk, Virginia, USA

Abstract

Data-dependence analysis can identify causally-unordered events in a pending event set. The execution of these events is independent from all other scheduled events, making them ready for execution. These events can be executed out of order or in parallel. This approach may find and utilize more parallelism than spatial-decomposition parallelization methods, which are limited by the number of subdomains and by synchronization methods. This work provides formal definitions that use data-dependence analysis to find causally-unordered events and uses these definitions to measure parallelism in several discrete-event simulation models.

A variant of the event-graph formalism is proposed, which assists with identifying ready events, by more clearly visualizing data dependencies between event types. Data dependencies between two event types may be direct or indirect, where the latter case considers the scheduling of intermediate events. Data dependencies and scheduling dependencies in a discrete-event simulation model are used to define time-interval limits that support the identification of events that are ready for execution. Experimental results from serial simulation testing demonstrate the availability of numerous events that are ready for execution, depending on model characteristics. The mean size of the ready-event set varies from about 1.5 to 110 for the tested models, depending on the model type, the size of the model, and delay distribution parameters. These findings support future work to develop a parallel capability to dynamically identify and execute ready events in a multi-threaded environment.

SIGSIM PADS '25, June 23–26, 2025, Santa Fe, NM

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

CCS Concepts

• **Computing methodologies** → **Discrete-event simulation**; *Massively parallel and high-performance simulations.*

Keywords

Discrete-event simulation, data-dependence analysis, event-level parallelism, independent events, available parallelism, causal ordering, spatial decomposition, event graphs, PDES

ACM Reference Format:
Erik J. Jensen, James F. Leathrum, Christopher J. Lynch, Katherine Smith, and Ross Gore. 2018. Out of Order and Causally Correct: Ready Event Discovery through Data-Dependence Analysis. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (SIGSIM PADS '25)*. ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Discrete-event simulation (DES) is widely used to simulate systems like manufacturing [13], supply-chain [27], transportation [31], computing [21, 43, 44], and telecommunication [28] networks, where performance improvements can lead to quicker insights into system behavior and design optimization. Parallel discrete-event simulation (PDES) typically utilizes spatial decomposition (SD) of models into distinct logical processes (LPs), where each LP is a serial simulation that may communicate with other LPs, to synchronize in order to avoid causality errors [4, 8, 14, 18]. In SD parallelization, the number of events available for parallel execution is limited by the number of LPs, where the maximum number of events any LP can execute concurrently is one. The number of LPs may be limited by the decomposability of the model. An LP in an SD system assumes events in its pending-event set are totally-ordered according to timestamp (TS), meaning that beyond the first event in the pending-event set, there is no capability to find additional events that are ready for execution within an LP [23].

While LP-level SD parallelization is the typical approach, DES parallelization can take several forms, depending on which level of the simulation is parallelized. Table 1 summarizes DES parallelism levels and highlights event-level parallelism, which is the focus of this work. In the case of event-level parallelism, multiple events in an LP's pending-event set may be executed simultaneously. This work builds the foundation for a DES execution method which may be used to achieve event-level parallelism for general-purpose DES.

Simulation Parallelism Level	Description
Simulation-level	multiple independent simulation applications
LP-level	multiple serial simulation processes/threads in a simulation application
Event-level	multiple events in a multi-threaded simulation process
Event-routine-level	multiple tasks in the execution of an event, in a multi-threaded simulation process

Table 1: Parallelism granularity in PDES, highlighting event-level parallelism.

To justify the viability of such a method, this work seeks to identify the available event-level parallelism in a DES model, using data-dependence analysis (DDA) [1, 30]. Instead of assuming that events in an LP's pending-event set are totally ordered by timestamp (TS), as is typical in SD parallelization methods, DDA is used in this work to discover causal ordering in any partially-ordered event set. If the event set is partially ordered, only some events impose ordering, and other events exist in a causally-unordered state. To explain the utility of causal ordering, as opposed to timestamp (TS) ordering, fig. 1 takes a backward-looking approach, using a completed event-execution sequence to visualize how causally-unordered events may be executed OoO or in parallel. The event-execution sequence in fig. 1 is partially ordered. Causally-unordered events are grouped by a non-white color and are contiguous in this simplistic example. All events in a group are ready to execute when all events preceding the group have finished executing. In each sequence in fig. 1, within any grouping of causally-unordered events, e.g. the set {*c*,*d*,*e*}, each event in each pair of events {*c*,*d*}, {*c*,*e*}, and {*d*,*e*} is *mutually independent* from the other event in the pair. In general, mutual independence does not imply that two events are totally causally unordered, meaning, independent from all other events in a set.

In this work, DDA is used in a forward-looking approach, to identify those causally-unordered events in an LP's pending-event set that may be executed out of order (OoO) and are ready for execution. Henceforth, these events are referred to as ready events (REs). State variable (SV) usage, event timestamps, and new-event scheduling are considered when finding these REs.

The event graph (EG) formalism depicts state changes within each event and scheduling dependencies between events [5, 12, 34, 35]. Any DES model that may be represented as an event graph is a candidate for parallelization through DDA. From the information in an event graph, a lookup table may be generated in pre-processing and used during a data-dependence-analysis-discrete-event simulation (DDA-DES) to identify events that are ready for execution. For DDA-DES, it is not necessary to identify an optimal decomposition strategy, which considers load balancing and communication efficiency, as is the case for traditional spatial-decomposition methods [17, 32]. The significant contributions of this work are:

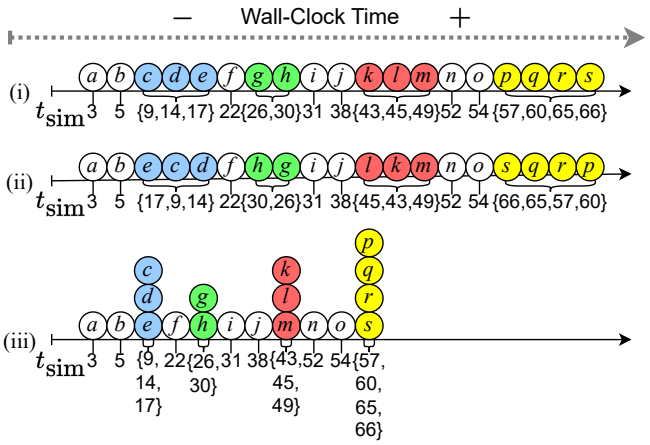


Figure 1: Three partially-ordered event-execution sequences, for a simulation with events a through s. Sets of brightly-colored causally-unordered events like $\{c, d, e\}$ may be executed out of order ((ii)) or in parallel ((iii)). Wall-clock time increases from left to right.

- a strategy for identifying events that are ready to execute within an LP's pending-event set, based on pairwise mutual independence, including mathematical definitions of:
  (1) direct data dependency;
  (2) indirect data dependency;
  (3) event conflict, which determines mutual independence;
  (4) a time-interval limit that quantifies mutual independence;
  (5) the set of ready events (REs), which may be executed OoO or in parallel without causality violation;
  (6) OoO simulation divergence from the in-order simulation;
  (7) divergence potential, given an OoO event;
- a variant of the event graph formalism, which adapts model data for DDA-DES and visualizes data dependencies;
- an algorithm for measuring parallelism in a DES model, and
- experimental results that quantify the available parallelism for multiple DES models.

The remainder of this paper is organized such that related work is discussed in section 2, the essential concepts underlying DDA-DES are explained in section 3, experimental results are in section 4, and discussion about the findings and the direction of future work are in section 5.

2 Related Work

Prior work on event-level DES parallelization has explored approaches such as parallelizing scheduling [19, 20, 41], leveraging

data-dependency analysis for specialized simulations [9–11, 26], and speculative execution using heuristics [22]. However, these methods are either domain-specific or lack support for general-purpose DES models with parallelizable state-variable updates.

2.1 Event-Level DES Parallelization

Prior works have presented ideas related to event-level-DES parallelization, which may be able to parallelize scheduling [19, 20], or prevent unnecessary rollbacks in an optimistic LP [33], or use data dependencies to parallelize specialized types of discrete-event simulations [9–11, 26], or speculatively execute events in parallel while learning simulation behavior [22], e.g. But, prior to this work, there is no work that approaches event-level DES parallelization for general-purpose simulation models, where safe events can be identified definitively and SV updates can be parallelized. Further, the authors have found no prior work that measures the available parallelism during the simulation of a DES model, based on event independence. Note, this differs from critical-path analysis (CPA), which may be used to bound simulation runtime [3, 38, 39]. However, CPA is not a parallelization technique and is not designed to identify independent events within a pending-event set during the simulation of a DES model.

Jones [19, 20] proposed an early alternative called "concurrent simulation", which finds parallelization through knowledge of scheduling behavior. There is no capability to parallelize SV updating, but scheduling may be parallelized. The design utilizes a linked-list event set, which would not be useful for large models with large event sets. The only experimental results presented are from a small simulation model.

Soliman [37] demonstrated a practical application of data-dependence analysis by running parallel simulations with precomputed sets of independent events that are generated from a completed event-execution sequence. Independent events are identified using predefined state-dependency rules. This approach has no capability to automatically parallelize a running simulation in which the event-execution sequence is unknown.

Quaglia and Baldoni [33] introduced the idea of weak causality (WEC), which suggests how causally-unordered events in an event set might be identified, using SV-access analysis. Given an earlier event and a later event in an event set, WEC can compare these two events, to find immediate SV-access conflicts. However, though it is inferred that scheduling is a necessary consideration, there is no mechanism for comparing intermediate events, that the earlier event might schedule, with the later event. The proposed use-case for WEC is in an optimistic simulator LP, where unnecessary rollbacks may be prevented if straggler events are independent from later events that have already executed. This backward-looking approach is appropriate for WEC, which has no capability in its published form to find independent events in a forward-looking manner, in a pending-event set. It is not a DES parallelization technique. No experimental results are provided.

Chen et al. [10, 11] introduced an OoO PDES simulator for the SystemC and SpecC system-level description languages [15, 16], leveraging data-dependence analysis between events to execute them out of order. By statically analyzing event code, they identify independent event-code segments and find parallelism within and across simulation cycles. An extension [9] introduced prediction mechanisms to further reduce conflicts and increase the number of parallel tasks, achieving substantial speedup in computationally intensive but simple and predictable applications like media encoding and decoding.

Kunz et al. [22] experimented with probabilistic heuristics to detect causality relationships. A centralized scheduler predicts if the next event in the event set can safely execute, given that worker threads may still be executing previously-distributed events. The system state rolls back if causality is violated. This approach appears to be useful only for long-running simulations, as it requires a training phase and an adaptation phase before reaching a steady-state phase. Further, it is unclear if this approach is useful beyond the wireless mesh-network model used in this work and may be successful for general-purpose DES.

Liu and Wainer [26] developed a technique for parallelizing P-DEVS and Cell-DEVS models using multicore processors, combining multiple levels of parallelism. Their work exploits SIMD-like patterns found in these models, although this approach is limited to models that utilize these easily-predictable patterns.

Wang et al. [42] converted the ROSS simulator [6] into a task-based framework (ROSS-TB) for multigrained PDES, combining both LP-level and event-level parallelism. A coordinator thread distributes ready events to team threads, where ready events are those with the same timestamp or otherwise identified as causally unordered within a simulation time window. This method is tested using a four-node cluster and an airports model, demonstrating effective load balancing and scalability through dynamic task distribution. The process that identifies independent events is unclear.

2.2 Event Graphs

The event graph (EG) formalism models DES systems by representing state changes as vertices and scheduling as directed edges connecting vertices [35]. State changes and scheduling may be conditional or non-conditional. At each vertex, conditions are evaluated only once per event execution, before updating SVs or scheduling vertices [36]. An edge may indicate a non-zero scheduling delay time as a number or variable at the tail of the edge, and conditional scheduling with an "S" symbol on the edge and the condition.

Several extensions to the event graph (EG) formalism have been proposed, including parameterization, which enables the passing of arguments from a scheduling (or originating) vertex to a scheduled (or terminating) vertex [5, 34]. Parameterization simplifies repetitive models by collapsing multiple instances of similar subgraphs into a single parameterized grouping of vertices. Specific subgraphs are accessed dynamically by passing index values as arguments along the edges connecting vertices.

3 Data-Dependence-Analysis-DES

In this work, data-dependence and scheduling analyses are used to find causally-unordered events in a general discrete-event simulation, for which event routines that contain SV-updating and scheduling behavior are clearly defined, including minimum scheduling delays. This data may be presented in an event graph, for the benefit of visualization. DDA-DES selects events from an LP's event set that are ready for OoO and conservative execution.

A pending event is considered ready to execute if it is mutually independent from all events ahead of it in the event set. To determine if the k^{th} event in the event set **E** is ready to execute, event k must be compared with each event in **E** preceding k. Up to $k-1$ pairwise evaluations are made, where each evaluation determines if the pair of events is mutually independent. If each pair of events (j,k) is mutually independent, event k is causally unordered and ready to execute. Section 3.1 explains how to determine mutual independence and how to define the set of ready events using formal definitions, section 3.1.1 provides a mathematical framework to DDA-DES, and section 3.2 introduces a variant of the event graph modeling formalism that benefits DDA-DES by abstracting away low-level information and highlighting data dependencies.

3.1 Identifying Ready Events

At a high level, ready events (REs) are defined as those events in the TS-ordered pending-event set **E**, which will not be affected by preceding events in **E** and cannot affect preceding events in **E**, if executed OoO. These events may execute immediately. Readiness for the k^{th} event in **E** requires mutual independence for each (j,k) pair in **E**, where $j \in \{1,\ldots,k-1\}$, for $k>1$. The determination that two events in a pair are mutually independent requires knowledge of data dependencies and timing.

Consider a simulation with a non-empty pending-event set **E**, where each event is a specific instance of an associated event type. An event type defines the event routine, which determines state-variable updates and scheduling of new events. An event contains an event type and a TS that determines when the associated event routine executes. Each event in **E** has an event type V_i^G with a set of SVs $\mathbf{S}_i$ that are read or written during event execution, where the simulation model has n different event types, and $i \in \{1,\ldots,n\}$. There are three different ways in which event routine i may use SVs in $\mathbf{S}_i$. For each state variable $s_{i,\ell}$ in $\mathbf{S}_i$, where $\mathbf{S}_i$ has n_i state variables, and $\ell \in \{1,\ldots,n_i\}$, event routine i may

(1) update $s_{i,\ell}$,
(2) read $s_{i,\ell}$ to update an SV in $\mathbf{S}_i$, or
(3) read $s_{i,\ell}$ to evaluate a condition, which determines conditional SV updating in $s_{i,\ell}$ or conditional event scheduling.

Case-(1) SVs are *output* state variables, meaning they are written to memory, and case-(2) and case-(3) SVs are *input* state variables, meaning they are read from memory. Within $\mathbf{S}_i$, a state variable may be both an input SV and an output SV. SVs in $\mathbf{S}_i$ are not necessarily exclusive to $\mathbf{S}_i$. Figure 2 visualizes input and output SVs for `Station-`j in a tandem n-queue network. In each station, the arrive and depart vertices both read from and write to the queue and server variables Q_j and P_j, meaning Q_j and P_j are input and output SVs in `Arrive-`j and `Depart-`j. Note, the `Station-`j EG may be reduced to remove the `Process-`j vertex, according to the rules in [35].

Consider that for two event types A and B, it may be true that $\mathbf{S}_\mathsf{A} \cap \mathbf{S}_\mathsf{B} \neq \emptyset$. [1] This is demonstrated in fig. 2, where for each station j, $\mathbf{S}_{\mathtt{Arrive-j}} \cap \mathbf{S}_{\mathtt{Depart-j}} \neq \emptyset$. If $\mathbf{S}_{\mathsf{A}\cap\mathsf{B}}$ contains SVs that are output SVs for either event type, there is a *data dependency* between these event types. To be more specific, this case of $\mathbf{S}_{\mathsf{A}\cap\mathsf{B}} \neq \emptyset$, where the

[1] To maintain consistency with the i notation, (i.e., event type i, $\mathbf{S}_i$, and $s_{i,\ell}$) named event types may map to integers in $\{1,\ldots,n\}$, e.g. A $\mapsto$ i=1, V_1, and B $\mapsto$ i=2, V_2.

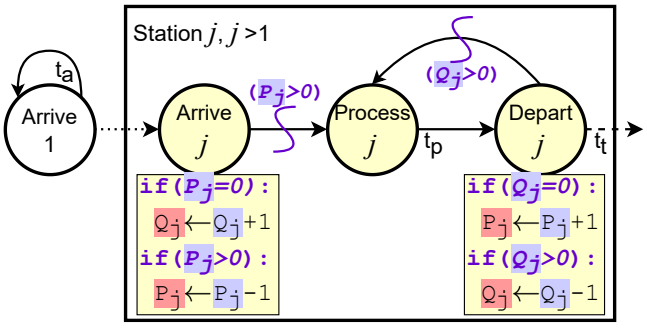


Figure 2: EG of an `Arrive`→`Process`→`Depart` station in an n-queue tandem network, with blue input SVs and red output SVs. Dotted line indicates abbreviated link between `Arrive-1` and `Arrive-`j. Dashed line indicates possible edge between `Depart-`j and `Arrive-`j+1.

intersection set contains an output SV, is a *direct* data dependency (DDD). In fig. 2 there is a direct data dependency between `Arrive-`j and `Depart-`j, which imposes causal ordering on events of these types. Definition 3.1 gives the mathematical definition of DDD, for two event types A and B.

Definition 3.1 (Direct Data Dependency (DDD)).

For two event types A and B :

$$\mathsf{A} \overset{\text{DDD}}{\longleftrightarrow} \mathsf{B} \iff \mathbf{S}_\mathsf{A} \cap \mathbf{S}_\mathsf{B} = \mathbf{S}_{\mathsf{A}\cap\mathsf{B}} \neq \emptyset \ \wedge \ \left(\mathbf{S}_{\mathsf{A}\cap\mathsf{B}} \cap \mathbf{S}_{\mathsf{A}_O} \neq \emptyset \ \vee \ \mathbf{S}_{\mathsf{A}\cap\mathsf{B}} \cap \mathbf{S}_{\mathsf{B}_O} \neq \emptyset\right). \tag{1}$$

Definition 3.2 (Indirect Data Dependency (IDD)).

For three event types A, A+, and B :

$$\mathsf{A} \overset{\text{IDD}}{\longleftrightarrow} \mathsf{B} \iff \exists \mathsf{A}+ \mid \left(\mathsf{A} \overset{\text{reach}}{\longrightarrow} \mathsf{A}+ \ \wedge \ \mathbf{S}_{\mathsf{A}+} \cap \mathbf{S}_\mathsf{B} = \mathbf{S}_{\mathsf{A}+\cap\mathsf{B}} \neq \emptyset \ \wedge \ \left(\mathbf{S}_{\mathsf{A}+\cap\mathsf{B}} \cap \mathbf{S}_{\mathsf{A}+_O} \neq \emptyset \ \vee \ \mathbf{S}_{\mathsf{A}+\cap\mathsf{B}} \cap \mathbf{S}_{\mathsf{B}_O} \neq \emptyset\right)\right). \tag{2}$$

Data dependencies may also be *indirect*. To identify if two event types have indirect data dependencies, the scheduling capabilities of the model are significant. Consider that for two event types A and B, it may be true that there exists a scheduling path from A to an event type A+. That is, A can *reach* A+, either immediately or intermediately through proxy event types. In fig. 2, any event type in `Station-`j_1 can reach any event type in `Station-`j_2, $j_1<j_2$. In the case that A can reach A+, it may be true that $\mathbf{S}_{\mathsf{A}+} \cap \mathbf{S}_\mathsf{B} \neq \emptyset$. If $\mathbf{S}_{\mathsf{A}+\cap\mathsf{B}}$ contains SVs that are output SVs for either A+ or B, there is an *indirect* data dependency (IDD) between A and B. In other words, if A can reach A+, and A+ and B have a DDD, then, transitively, A and B have an IDD. Note, A+ may be B. Definition 3.2 gives a mathematical definition for IDD, for two event types A and B.

As specified in definition 3.3, events a and b in a pending-event set **E**, which have event types A and B, are in conflict if:

(1) A and B have a DDD, or
(2) A and B have an IDD, and there is an event type A+, such that A+ is reachable from A, A+ and B have a DDD, and an a+ event can be scheduled with a TS $t_{a+} \leq t_b$.

i	t_{sim}	**E**	**R**
30	23.11	{(3-D, 23.53), (1-A, 24.54), (2-D, 24.72), **(1-D, 26.60)**}	{(3-D, 23.53), (1-A, 24.54), (2-D, 24.72)}
31	23.53	{(4-A, 23.69), (1-A, 24.54), (2-D, 24.72), **(1-D, 26.60)**}	{(1-A, 24.54), (2-D, 24.72), (4-A, 23.69)}
32	23.69	{(4-P, 23.69), (1-A, 24.54), (2-D, 24.72), **(1-D, 26.60)**}	{(4-P, 23.69), (1-A, 24.54), (2-D, 24.72)}
33	23.69	{(1-A, 24.54), (2-D, 24.72), **(1-D, 26.60)**, **(4-D, 32.99)**}	{(1-A, 24.54), (2-D, 24.72)}
34	24.54	{(2-D, 24.72), (1-A, 26.51), **(1-D, 26.60)**, **(4-D, 32.99)**}	{(1-A, 26.51), (2-D, 24.72)}
35	24.72	{(2-P, 24.72), (3-A, 24.87), (1-A, 26.51), **(1-D, 26.60)**, **(4-D, 32.99)**}	{(3-A, 24.87), (2-P, 24.72), (1-A, 26.51)}
36	24.72	{(3-A, 24.87), (1-A, 26.51), **(1-D, 26.60)**, **(4-D, 32.99)**, **(2-D, 33.64)**}	{(3-A, 24.87), (1-A, 26.51)}
37	24.87	{(3-P, 24.87), (1-A, 26.51), **(1-D, 26.60)**, **(4-D, 32.99)**, **(2-D, 33.64)**}	{(3-P, 24.87), (1-A, 26.51)}

Table 2: Simulation of 4-queue tandem network `Tdm4`, with $t_a \sim \mathcal{U}(1,2)$, $t_p \sim \mathcal{U}(2,10)$, $t_t \sim \mathcal{U}(0.1,0.2)$. Column i indicates IO event-execution iteration. Events in E with DDD-ECs are bold; events with IDD-ECs are bold and red.

If a pair of events (*a*,*b*) do not have an event conflict (EC), they are mutually independent. Case (1) ECs and case (2) ECs may be referred to as DDD-ECs and IDD-ECs. Definition 3.3 implies that there is an upper time-interval limit between A-type event timestamps and B-type event timestamps, at which point the potential for conflict arises. If A and B have a DDD-EC, this time limit is zero. If A and B have an IDD-EC, this time limit is defined by minimum scheduling delays and the shortest path to a DDD. Note, A and B may have both types of ECs, direct and indirect. However, if there is a DDD-EC, it is not necessary to consider an IDD-EC, as it cannot introduce a time-interval limit less than zero. Definition 3.4 quantifies this upper time-interval limit, for event-type pair (A, B) independence. Note, if A and B have a DDD, $\mathsf{A} = \mathsf{V}^G_\ell$, that is, $\mathsf{A} \xrightarrow{\text{reach}} \mathsf{A} \wedge \mathsf{A} \xleftrightarrow{\text{DDD}} \mathsf{B}$. In that case, the time to traverse the shortest path from A to $\mathbf{B'}$, $\mathsf{A} \in \mathbf{B'}$, is zero.

Definition 3.3 (Event Conflict (EC)).

For events a, $a{+}$, b with types A, A+, B, timestamps t_a, t_{a+}, t_b :

$$a \xleftrightarrow{\text{EC}} b \iff \mathsf{A} \xleftrightarrow{\text{DDD}} \mathsf{B} \;\vee\; \Big(\mathsf{A} \xleftrightarrow{\text{IDD}} \mathsf{B} \;\wedge\; \exists \mathsf{A}{+}\;|\; \big(\mathsf{A} \xrightarrow{\text{reach}} \mathsf{A}{+} \;\wedge\; \mathsf{A}{+} \xleftrightarrow{\text{DDD}} \mathsf{B} \;\wedge\; \min(t_{a+}) \le t_b\big)\Big). \tag{3}$$

Definition 3.4 (Event-Type-Pair Independence Time Limit (ITL)).

For a model G with event types $\mathbf{V^G} \coloneqq \{\mathsf{V}^G_1, \ldots, \mathsf{V}^G_n\}$ and

alias types $\mathsf{A}, \mathsf{B}, \mathsf{B'},\ \mathsf{A} \mapsto \mathsf{V}^G_i,\ \mathsf{B} \mapsto \mathsf{V}^G_j,\ \mathsf{B'} \mapsto \mathsf{V}^G_k,\ i, j, k \in \{1, \ldots, n\}$,

$$\mathbf{B'} \coloneqq \left\{ \mathsf{V}^G_\ell \in \mathbf{V^G} \mid \mathsf{A} \xrightarrow{\text{reach}} \mathsf{V}^G_\ell \;\wedge\; \mathsf{V}^G_\ell \xleftrightarrow{\text{DDD}} \mathsf{B},\ \ell \in \{1, \ldots, n\} \right\} :$$

$$\delta^G_{\mathsf{A},\mathsf{B}} \coloneqq \begin{cases} \text{time}\left(\mathsf{A} \xRightarrow{\min} \mathbf{B'}\right), & \text{if } \mathbf{B'} \neq \emptyset, \\ \infty, & \text{otherwise,} \end{cases}$$

where $\mathsf{A} \xRightarrow{\min} \mathbf{B'}$ is the minimum of all shortest paths,

from A to each $\mathsf{B'} \in \mathbf{B'}$, using minimum edge delays. (4)

The high-level definition of REs can now be updated to reflect knowledge of data dependencies and scheduling dependencies. An RE r is any event in the TS-sorted event set **E**, whose SVs will not be updated by preceding events in **E** or potential intermediate events scheduled by those preceding events, *and* which cannot update the SVs of preceding events in **E**, if r is executed OoO. The set **R** is composed of all REs in **E** at any point during the simulation of the DES model. **R** is defined mathematically in definition 3.5, for a specific DES model G, at a specific point in the simulation of G, using the terms defined as follows:

- G: the DES model, n event types,
- $\mathbf{V}^G$: the set of event types in G, size n,
- $\mathbf{N}^G$: the set of initial events, in a simulation of G,
- $\mathbf{X}^G$: the set of executed events, in a simulation of G,
- $\mathbf{X}^{G*}$: a specific set of executed events, in a simulation of G, used to derive $\mathbf{R}^G_{\mathbf{N},\mathbf{X}^*}$ from $\mathbf{E}^G_{\mathbf{N},\mathbf{X}^*}$
- $\mathbf{E}^G_{\mathbf{N},\mathbf{X}^*}$: the timestamp-sorted pending-event set in a simulation of G that is initialized with events in $\mathbf{N}^G$, after events in $\mathbf{X}^{G*}$ have executed (size m),
- $\mathbf{R}^G_{\mathbf{N},\mathbf{X}^*}$: the set of ready events in a simulation of G that is initialized with events in $\mathbf{N}^G$, after events in $\mathbf{X}^{G*}$ have executed, $\mathbf{R}^G_{\mathbf{N},\mathbf{X}^*} \subseteq \mathbf{E}^G_{\mathbf{N},\mathbf{X}^*}$ (size mm),
- e_k: the k^{th} event in $\mathbf{E}^G_{\mathbf{N},\mathbf{X}^*}$, $k \in \{1, \ldots, m\}$,
- V^G_{kk}: the event type of e_k, $k \mapsto kk$, $kk \in \{1, \ldots, n\}$,
- t_k: the TS of event e_k,
- e_j: the j^{th} event in $\mathbf{E}^G_{\mathbf{N},\mathbf{X}^*}$, $j \in \{1, \ldots, k{-}1\}$, for $k{>}1$,
- V^G_{jj}: the event type of e_j, $j \mapsto jj$, $jj \in \{1, \ldots, n\}$,
- t_j: the TS of event e_j,
- $\delta^G_{jj,kk}$: the ITL for the event-type pair $(\mathsf{V}^G_{jj}, \mathsf{V}^G_{kk})$, and
- $t_k - t_j < \delta^G_{jj,kk}$: a condition for e_k to be in $\mathbf{R}^G_{\mathbf{N},\mathbf{X}^*}$, specifying that the TS difference between events e_k and e_j is less than the ITL for the event-type pair $(\mathsf{V}^G_{jj}, \mathsf{V}^G_{kk})$.

Definition 3.5 (DDA-DES Ready Event Set $\left(\mathbf{R}^G_{\mathbf{N},\mathbf{X}^*}\right)$).

$$\mathbf{R}^G_{\mathbf{N},\mathbf{X}^*} \coloneqq \left\{ e_k \in \mathbf{E}^G_{\mathbf{N},\mathbf{X}^*} \;\middle|\; \begin{array}{l} \forall e_j \in \mathbf{E}^G_{\mathbf{N},\mathbf{X}^*},\ j{<}k, \\ t_k - t_j < \delta^G_{jj,kk} \end{array} \right\} \tag{5}$$

The ITL for each event pair in G may be pre-computed and arranged in a lookup table Δ_G. Any events e_j and e_k in any $\mathbf{E}^G$ map to enumerated event types V^G_{jj} and V^G_{kk}, which correspond to indices in Δ_G. The time limit $\delta^G_{jj,kk}$ in definition 3.5 is found at $\Delta_G(jj, kk)$.

Table 2 logs serial in-order (IO) simulation output from a 4-queue tandem network, that is, for j=4 in fig. 2, abbreviated as `Tdm4`. Event types in the table are abbreviated. Within the event set at each iteration, events with DDD-ECs or IDD-ECs are highlighted, and the remaining ready events are listed. Given the data dependencies of the model, no two `Arrive` and `Depart` events from the same

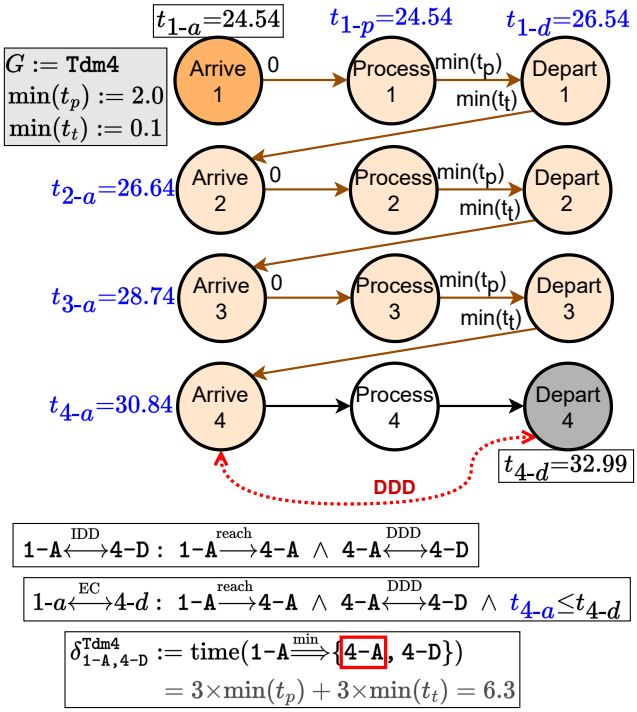


Figure 3: Visualization of the IDD-EC for i=33 in table 2, for events (1-A, 24.54) and (4-D, 32.99). Blue timestamps are the minimum times at which events can be scheduled, immediately or intermediately, from Arrive-1 at 24.54. In the Arrive-1–Depart-4 ITL calculation, Arrive-1 to Arrive-4 is the shortest path to a DDD with Depart-4.

station may execute independently at the same time. Therefore, the (1-D, 26.60) event is in conflict with a 1-A event at all iterations shown, thus the DDD-EC for (1-D, 26.60). There are two events with IDD-ECs, one of which is (4-D, 32.99), whose conflict with (1-A, 24.54) in i=33 is depicted in fig. 3. Figure 3 shows, from Arrive-1, the shortest path to conflict with Depart-4, which is Arrive-4, 6.3 time units following the execution of Arrive-1 at 24.54. To determine this most-conservative scenario, the diagram ignores any conditionality in the original EG in fig. 2 and assumes only minimum scheduling delays.

3.1.1 Mathematical Framework. Let $\mathcal{S} \coloneqq \{\mathrm{S}_1, \ldots, \mathrm{S}_n\}$ denote the set of all sets of SVs, associated with each event type $\mathrm{V}_i^G \in \mathbf{V}^G$, where $\mathbf{V}^G \coloneqq \{\mathrm{V}_1^G, \ldots, \mathrm{V}_n^G\}$ is the set of event types in the DES model G. Each set of state variables $\mathrm{S}_i \in \mathcal{S}$ corresponds to an event type V_i^G. Assume the absence of simultaneous events in the simulation of G.

For each state variable $s_{i,\ell} \in \mathrm{S}_i$, let $s_{i,\ell}(p)$ represent its value at the $p^{(\mathrm{th})}$ discrete time step, $0 \le p \le p_{\ell\ell}$, where $p_{\ell\ell}$ is the current $s_{i,\ell}$ discrete-state-change count, and subscript $\ell\ell$ indexes SV $s_{i,\ell}$ in the set of all SVs in G. Index $\ell\ell \in \{1, \ldots, N\}$, where N is the number of SVs in the model G. If $s_{i,\ell}$ is an output SV in S_i, $0 \le q_i \le p_{\ell\ell}$, where q_i is the number of executions of event type V_i^G. The TS of execution number q_i of event type V_i^G is defined by $t_{q_i}^i$. For each output SV in S_i, SV time step number $p_{\ell\ell}$ corresponds to SV TS $t_{\ell\ell} = t_{q_i}^i$. The set of all timestamps for event type V_i^G, is given by $\mathbf{T}^i \coloneqq \{t_1^i, \ldots, t_{q_i}^i\}$.

For event type V_i^G, the set of all values of each $s_{i,\ell}(q)$ at each time step q, $0 \le q \le q_i$, for q_i time steps, is in $\mathcal{S}^i \coloneqq \{\mathrm{S}_1^i, \ldots, \mathrm{S}_{q_i}^i\}$, where $\mathrm{S}_q^i \coloneqq \{s_{i,1}(q), \ldots, s_{i,n_i}(q)\}$. Values in $\mathrm{S}_{q_i}^i \equiv \mathrm{S}_i$, if SVs in S_i have not been updated since $\mathrm{S}_{q_i}^i$ was recorded. The simulation trace $(\mathcal{T}, \mathbb{S})$ contains all $\mathbf{T}^i$ and all $\mathcal{S}^i, i \in \{1, \ldots, n\}$. Definition 3.6 specifies the conditions under which a serial OoO simulation of model G is inconsistent with the in-order simulation of G, using the OoO and in-order traces.

Definition 3.6 (OoO Simulation Divergence).

A serial OoO simulation of DES model G has diverged from the in-order simulation of G, given OoO simulation trace $(\mathcal{T}, \mathbb{S})$ and in-order simulation trace $(\mathcal{T}^{(\mathrm{IO})}, \mathbb{S}^{(\mathrm{IO})})$:

$$\mathrm{OoO_S}(G) \neq \mathrm{IO}(G) \iff \exists i \in \{1, \ldots, n\}, \exists q \in \{1, \ldots, q_i\},$$
$$\exists s_{i,\ell} \in \mathrm{S}_i \mid \Big(\big(t_q^i \neq t_q^{i(\mathrm{IO})}\big) \vee \big(s_{i,\ell}(q) \neq s_{i,\ell}^{(\mathrm{IO})}(q)\big) \Big). \tag{6}$$

Definition 3.7 (Potential for Divergence).

A serial OoO simulation of DES model G may diverge from the in-order simulation of G, given OoO event e_k with event type V_{kk}^G executes at TS t_k :

$$\mathrm{OoO_S}(G) \not\perp \mathrm{IO}(G) \mid e_k \iff \exists \mathrm{V}_i^G \in \mathbf{V}^G, \mathrm{V}_i^G \neq \mathrm{V}_{kk}^G \mid$$
$$\Big(\big(\mathrm{S}_{\mathrm{kk}\cap \mathrm{i}} \cap \mathrm{S}_{\mathrm{kk}_O} \neq \emptyset \;\vee\; \mathrm{S}_{\mathrm{kk}\cap \mathrm{i}} \cap \mathrm{S}_{\mathrm{i}_O} \neq \emptyset\big) \wedge$$
$$\mathbf{T}^i \neq \mathbf{T}^{i(\mathrm{IO})}(<t_k) \Big) \;\vee\; \mathbf{T}^{kk} \neq \mathbf{T}^{kk(\mathrm{IO})}(<t_k). \tag{7}$$

Before executing an OoO event e_k, it is possible to check that e_k may cause simulation divergence, using definition 3.7. This states that OoO execution of e_k may cause divergence if:

(1) there is an event type V_i^G that shares SVs with event type V_{kk}^G, for which these SVs are output SVs in either V_{kk}^G or in V_i^G, and there is an inconsistency in the states of the executed V_i^G–event TS records, up to the execution of e_k at TS t_k, between the serial OoO run and the IO run, or
(2) there is an inconsistency in the states of the executed V_{kk}^G–event TS records, up to the execution of e_k at TS t_k, between the serial OoO run and the IO run.

If (2) is true, the OoO simulation will certainly diverge. In either case, if e_k executes at this point, the state of the simulation of G from the perspective of e_k is inconsistent with the analogous event-local state in the in-order simulation, that is, inconsistent with consideration to the execution of all prior dependent events. V_{kk}^G–input SVs may be inconsistent with the expected values in the in-order simulation, at the point of e_k execution. SV-inconsistency is not guaranteed under these circumstances, since SV updates may be conditional, or perhaps by chance the SVs could be equivalent. Note, definition 3.7 may function without an exact sequence of specific timestamps, for each dependent event type and for V_{kk}^G, to satisfy the requirement that $\mathrm{OoO_S}(G)$ will not diverge if event e_k executes OoO. However, the number of completed executions,

for each event type, at the specific time t_k must be known. So, for practical reasons, $\mathbf{T}^i$ and $\mathbf{T}^{kk}$ are used.

Given the TS-ordered pending-event set $\mathbf{E}$, and a pair of events (e_j, e_k), $j,k \in \{1,\ldots n\}$, $j<k$, in $\mathbf{E}$, with event types $\mathsf{V}^G_{jj}, \mathsf{V}^G_{kk}$, definition 3.7 is sufficient for identifying potential divergence caused by the following problematic cases:

(1) conflict between $e_{j/+}$ input SVs and e_k output SVs,
(2) conflict between $e_{j/+}$ output SVs and e_k input SVs, and
(3) conflict between $e_{j/+}$ output SVs and e_k output SVs,

where $e_{j/+}$ is shorthand for event e_j or any event e_{j+} scheduled by e_j. In any case, if e_j and e_k are causally ordered, and e_k is executed OoO, definition 3.7 can detect the potential for divergence, before executing e_k. If there is conflict for an (e_j, e_k) pair, then there is a V^G_i in definition 3.7 corresponding to V^G_{jj} or V^G_{jj+}.

Theorem 3.8. *Given a set of ready events $\mathbf{R}^G_{\mathrm{N,X}^*}$, each event r_y in the set, $y \in \{1,\ldots,mm\}$, with TS t_y and event type V^G_{yy} may be executed in any order without causing simulation divergence.*

Proof. This is proven by contradiction. Assume that executing ready events in $\mathbf{R}^G_{\mathrm{N,X}^*}$ in any order may cause simulation divergence, according to definition 3.7. This supposes that for any $r_y \in \mathbf{R}^G_{\mathrm{N,X}^*}$,

(1) the event-local state of the simulation is dependent on an event $r_x \in \mathbf{R}^G_{\mathrm{N,X}^*}, x \in \{1,\ldots,mm\}, y \neq x$, or
(2) the event-local state of the simulation is dependent on a non-ready event w in $\mathbf{E}^G_{\mathrm{N,X}^*}$ with TS $t_w, t_w < t_y$.

In either case, call the dependent event d, which has a TS t_d and event type V^G_{dd}. For event d to exist, d and r_y must be in conflict, that is, $d \overset{\mathrm{EC}}{\longleftrightarrow} r_y$. For this to be true, $t_y - t_d >= \delta^G_{dd,yy}$, meaning event d, or another event that d may schedule immediately or intermediately, can affect the event-local simulation state of r_y implied in definition 3.7, where $e_k \leftarrow r_y$. This implies that d and r_y are a causally-ordered event pair, in which case, $r_y \notin \mathbf{R}^G_{\mathrm{N,X}^*}$. □

3.2 Data-Dependence Event Graph

The event-graph formalism is modified to data dependence event graph (DDEG) to better define and visualize the SV and scheduling information in a simulation model's EG, for DDA-DES. The modifications to the event graph formalism are enumerated as follows:

(1) For each EG vertex, event-routine SV usage is distilled to sets of input and output SVs.
(2) Conflicting SV relationships between vertices are depicted with data-dependency edges.
(3) Only minimum scheduling delays are considered.
(4) Self-scheduling edges are eliminated.
(5) Conditional scheduling of new events is regarded as and depicted as non-conditional.

The DDEG of a simulation model does not replace the EG, for the purpose of understanding and implementing a simulation model, as some key information is not included in the DDEG. The DDEG is used to understand the DDA simulation of the model, that is, how the DDA simulation executive [24, 25, 41] selects ready events. This knowledge may be encapsulated in an independence time limit (ITL) lookup table, which is pre-computed prior to simulation.

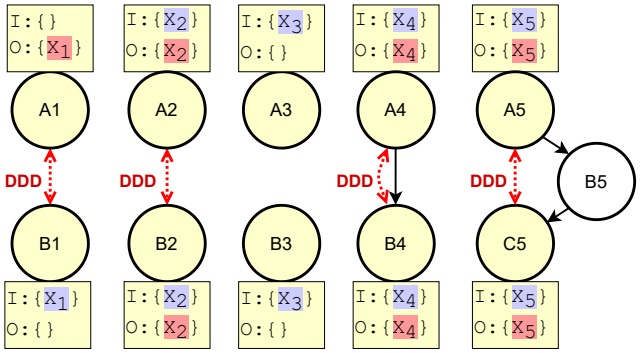


Figure 4: DDEG data-dependence relationships depicted by data-dependency edges. Vertices shown may be subsets of graphs, which have additional vertices and edges.

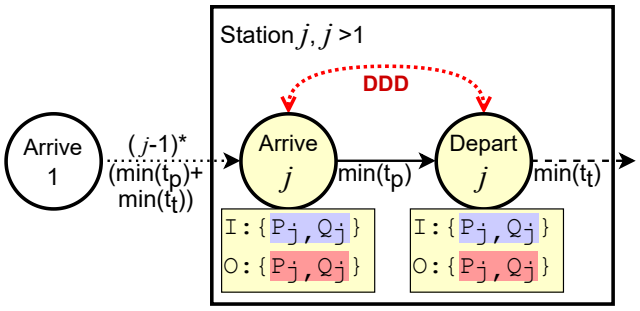


Figure 5: An `Arrive`→`Depart` station DDEG. The EG is reduced, removing the unnecessary `Process` vertex, and then translated to the DDEG, removing conditionality and self-scheduling edges, and using minimum scheduling delays.

Modification (1) clearly defines SV usage for each vertex, which assists in drawing SV-usage and data-dependence relationships between vertices in (2). Seeing these relationships with data-dependency edges may provide insight into the capacity of a model for OoO and parallel execution. Figure 4 provides multiple examples of SV usage and corresponding data-dependency edges.

As stated in modification (3), the DDEG considers only minimum scheduling delays. These minimum delays are used to compute the shortest paths between vertices, for ITL calculations, as per definition 3.4. For similar reasons, modification (4) is implemented, removing self-scheduling edges. These edges are irrelevant for computing shortest paths for DDA-DES.

Modification (5) to the EG formalism is needed for DDA-DES, as conditional scheduling is regarded as non-conditional. This is a constraint that DDA-DES requires to ensure causality. Currently, there is no "optimistic" mode, with the capacity to correct causality violations, as exists for optimistic SD-PDES [7, 18].

In fig. 5, the `Arrive`→`Process`→`Depart` station EG is translated to a DDEG, fulfilling variant requirements (1)-(4). Note, there is no possibility of pairwise event independence within station j, as stated in section 3.1.

4 Packet-Routing Network Model Experiments

Several packet-routing network simulation models are used to test DDA-DES parallel capacity. Each network node is similar to the `Arrive`→`Process`→`Depart` station model in fig. 2, with additional packet-routing functionality in the arrival and departure vertices. Also, the implementations lack the superfluous `Process` event type, which can be removed without affecting functionality. In this case, an arrival event can directly schedule a departure, if a processor is available, and a departure event can schedule another departure, if there are more packets in the queue. For all models, one processor is available per node. Various network topologies are employed, including ring, 2-D and 3-D mesh, and 3-D torus. The 2-D and 3-D models feature adaptive routing, similar to strategies employed for network-on-chip (NoC) packet routing in [2, 29, 40]. For these models with adaptive (or dynamic) routing, several hop radii are tested, where each node has queue-size information for all neighbors within a specified number of hops. At each hop, a locally-optimal route is identified.

Table 3 defines experimental parameters for each model, including model size and dimensions, tested hop radii and delay distribution configurations, and the number of packets generated in each node. Each model is tested with several triangular-delay distribution configurations, which scale transit delays relative to processing delays, as defined in table 4. Note that some nodes in each configuration have quicker generation delays (t_g), to create busier regions in the network. Algorithm 1 defines at a low level how the simulation models are executed, to measure the number of ready events (REs) that are available throughout the simulation runs. Red text in algorithm 1 refers to formal definitions in definition 3.5. That is, when the working event sets **E** and **R** are equivalent to the formal event sets $\mathbf{E}^G_{\mathrm{N,X}^*}$ and $\mathbf{R}^G_{\mathrm{N,X}^*}$, **E** and **R** are in red. Gray text in algorithm 1 indicates executed-event set data and operations that are superfluous to the experimental design, but support references to the formal definitions of $\mathbf{E}^G_{\mathrm{N,X}^*}$ and $\mathbf{R}^G_{\mathrm{N,X}^*}$.

Results from algorithm 1 for each experimental scheme are in table 5 and fig. 6, where each plot marker in fig. 6 is the mean of all means of **M**, for a given delay-distribution configuration tested with ten different random-number generator seed values. That is, take the mean $\overline{\mathbf{M}}_{dd}$ of each $\mathbf{M}_{dd}$ for a given configuration using seed index dd, $dd \in \{1, \ldots, 10\}$, and then the number plotted is the mean of $\{\overline{\mathbf{M}}_1, \ldots, \overline{\mathbf{M}}_{10}\}$. In table 5, the minimum, median, and maximum ready-event measurements correspond to plotted values on each line in fig. 6. For the tested schemes, the mean size of the ready-event set varies from roughly 1.5 to 110. These extremes correspond to the 64-node four-hop torus model and the 729-node one-hop 2D mesh model. If the ring model were tested with 729 nodes, it likely would exhibit greater parallelism than this 2D mesh model. Note, these measurements are the results of finding and executing all ready events in the pending-event set in an iterative manner, as defined in algorithm 1.

According to table 5 and fig. 6, the available parallelism in a model varies greatly, depending on relationships with neighboring nodes, the size of the model, and delay distribution parameters. There is a trend of decreasing parallelism, as the predictability of the future state of a model decreases. For example, in the ring model, any node may be affected by incoming packets from only two directions, but in the 3D mesh and 3D torus models, incoming packets may arrive from six directions. This behavior explains the decreasing parallelism with higher-dimensional models. For 2D and 3D models, larger hop radii reduce parallelism since nodes have more data dependencies, which increases opportunities for event conflict. For example, for the 8×8 2D mesh models with hop radii 1, 2, and 4, the maximum number of parallel departure events are

Table 3: Experimental Network Model Schemes

Model	Sizes	Dims.	Hop Radii	Delay Configs.	Num. Gens.
Ring (1D)	64	64	N/A	1-10	100
Mesh (2D)	64	8×8	1, 2, 4	1-10	100
	729	27×27	1, 2, 4	1-10	100
Mesh (3D)	64	4×4×4	1, 2, 4	1-10	100
	729	9×9×9	1, 2, 4	1-10	100
Torus (3D)	64	4×4×4	1, 2, 4	1-10	100
	729	9×9×9	1, 2, 4	1-10	100

Table 4: Triangular-Delay Distribution Configurations

Config.	Generate (t_g)	Process (t_p)	Transit (t_t)
1	(10, 12, 16)	(2, 3, 4)	(0.50, 0.75, 1.00)
2			(1.00, 1.50, 2.00)
3			(1.50, 2.25, 3.00)
4			(2.00, 3.00, 4.00)
5			(2.50, 3.75, 5.00)
6			(3.00, 4.50, 6.00)
7			(3.50, 5.25, 7.00)
8			(4.00, 6.00, 8.00)
9			(4.50, 6.75, 9.00)
10			(5.00, 7.50, 10.00)

Algorithm 1 Experimental Design for Network Model Simulations

Require: G: network simulation model with n nodes,
E: empty pending-event set (when **E** shown, $\mathbf{E} = \mathbf{E}^G_{\mathrm{N,X}^*}$),
R: empty ready-event set (when **R** shown, $\mathbf{R} = \mathbf{R}^G_{\mathrm{N,X}^*}$),
$\mathbf{N}^G$: non-empty initial-event set (size n),
$\mathbf{X}^G$: empty executed-event set,
$\mathbf{X}^{G*}$: empty reference executed-event set
Ensure: **M**: set of all **R** size measurements
1: Set **E** ← $\mathbf{N}^G$
2: Set **M** ← []
3: **while** **E** is not empty **do**
4: Identify all ready events in **E**, copy to **R**
5: Append |**R**| to **M**
6: **while** **R** is not empty **do**
7: Randomly select an event r from **R**
8: Execute r and schedule new r events in **E**
9: Remove r from **R**, **E** and move to $\mathbf{X}^G$
10: **end while**
11: Set $\mathbf{X}^{G*} \leftarrow \mathbf{X}^G$
12: **end while**

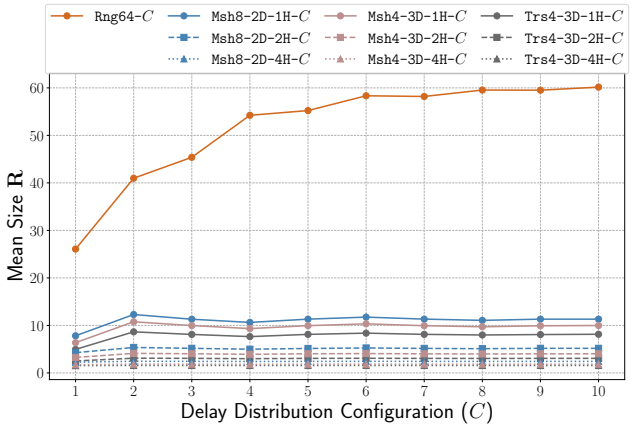
32, 16, and 4, under ideal conditions. Queues within the hop radius cannot update during route planning. In addition to the effects that network topology and route-planning data dependencies have on parallelism, there is a clear positive correlation between the size of the model and the available parallelism, across all tested schemes.

Finally, changing the delay distribution parameters changes the speed with which a node may affect the future state of another node. This is seen clearly in the ring model, where parallelism appears to be positively correlated with $\min(t_t)/\min(t_p)$. As minimum transit delays increase, relative to processing delays, packets in neighboring nodes are more distant, and the future state of a node is increasingly predictable. However, this increase in parallelism is asymptotic for larger transit delays, as model parallelism reaches saturation. This correlation is less clear with the 2D and 3D models, for the tested configurations. Perhaps the network topologies create more complex relationships between nodes, where increasingly long transit delays cannot so proportionately increase event readiness, compared with the ring model. Further analysis regarding the interaction of these factors and their effect on event readiness is left for future work.

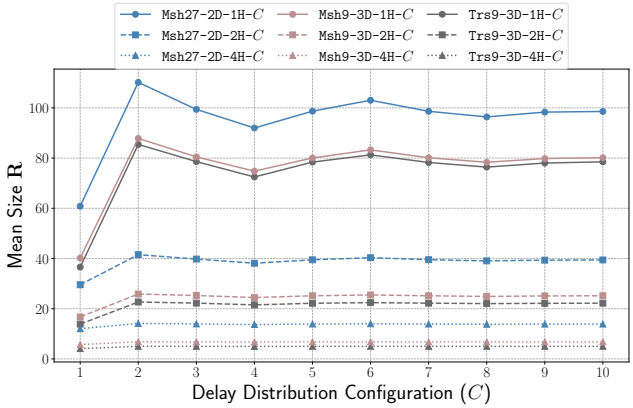
Table 5: Network Model Ready-Event Statistics

Network	Size	Hop Radius	Mean Ready Events		
			Min	Median	Max
Ring (1D)	64	N/A	26.07	55.22	60.16
Mesh (2D)	64	1	7.82	11.32	12.30
		2	4.27	5.15	5.36
		4	2.26	2.43	2.45
	729	1	60.84	98.68	110.14
		2	29.56	39.48	41.53
		4	12.04	13.90	14.14
Mesh (3D)	64	1	6.37	9.95	10.77
		2	3.26	4.03	4.12
		4	1.74	1.84	1.84
	729	1	40.17	80.04	87.84
		2	16.72	25.15	25.85
		4	5.71	6.81	6.84
Torus (3D)	64	1	4.96	8.12	8.65
		2	2.52	3.06	3.10
		4	1.53	1.57	1.57
	729	1	36.57	78.39	85.44
		2	13.87	22.17	22.67
		4	4.10	5.03	5.04

To be certain that events are being executed OoO, table 6 and fig. 7 provide evidence of OoO execution. For each of the experimental schemes in table 6 and fig. 7, the OoO event-execution sequence is compared with the in-order event-execution sequence. Each OoO-run event is located in the in-order sequence, and the difference in the index values, of the same events in the two sequences, is appended to a set of all such event-order differences for the OoO run. After the OoO run completes, the mean of this set of differences is taken, to get an aggregate measurement for the OoO run. To get the values in table 6 and fig. 7, take the mean of all such single-run means, across all seed-varied runs for a given experimental



(a) 64-nodes experiments.



(b) 729-node experiments.

Figure 6: Mean ready-event measurements for simulation experiments using 64-node and 729-node network schemes. Numerical values of some plotted data points are in table 5.

Table 6: Event-Execution Order Difference Statistics

Network	Size	Hop Radius	Mean Order Difference		
			Min	Median	Max
Ring (1D)	64	N/A	53.41	133.26	232.18
Mesh (2D)	64	1	14.40	23.47	24.08
		2	7.80	9.85	10.10
		4	2.81	3.01	3.04
Mesh (3D)	64	1	12.14	18.52	18.73
		2	5.52	6.49	6.77
		4	1.41	1.46	1.49
Torus (3D)	64	1	10.03	13.65	14.23
		2	3.44	3.90	4.01
		4	0.81	0.82	0.83

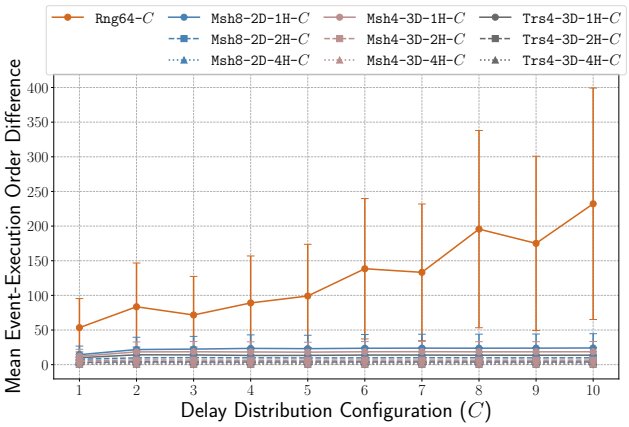


Figure 7: Mean event-execution order difference measurements, for 64-node experiments. Numerical values of some plotted data points are in table 6.

scheme. For all 64-node experimental schemes, this composite mean varies from roughly 0.8 for the four-hop torus model to 232 for the ring model, meaning this torus model exhibits scant unordered-ness, and the ring model is highly unordered. Recall that this torus model also demonstrates the least parallelism of all the experimental models. There is a clear relationship between unordered-ness and parallelism, as it is the capacity of a model to execute events OoO that yields parallelism, as is depicted in fig. 1.

To validate that OoO runs are consistent with in-order runs, each event type generates a trace, where upon each event execution, the event timestamp and the value of each input SV and each output SV at the time of execution is recorded. Then, each OoO event-type trace may be compared with each in-order event-type trace, to check for divergence, as defined in definition 3.6. In this work, all experimental OoO runs are consistent with the corresponding in-order runs, according to this definition.

5 Discussion and Future Work

This work has measured the available event-level parallelism in several discrete-event simulation models, using data dependencies and scheduling dependencies to identify events that are ready to execute. These events are referred to as ready events.

Ready events are identified by applying data-dependence analysis to discrete-event simulation to determine state-variable relationships between simulation model event types. These state-variable relationships, along with scheduling dependencies, are used to define mutual independence between pairs of event types. This pairwise mutual independence is quantified as a time limit (ITL), which specifies the magnitude with which timestamps belonging to these event-type pairs may differ and still maintain mutual independence. Several mathematical definitions are introduced, which support the definitions of the ITL and the set of ready events. A variant of the event-graph formalism, data-dependence event graph (DDEG), is proposed, which benefits this novel approach, data-dependence-analysis-DES (DDA-DES). DDEG abstracts away low-level state-variable updating and scheduling capabilities that are invisible to DDA-DES. Multi-factorial experiments that measure model parallelism demonstrate or imply several points:

(1) Model parallelism can be measured explicitly, meaning that given a DES model G with a set of experimental parameters, it is possible to define exactly what the set of ready events $\mathbf{R}^G_{\mathbf{N},\mathbf{X}^*}$ is at any point in the simulation, based on the set of initial events $\mathbf{N}^G$ and an exact set of executed events $\mathbf{X}^{G*}$. Assuming the absence of simultaneous events, this is a well-defined and deterministic set of events.
(2) Model parallelism is dependent on several factors, including network topology, network size, data dependencies between neighboring nodes, and delay distribution parameters. Experimental results are consistent with model characteristics, that is, models that feature fewer data dependencies between neighboring nodes exhibit greater parallelism.
(3) For some models, the definitions presented that support the identification of ready events may not not always capture realistic model behavior, and are perhaps too restrictive.

This final point refers to the behavior of the 2D and 3D models, in which outgoing packets at neighboring nodes, which cannot visit or re-visit a node of interest, may still needlessly threaten to affect the future state of a node, according to the ITL in definition 3.4. The execution of an event at a neighboring node cannot affect the future state of the node of interest through data dependencies, as there is no pathway to the node of interest. In this case, there must be special rules for determining if one event type may schedule another event type, either immediately or intermediately, based on parameters that define the state of the model. Say there is a type-A event with an outgoing packet scheduled at a node in the vicinity of a type-B-event node, where the type-A packet cannot visit or re-visit the type-B node. Referring to definition 3.4, the network topology should be artificially constrained in this case such that $\mathsf{A} \xrightarrow{\text{reach}} \mathsf{V}^G_\ell \ \wedge \ \mathsf{V}^G_\ell \xleftrightarrow{\text{DDD}} \mathsf{B}$ is not true, where V^G_ℓ is an event type that has a direct data dependency (DDD) with event-type B. In this case, V^G_ℓ is an arrival event type on the type-B node. To implement this constraint, network edges should be removed in this case.

In summary, all tested models demonstrate the ability to continuously or regularly execute multiple events in parallel, with up to 110 ready events available on average per event-execution cycle. These findings justify continued research into finding parallelism in DES models, and creating a practical approach to the dynamic identification of ready events, for parallel execution. DDA-DES may benefit models that are not easy to parallelize through spatial decomposition (SD), such as the 2D and 3D models presented in this work, which use queue data from neighboring nodes for decision making. Future work may include but is not limited to:

- further analysis of relationships between model types and parameters, pending-event sets, and ready-event sets,
- modification of definitions to find increased parallelism, while still maintaining consistency with in-order execution,
- experimentation to measure parallelism in other DES models,
- comparisons to SD-method parallel capacities,
- the development of parallel DDA-DES (DDA-PDES) algorithms, implementations, and experiments, and
- integration of DDA-DES into multi-LP SD methods.

References

[1] Randy Allen and Ken Kennedy. 1987. Automatic translation of Fortran programs to vector form. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 9, 4 (1987), 491–542.

[2] Giuseppe Ascia, Vincenzo Catania, Maurizio Palesi, and Davide Patti. 2008. Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip. *IEEE transactions on computers* 57, 6 (2008), 809–820.

[3] Oran Berry and David Jefferson. 1985. Critical path analysis of distributed simulation.. In *SCS Conf. Distributed Simulation*. 57–60.

[4] Randal R Bryant. 1977. Simulation of packet communication architecture computer systems. (1977).

[5] Arnold H Buss. 1996. Modeling with event graphs. In *Proceedings of the 28th conference on winter simulation*. 153–160.

[6] Christopher D Carothers, David Bauer, and Shawn Pearce. 2002. ROSS: A high-performance, low-memory, modular Time Warp system. *Journal of parallel and distributed computing* 62, 11 (2002), 1648–1669.

[7] Christopher D Carothers, Kalyan S Perumalla, and Richard M Fujimoto. 1999. Efficient optimistic parallel simulations using reverse computation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 9, 3 (1999), 224–253.

[8] K. Mani Chandy and Jayadev Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on software engineering* 5 (1979), 440–452.

[9] Weiwei Chen and Rainer Dömer. 2013. Optimized out-of-order parallel discrete event simulation using predictions. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 3–8.

[10] Weiwei Chen, Xu Han, Che-Wei Chang, Guantao Liu, and Rainer Dömer. 2014. Out-of-order parallel discrete event simulation for transaction level models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 12 (2014), 1859–1872.

[11] Weiwei Chen, Xu Han, and Rainer Dömer. 2012. Out-of-order parallel simulation for ESL design. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 141–146.

[12] BK Choi. 2013. *Modeling and simulation of discrete event systems*. Wiley.

[13] Jonas Friederich, Wentong Cai, Boon Ping Gan, and Sanja Lazarova-Molnar. 2023. Equipment-centric Data-driven Reliability Assessment of Complex Manufacturing Systems. In *Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 62–72.

[14] Richard M Fujimoto. 2000. *Parallel and Distributed Simulation Systems*. John Wiley & Sons.

[15] Daniel D Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, and Shuqing Zhao. 2012. *SpecC: Specification language and methodology*. Springer Science & Business Media.

[16] Thorsten Grötker. 2002. *System Design with SystemC™*. Springer Science & Business Media.

[17] Martin C Herbordt, Md Ashfaquzzaman Khan, and Tony Dean. 2009. Parallel discrete event simulation of molecular dynamics through event-based decomposition. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 129–136.

[18] David R Jefferson. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 7, 3 (1985), 404–425.

[19] Douglas W Jones. 1986. Concurrent simulation: An alternative to distributed simulation. In *Proceedings of the 18th conference on Winter simulation*. 417–423.

[20] Douglas W Jones, C-C Chou, Debra Renk, and Steven C Bruell. 1989. Experience with concurrent simulation. In *Proceedings of the 21st conference on Winter simulation*. 756–764.

[21] Yao Kang, Xin Wang, and Zhiling Lan. 2023. Workload interference prevention with intelligent routing and flexible job placement on dragonfly. In *Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 23–33.

[22] Georg Kunz, Mirko Stoffers, James Gross, and Klaus Wehrle. 2012. Know thy simulation model: analyzing event interactions for probabilistic synchronization in parallel simulations. In *5th International ICST Conference on Simulation Tools and Techniques 2012 (SIMUTools 2012); Desenzano, Italy, 19-23 March 2012*. 119–128.

[23] Leslie Lamport. 2019. Time, clocks, and the ordering of events in a distributed system. In *Concurrency: the Works of Leslie Lamport*. 179–196.

[24] James F Leathrum, Roland R Mielke, Andrew J Collins, and Michel A Audette. 2017. Proposed unified discrete event simulation content roadmap for M&S curricula. In *2017 Winter Simulation Conference (WSC)*. IEEE, 4300–4311.

[25] James F Leathrum Jr, Reejo Mathew, and Thomas W Mastaglio. 2011. Modeling the impact of security and disaster response on cargo operations. *Simulation* 87, 8 (2011), 696–710.

[26] Qi Liu and Gabriel Wainer. 2012. Multicore acceleration of discrete event system specification systems. *Simulation* 88, 7 (2012), 801–831.

[27] Tushar Mohanrao Lone. 2024. Development of an open-source library for supply chain modeling and optimization. In *Proceedings of the 38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 73–74.

[28] AA Lyubchenko, EY Kopytov, AA Bogdanov, and VA Maystrenko. 2020. Discrete-event simulation of operation and maintenance of telecommunication equipment using anylogic-based multi-state models. In *Journal of Physics: Conference Series*, Vol. 1441. IOP Publishing, 012046.

[29] Terrence Mak, Peter YK Cheung, Kai-Pui Lam, and Wayne Luk. 2010. Adaptive routing in network-on-chips using a dynamic-programming network. *IEEE Transactions on industrial electronics* 58, 8 (2010), 3701–3716.

[30] Dror E Maydan, John L Hennessy, and Monica S Lam. 1991. Efficient and exact data dependence analysis. In *Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*. 1–14.

[31] Zhuoxiao Meng, Anibal Siguenza-Torres, Mingyue Gao, Margherita Grossi, Alexander Wieder, Xiaorui Du, Stefano Bortoli, Christoph Sommer, and Alois Knoll. 2023. Towards Discrete-Event, Aggregating, and Relational Control Interfaces for Traffic Simulation. In *Proceedings of the 2023 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 12–22.

[32] Eric Mikida and Laxmikant Kale. 2018. Adaptive methods for irregular parallel discrete event simulation workloads. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 189–200.

[33] Francesco Quaglia and Roberto Baldoni. 1999. Exploiting intra-object dependencies in parallel simulation. *Inform. Process. Lett.* 70, 3 (1999), 119–125.

[34] Robert G Sargent. 1988. Event graph modelling for simulation with an application to flexible manufacturing systems. *Management science* 34, 10 (1988), 1231–1251.

[35] Lee Schruben. 1983. Simulation modeling with event graphs. *Commun. ACM* 26, 11 (1983), 957–963.

[36] Lee Schruben and Enver Yucesan. 1994. Transforming Petri nets into event graph models. In *Proceedings of Winter Simulation Conference*. IEEE, 560–565.

[37] Hussam M Soliman and Adel S Elmaghraby. 1995. The parallel-event approach to discrete-event simulation. *ACM SIGSIM Simulation Digest* 24, 3 (1995), 21–39.

[38] Sudhir Srinivasan and P Reynolds. 1993. On critical path analysis of parallel discrete event simulations. *Computer Science Report No. TR-93-29* (1993).

[39] Simon JE Taylor, Farshad Fatin, and Thierry Delaitre. 1995. Estimating the benefit of the parallelisation of discrete event simulation. In *Proceedings of the 27th conference on Winter simulation*. 674–681.

[40] Mohammad Trik, Saadat Pour Mozaffari, and Amir Massoud Bidgoli. 2021. Providing an Adaptive Routing along with a Hybrid Selection Strategy to Increase Efficiency in NoC-Based Neuromorphic Systems. *Computational Intelligence and Neuroscience* 2021, 1 (2021), 8338903.

[41] Brandon Waddell and James F Leathrum. 2019. A multithreaded simulation executive in support of discrete event simulations. In *2019 Winter Simulation Conference (WSC)*. IEEE, 2677–2688.

[42] Yizhuo Wang, Zhiwei Gao, Weixing Ji, Han Zhang, and Duzheng Qing. 2018. Exploiting task-based parallelism for parallel discrete event simulation. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 562–566.

[43] Xiaoliang Wu, Alexander Kolar, Joaquin Chung, Dong Jin, Martin Suchara, and Rajkumar Kettimuthu. 2024. Parallel simulation of quantum networks with distributed quantum state management. *ACM Transactions on Modeling and Computer Simulation* 34, 2 (2024), 1–28.

[44] Xiongxiao Xu, Kevin A. Brown, Tanwi Mallick, Xin Wang, Elkin Cruz-Camacho, Robert B. Ross, Christopher D. Carothers, Zhiling Lan, and Kai Shu. 2024. Surrogate Modeling for HPC Application Iteration Times Forecasting with Network Features. In *Proceedings of the 38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 93–97.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009