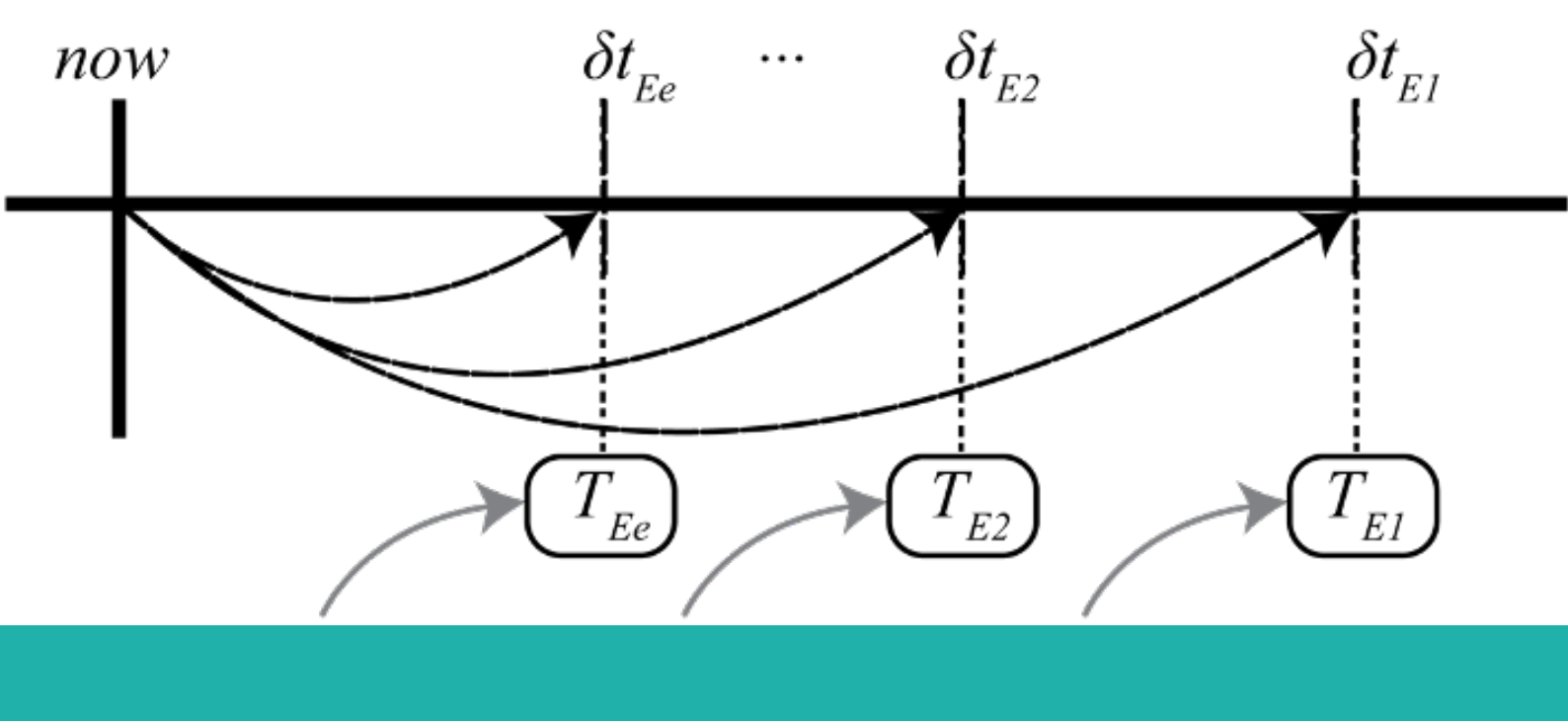


Computer Science Research Needs for Parallel Discrete Event Simulation (PDES)

U.S. Department of Energy

Advanced Scientific Computing Research

Roundtable Report

Date: May 11, 2022

DOI: 10.2172/1855247

U.S. DEPARTMENT OF ENERGY
Office of Science

Contributors

Kalyan Perumalla	Oak Ridge National Laboratory
Peter Barnes	Lawrence Livermore National Laboratory
Maximilian Bremer	Lawrence Berkeley National Laboratory
Kevin Brown	Argonne National Laboratory
Cy Chan	Lawrence Berkeley National Laboratory
Stephan Eidenbenz	Los Alamos National Laboratory
K. Scott Hemmert	Sandia National Laboratory
Adolfy Hoisie	Brookhaven National Laboratories
Benjamin Newton	Sandia National Laboratories
James Nutaro	Oak Ridge National Laboratory
Tomas Oppelstrup	Lawrence Livermore National Laboratory
Robert Ross	Argonne National Laboratory
Markus Schordan	Lawrence Livermore National Laboratory
Nathan Urban	Brookhaven National Laboratory

DOE/ASCR Points of Contact: Hal Finkel and Randall Laviolette

Disclaimer

Acronyms

AI	Artificial Intelligence
API	Common Unified Data Architecture
ASCAC	Advanced Scientific Computing Advisory Committee
ASCR	Advanced Scientific Computing Research
ASIC	Application-Specific Integrated Circuit
CGRA	Coarse-Grained Reconfigurable Array
CPU	Central Processing Unit
CUDA	Common Unified Device Architecture
DOE	Department of Energy
FPGA	Field Programmable Gate Array
GPU	Graphical Processing Unit
HPC	High Performance Computing
IPU	Intelligent Processing Unit
KMC	Kinteic Monte Carlo
LP	Logical Process
ML	Machine Learning
MPI	Message Passing Interface
NPI	Non-Pharmaceutical Intervention
OpenMP	Open Multi-Processing
PDES	Parallel Discrete Event Simulation
PRO	Priority Research Opportunity
SEIR	Susceptible Exposed Infective Recovered
SIMD	Single Instruction Multiple Data
SIR	Susceptible Infected Recovered

Contents

Chapter 1 Executive Summary

Historically, scientific computing efforts have demonstrated the clear need for, and effective use of, supercomputing with traditional time-stepped simulations. Nevertheless, there are several areas in the mission spaces of the U.S. Department of Energy and other agencies waiting to tap advanced computing research using a different, discrete event style of modeling, simulation, and analysis. These span a wide spectrum of applications including energy grid resilience, urban planning and policy, transportation science, building technologies, emergency response and planning, environmental impact analysis, computational epidemiology, Internet communications, cyber security, and cyber-physical systems, to name only a few. Even within traditional scientific applications, the role of discrete event modes of execution is increasing in the form of new event-based mathematical solvers such as quantized state integration methods and discrete-continuous hybrid system solvers. Co-design of advanced supercomputing hardware systems is another area that exploits discrete event simulation at its core for effective analyses. Complex systems, entity behaviors and interconnections play a significant role in all these applications, which are mapped to large-scale models with discrete event formulations.

To make advancements in all the aforementioned scientific areas, many technical aspects need to be more thoroughly studied and deeply understood in parallel discrete event simulation (PDES). The unique dynamics inherent in a discrete event modeling approach, by their very nature, intersect and influence the entire stack of the computing system, including (a) the unique nature of the instruction sets exercised in PDES workloads without a predominance of high-precision floating point operations, (b) virtual time-constrained multi-threaded execution of many logical processes per processor, (c) extremely variable and difficult to predict network traffic characteristics, (d) interfaces and inter-dependencies with machine learning and artificial intelligence codes at higher software layers, and (e) highly challenging load balancing needs, especially in effectively accounting for accelerated/extremely heterogeneous computing in current and future high-performance computing systems. Efficient and accurate parallel execution of PDES workloads is also dominated by challenges in dealing with their asynchronous concurrency fundamentally present at the model level. Conservative synchronization, optimistic/speculative synchronization, and their hybrid schemes open new questions in fundamental computer science with respect to reversibility of computation and prediction (lookahead) of behaviors inherent within model codes. On the implementation front, there are relatively few scalable, general-purpose parallel discrete event simulators in the world, and even fewer have been studied on emerging hardware platforms. To enable scientific advances using PDES, the research needs in computer science must also be pursued and met in the intersection of the algorithmic and hardware-aware aspects of scalable PDES engines.

This report is aimed at capturing a computer science-oriented view of this important area of research in PDES, presenting a sample of important applications with their inherent discrete event technology elements. Needs are outlined in core areas of parallel discrete event research as well as cross-cutting directions in computer science research that positively impact scientific advancements across several important application areas. A selection of priority research opportunities in advanced computing for PDES is identified to serve as reference for key research topics and their order of importance for scientific advancements.

Chapter 2 Overview

2.1 Challenges from a Changing Landscape

In the rapidly evolving landscape of high-end computing, significant changes are happening across scalable hardware system solutions, complex software technologies as well as application needs, which in combination are bringing research in PDES to the fore.

Hardware: On the hardware front, fundamental changes and advancements are occurring in accelerated computing, beyond-Moore computing, and extremely heterogeneous computing. Supercomputing is poised to undertake the next set of challenges in moving from exascale computing to zettascale computing. An amalgamation of technological offerings such as graphical processing units (GPUs), Field Programmable Gate Arrays (FPGAs), Intelligent Processing Units (IPUs), massive integrated circuit wafers, neuromorphic machines, and various combinations of such hardware technologies are fundamentally transforming the hardware landscape. On many of these new hardware platforms, relatively little is known regarding how to efficiently achieve discrete event execution that has traditionally focused on conventional central processing unit (CPU) and multi-CPU-core platforms. The gap in our scientific understanding creates a major hurdle when the next generation of computing by DOE and other agencies is aimed at applications in their mission.

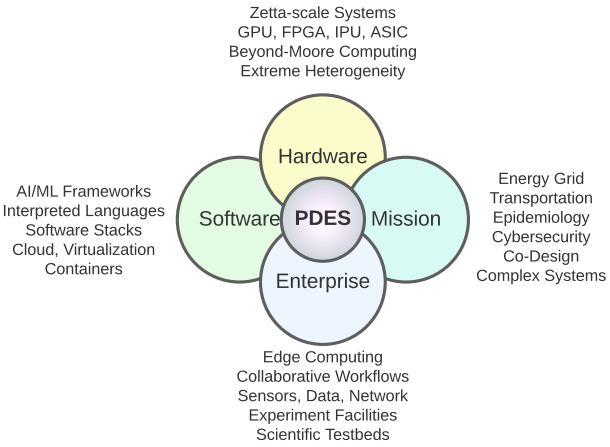


Figure 2.1: PDES in a Changing Landscape

Software: On the software front, a new generation of software technologies has overtaken all aspects of development, deployment and enhancement, with profound implications to high-end computing performance, maintenance, and usability. These include new high performance programming languages, compiled-plus-interpreted frameworks built on mature just-in-time compilation, continuing dissolution of distinctions between fine- versus coarse-grained execution as well as parallel versus distributed execution, and an explosion of open-source and massive software enterprises. The domination of powerful, new tools such as artificial intelligence (AI) and machine learning (ML) libraries or tool-chains are disrupting traditional views of software for high-performance computing. New challenges and opportunities have thus arisen in tapping the power of these software and algorithmic advancements to yield a range of benefits in discrete event execution. Without addressing these key software challenges, advancement will be severely hindered across many important applications.

Mission Applications: On the application front, several mission-relevant areas are emerging that can deliver great benefits when high-performance computing and distributed computing infrastructures can be effectively utilized. There is an increase in the significance of newer, high-end computing-based solutions for national-scale problems such as the resilience of the massive energy grid, transformation of the transportation enterprise including increasingly intelligent mobility infrastructure as well as intelligent carriers such as AI-powered

commercial and consumer vehicles, and the immense significance of non-pharmaceutical interventions and feedback effects in global-scale epidemiological phenomena. Advancements from PDES research will directly enable the development of novel solutions in such computational applications. On the scientific modeling front, PDES can directly benefit the simulation of complex phenomena that are dominated by multi-scale effects such as shocks and magneto-hydrodynamics. Applications are necessarily being recast as fundamentally new model formulations that are discrete event in nature. The actual realization of the benefits of PDES models via efficient computer science methods in these applications represents a great challenge and opportunity to be tapped for the greater good of the society and nation. PDES is the only approach to delivering the required speed and scale in many of these applications, and greatly complements existing methods in other applications as well.

Science Enterprise: The enterprise of basic science itself is undergoing a transformation in the nature of how research and development is performed to tackle the next set of scientific grand challenges. The computational needs of the past were adequately solved with experimentation, data generation and computation in relative isolation or localized settings. By contrast, the scientific enterprise of the immediate future is dominated by many new dimensions. Edge computing, geographically inter-connected workflows, and distributed scientific interactions among scientific groups are changing the way we conceptualize, virtually prototype, test, and validate designs and changes across the science enterprise. Effective exploration, planning, design, prototyping, testing, and iterating through the solutions is fundamentally defined by a set of discrete event models of all the important components in the scientific enterprise. Advances in the DOE scientific enterprise, therefore, are critically reliant on corresponding advances in large-scale parallel discrete event simulations of the large aggregate of the Science Enterprise: scientific instruments, scientific workflows, scientists' behaviors, and high-end computational systems, all of which are interconnected by complex networking infrastructures. The benefits from the next wave of research in PDES are expected from rich collaborations and contributions by the simulation community, the high-performance computing community, the subject matter experts in the application domains, and various policy-making and decision-making communities.

2.2 Priority Research Opportunities

In this new milieu, PDES represents a fundamental core that enables effective solution in the design of the national science enterprise as well as in important mission applications. Advancements from research and development in PDES are needed to fill the gaps in our ability to meet the DOE's scientific mission needs, needs across the national scientific enterprise, and new applications for the society. The rest of this chapter provides a summary of some of the priority research opportunities (PROs) in relation to PDES that need to be immediately pursued in a concerted research program. Chapter 3 provides a brief background on PDES and illustrations of important applications. The technical background and additional details on the PROs are presented in Chapter 4 and Chapter 5.

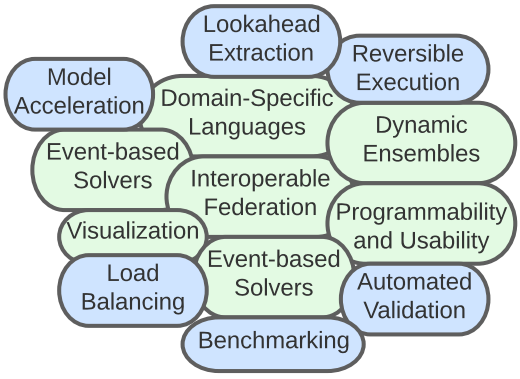


Figure 2.2: Selected PDES Research Items

PRO Category I: Core Advancements (Inner Technologies)

This category of priority research opportunities encompasses the advancements needed in the core methodological and technological aspects of PDES in terms of fundamental algorithms, integration methods, and efficient execution of complex discrete event models on new and emerging hardware architectures. This includes the immense challenge in an effective exploitation of the latest insights and technical leaps in artificial intelligence (AI) and machine learning (ML) as well as adapting to the significantly changing hardware landscape.

1. Even as the mainstream computing hardware is not specifically PDES-oriented in nature, how can new PDES research best exploit the current supercomputing architectures that are designed for non-PDES workloads and applications?
2. How can PDES programming models, libraries, and runtimes be enhanced to improve parallel decomposition, dynamic load balancing, and event scheduling performance? This direction would take into account the following observations.
    - Due to virtual time mechanisms, the relative importance of balancing computation versus minimizing communication is different when compared with those of non-PDES applications.
    - PDES load balancers and event schedulers need to take into account both application characteristics and hardware costs, especially with the awareness that the relative costs of computation, data movement, and synchronization are in great flux on forthcoming computer systems.
    - How can runtime capabilities be improved (e.g., using artificial intelligence) to better determine when and how to load balance and schedule events? In what ways should programming language support and/or library support be augmented to assist the runtime in making better decisions?
    - How can load balancing account for highly heterogeneous PDES models across a wide spectrum of fine-grained to coarse-grained event codes inherently imposed by the application model behaviors.
3. Since PDES performance critically depends on the amount of lookahead available in the model (necessarily for conservative synchronization and supplementarily for optimistic synchronization), new compiler analyses are needed for automated lookahead extraction.
    - What are the limits of new compiler techniques that can extract (virtual) temporal information from model code? What are the fundamental time and space complexities in such analyses?
    - What model augmentation can provide the PDES engines and environments the hooks needed for increased lookahead extraction while not burdening the modeler? What are the language primitives or directives (such as pragmas) that can help in achieving this improvement?
4. What are the novel methods by which the new accelerator-dominant hardware of extant high performance computing platforms (such as GPUs and FPGAs) can be used for executing the logical process-based discrete event models?
    - How can optimistic PDES execution be realized on modes of execution such as streamed Single Instruction Multiple Data (SIMD) computation of GPUs?
    - How can reversible event handlers be efficiently realized on non-CPU-based hardware to enable optimistic modes of PDES execution?
    - Just as OpenMP and CUDA have been successful in efficiently bridging the gap between the programmer and the parallel hardware, how can compilation and automation tools be targeted at automated generation of efficient PDES codes on next generation platforms?
5. In light of advances in AI/ML technologies, what new AI/ML-based dynamic model simplification methods can be developed and exploited to provide leaps in PDES execution, and provide efficient and

intelligent multi-scenario execution? How can AI/ML frameworks be incorporated into PDES core engines and models?

- How can AI/ML methods be used to speed up models, providing the same results but with automatically coalesced event execution that reduces event processing costs? How can models be learned on-the-fly for simplified model substitution or for enabling multi-scale models?
- Within actual runtime implementations of the PDES engines, how can AI/ML methods help inform key decisions made by the engines for synchronization and efficiency in conservative and optimistic settings (such as in minimizing rollbacks or increasing dynamic lookahead)?

6. On the emerging hardware defined by extreme levels of heterogeneity, new hybrid virtual time synchronization methods are needed to be optimized for heterogeneous platforms.
   - What new intra-accelerator synchronization schemes can be developed and used with multiple threads within accelerators?
   - How can intra-node and inter-node synchronization methods efficiently integrate the inter-accelerator synchronization with CPU-to-accelerator inter-node algorithms for correct and efficient virtual time advances across interconnects?
7. What are the new theoretical, temporal ordering frameworks that can be designed and developed to enable the next generation of discrete event modeling and execution?
   - What are the new ways in which under-explored concepts such as approximate time, aggregate time, and time intervals can be developed and applied to scalable PDES execution?
   - How can discrete event engines be designed and implemented to effectively exploit the relaxations uncovered by these new temporal paradigms?
   - How can the implicit conflict between repeatability and concurrency in the presence of bursts of zero-delay events be resolved? What new metrics or frameworks are feasible to assure measures of correctness in the presence of concurrent events (as in processor models in large hardware co-design)?

PRO Category II: Usability Advancements (Outer Technologies)

A major impediment in realizing the benefits of PDES is the high bar the users face with respect to usability. While a sequential discrete event simulator is relatively easy to develop, the complexity of a parallel version increases significantly even for small-scale parallelism (such as for multi-core workstations), let alone for high-end clusters and supercomputing platforms. Furthermore, modeling interfaces, automated support for reversible event computation or lookahead extraction, event-based solvers, and so on, need to be advanced such that the complexity and effort for the scientists are significantly lessened. The ideal target would be fully automated approaches that can shield the user via intelligent compiler analyses, powerful model transformations, and smart code generation techniques.

1. It is relatively rare to find new PDES applications written from the ground up in isolation. As a result, advancements in applications are often made from linking multiple functionalities together. For example, new energy grid simulations combine specialized simulators of transmission grid with renewable energy models or distribution networks. Similarly, transportation networks combine macroscopic traffic models with detailed individual models of smart mobility at the vehicle level or driver level.
   - What support for interoperability of discrete event execution and for federated, virtual time-synchronized execution can be defined and implemented specifically aimed at high-performance

computing platforms?

- How can interoperable and federated execution of multiple simulators be built over standards such as the Message Passing Interface (MPI) for assured portability and efficiency on emerging platforms?

2. How can PDES ensembles be managed intelligently, especially as large scenarios unravel new decision-points dynamically based on the states discovered in the course of the simulation? For example, how can ensemble road transportation simulations be efficiently realized to share computation and memory when incremental changes are inserted as "what-if" scenarios, such as accidents inserted based on congestion levels on routes. Similarly, how can the trajectories of large PDES runs be steered by the user to adjust and/or realize configurations of interest that are hard to anticipate at initialization time?
3. What are the new visualization techniques and systems that can be employed across the PDES stack, from the engine internals (such as event blocking times and rollback behaviors) to the application level (such as integration with sophisticated 3-dimensional animations)? The following categories are important in such visualizations:
   - Detailed visibility into event behaviors and dynamics across processors, with low runtime overheads introduced for visualization and insights
   - Complex rollback behavior display and analyses with automated mitigation insights for optimistic parallel discrete execution
   - Detailed event blocking and idling analyses and insights for conservative parallel discrete execution
   - Interfaces and integration of discrete event engines with visualizers and immersive displays for gaining scientific insights of simulated systems.
4. How can dynamic, streaming data be efficiently integrated into PDES applications? How can the important PDES-specific effects from file system characteristics and interconnection network dynamics be addressed when event-based execution results in highly variable access patterns across the parallel and distributed system?
5. What new modeling interfaces and domain-specific languages can be developed that are more readily amenable to PDES-specific optimizations as compared to using general-purpose languages? What source-to-source compilation techniques enable retaining the best state-of-the-art optimizations of native compilers? Of specific benefit are primitives that are easy to integrate into extant popular programming languages while making it easier to extract lookahead and/or automatically generate corresponding reverse code from the forward event code.
6. Another important area of research concerns PDES runtime support. In order to enable advanced partitioning, dynamic load balancing, and event scheduling policies, the runtime requires support from the programming model (i.e., language and libraries) to accomplish tasks that may be difficult to automatically infer (such as component state serialization). The PDES programming model should be enhanced to make these tasks as easy as possible for the application developer to specify.
   - While the runtime can observe program behavior to make dynamic load balancing decisions, it is still helpful for knowledge of the application behaviors and constraints to be made available to the runtime through the programming interface. This could appear in forms such as nominal connectivity information (such as which hardware components are allowed to send events to which other components).
   - Event scheduling optimizations (e.g., avoiding mis-speculation when the runtime can anticipate future event timestamps) can be enabled through mechanisms that allow the programming model to derive and pass hints to the runtime.

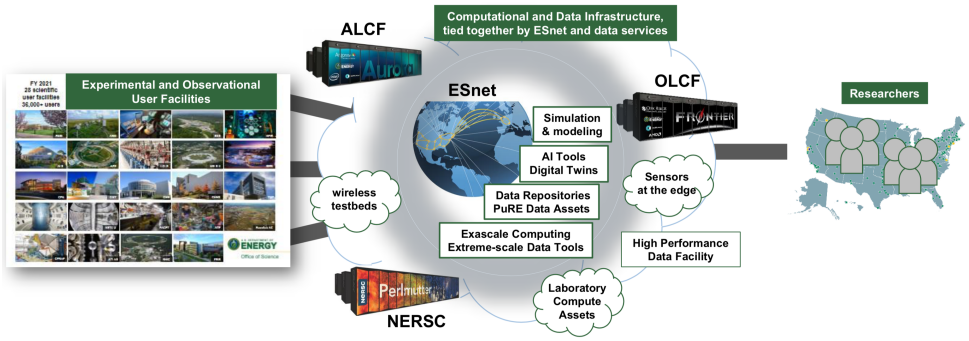


Figure 2.3: DOE Science Enterprise reproduced from Benjamin Brown's presentation to the Advanced Scientific Computing Advisory Committee (ASCAC), 2021 [10]

- Dynamic event aggregation policies can be enabled by ensuring that application code (*what* is being computed) is sufficiently abstracted away from *how* those computations are scheduled and at what granularity in space and time.

PRO Category III: Advancements for the Scientific Enterprise and Mission Applications (Cross-cut Technologies)

In addition to the inner and outer technological advancements, a related set of cross-cut technologies need to be advanced. Failing to make these advancements will severely limit the ability of DOE to broaden the scientific spectrum beyond the traditional basic science applications. Without making progress on these research opportunities, many applications of interest to national security and resilience will remain untapped in the DOE mission space.

Discrete Event Simulations of the DOE Science Enterprise

In the broader vision for DOE's scientific enterprise (see Figure 2.3), a new PDES-based system can provide a national-scale facility by which major additions, changes, or needs can be thoroughly evaluated in a system-scale simulation of the enterprise operation. Feasibility can be confirmed, conformance to constrains can be verified, investments can be vetted, cost trade-offs can be explored, and so on, all in the holistic view of a nation-scale scientific enterprise system encompassing the behaviors of all infrastructural matter and scientific minds.

In order to move towards this vision, research is needed to develop effective discrete event models of the scientific facilities (including the numerous, interconnected cyber-physical processes), computing and networking infrastructures, the scientific workflows, and the (human inquiry-driven) behavioral elements of scientific inquiry. Owing to the scale, size, speed, and complexity of all the factors at play, the underlying problem is fundamentally a distributed and parallel manifestation of many interconnected PDES models. These need to be exercised in the aggregate for evaluating their effectiveness for scientific discovery and innovation.

- What are the model development and repository infrastructures that can result in a distributed set of models for DOE facilities that can be interconnected for holistic evaluation?
- How can we enable each facility to develop and maintain the discrete event simulation models of their own components that are part of the broader DOE enterprise?
- How can the holistic enterprise simulations be hosted and offered to stakeholders across the DOE science and application offices to enable comprehensive evaluations for major installations and innovations that

affect the large user base of the scientific community?

Discrete Event-based Mathematical Solvers

Fundamentally new numerical methods based on highly asynchronous and dynamic updates can be applied in scientific codes (see Section 5.2). However, major research advances remain to be made to reap their benefit in applications.

- In certain classes of *unstable-source* numerical systems, such as fluid systems with large variations in wave speeds, how can PDES make adaptive local time stepping integrators more efficient while maintaining numerical stability?
- By making numerical simulations more work efficient, PDES offers great potential to enable the use of more complex computational methods and analysis. How can the potential be realized in such methods such as discontinuous Galerkin methods, ensemble forecasting, and uncertainty quantification?
- While previous work has demonstrated the impact with low-order explicit methods, how can future research extend the methods to both higher order systems and implicit/iterative methods with irregular spatial convergence properties?

Key Application-Driven PDES Technology Advancements

In addition to the core advancements detailed in the previous PRO, there are additional cross-cutting capabilities demanded by many PDES applications that are waiting to be addressed. As task-specific hardware accelerators become more commonplace, PDES runtimes should provide a mechanism for logical processes to access shared accelerator resources in an efficient manner. For example, in transportation simulation, scientists are exploring how to deploy and optimize distributed traffic controllers that utilize reinforcement learning, which involves a large amount of AI training and inference. The traffic controllers may be executing in parallel on different CPU threads, all submitting tasks to a set of shared offload tensor co-processors. Furthermore, the runtime itself may be submitting AI tasks to the accelerators (e.g., to determine how to rebalance the simulation). In this example, there is a critical need for the PDES runtime to present a unified mechanism for shared access to the accelerators in a way that can prioritize the submitted tasks (e.g. taking events' virtual times into account) to most efficiently utilize the shared hardware resources.

Furthermore, what are some benchmarks that are relevant in that domain that can be contributed to the PDES community for inner and outer technology advancement? A repository of benchmarks spanning application areas such as material science and transportation science is necessary to serve the applications and PDES core proper. Similarly, key proofs-of-concept are necessary to exercise, demonstrate, and enable PDES-based solutions across applications. Simulator engines and actual implementations are needed, along with definition and validation in important scenarios, which, when in place, will be key enablers to advance the specific applications as well as PDES in general.

Chapter 3 Background and Illustrative Application Areas

In this chapter, a brief background on discrete event simulation and parallel discrete event simulation is provided, followed by an illustrative overview of PDES that underlies important application areas of particular relevance to DOE, including the science enterprise, transportation, energy grid, cyberinfrastructure, material sciences, and epidemiology.

3.1 Parallel Discrete Event Simulation

Parallel and distributed discrete event simulation deals with ways of making leaps in virtual time and employing multiple computational resources in a simulation containing a large set of entities defined as discrete event-based models. Achieving correctness of parallel execution requires efficient and accurate synchronization across all processing elements despite the simulation being partitioned spatially or temporally and the partitions being mapped to processors. All processors together serve to collectively simulate an integrated set of application models such that the results are correct and reliable despite each processing element evolving virtual time asynchronously in complex patterns.

To be meaningful, the results produced by a parallel simulation run must ideally match those that could be produced by an equivalent sequential simulation run. To achieve this match, parallel execution must be properly synchronized to preserve the right orderings and dependencies during computation of simulation state across processors. One of the significant challenges in this synchronization is in minimizing the runtime execution overheads (memory, computation and communication) incurred during parallel execution. It is thus important to keep the overhead within acceptable levels, in order for the parallel execution to deliver sufficient value above and beyond sequential simulation.

In this vein, parallel and distributed discrete event simulation techniques have been studied in the past two to three decades. The literature spans books and survey articles covering traditional techniques in parallel and distributed simulation. The interested reader is referred to a few early articles [18, 33, 34, 42] and a few recent publications in this area [3, 35, 43, 70, 81, 84]. Notwithstanding these, recent emergence of new application demands, techniques and hardware platforms are resulting in the need to revisit traditional techniques and to develop newer techniques.

Spatial decomposition is by far the most commonly used parallel simulation scheme. In this scheme, application models are partitioned into logical processes (LPs). Each LP contains its own individual state variables, and interactions among LPs are only via exchange of time-stamped events. The simulation progresses via execution of these events in temporal order. The temporal ordering is either inviolably maintained at every instant during simulation (this approach being known as conservative execution), or is achieved in an asymptotic manner (this approach being known as optimistic execution) in which the system guarantees eventual convergence to the correct overall temporal order.

In general, there is a one-to-one mapping from the modeled physical time to simulation time. In contrast, there may or may not exist a specific relationship between simulation time and wallclock time. The mode of simulation execution determines this particular relationship. In an as-fast-as-possible execution, the simulation time is advanced as fast as the computing speeds allow, unrelated to wallclock time. In real-time execution, on the other hand, the advances in simulation time are performed in lockstep with wallclock time – that is, one unit of simulation time is advanced in exactly one same unit of wallclock time. A variation of real-time execution is

scaled real-time execution, in which simulation time period is some constant factor times an equivalent wallclock time period. Synchronization algorithms are required to provide correct execution, avoiding undesirable effects such as deadlocks, live-locks and termination problems. What more, in analytic simulations, which are executed in an as-fast-as-possible fashion, an important system goal is to minimize the overheads of synchronization such that the simulation completes as faster than real-time as possible. This adds the need for delivering rapid simulation progress, on top of correctness of parallel operation.

PDES Synchronization Approaches: Conservative, Optimistic, Hybrid

Broadly speaking, parallel discrete event simulations have been viewed as conservative, optimistic, or hybrid simulations. Conservative parallel simulations ensure that every event is executed only when the value of the model's state just prior to the event is always correctly ascertained (that is, the read set of an event handler holds the correct, lasting value). Optimistic simulations relax this mode by ensuring that the value of the model's state just after the event will (eventually) be correct prior to the end of the simulation (that is, the read set of an event handler may not necessarily be fully ready before the event is executed). Hybrid simulations employ various combinations of these two approaches, along with a wide set of variants in terms of the amounts and types of knowledge that becomes available in different parts of the system as the simulation progresses, and in different methods of exploitation of such dynamically generated knowledge.

These modes result in vastly different technologies necessary to deliver the best performance on high performance computing systems. A tremendous degree of complexity underlies all the modes, spanning many dimensions such as compiler technologies, programming patterns, domain-specific modeling languages, inter-processor communication, and software engineering.

Among them, the optimistic style of parallel discrete event simulation has lately received a significant amount of attention in terms of a rich set of techniques that add reversibility to model execution. There remain significant challenges and opportunities in advancing them to the new hardware and software in high performance computing systems. At the same time, a comparable degree of attention is also overdue to new conservative techniques that can discover and learn model-level concurrency latent in the models via lookahead-extraction techniques (see Section 4.3), potentially via sophisticated code analyses and machine learning techniques that were not available until recently.

Computational Characteristics of PDES

In general, the computational characteristics of PDES application workloads differ starkly from those of traditional scientific, high-performance computing codes.

Bandwidth and Latency Demands: Due to the inherently bursty and asynchronous nature of computation, PDES applications tend to be significantly affected by network latency parameters and relatively less by network bandwidth parameters. The speed of system-wide distribution of virtual time information is by far the most critical factor that determines the overall efficiency of an PDES application. The next critical parameter is the speed with which inter-LP events are exchanged, which not only limits the progress of the simulation on each processing element but also indirectly affects the virtual time advancements that account for transient events across processors.

Floating Point versus Fixed Point Operations: Generally speaking, the computational core of PDES applications involves event handling codes that are not as dominated by floating point operations as scientific codes tend to be. Moreover, the most frequently encountered operations involve integer operations, as well as

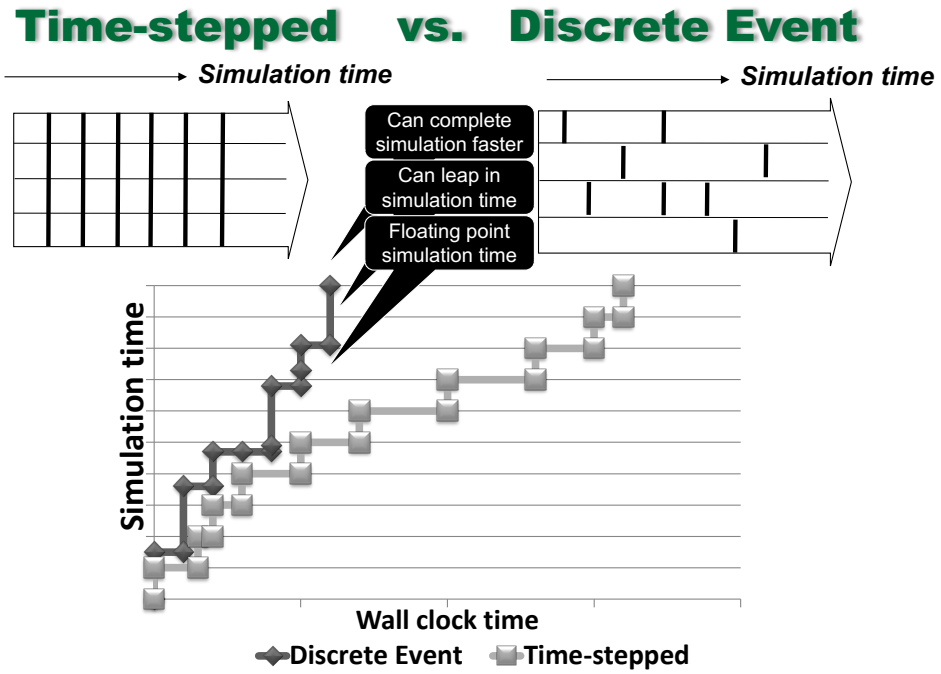


Figure 3.1: Illustration of time-stepped and discrete event style of modeling and execution

complex, linked data structures. For example, cyberinfrastructure simulations involve network packet transfer models dominated by integer operations; similarly, enqueing and dequeing operations dominate vehicular and manufacturing models. This is in contrast to the computationally-heavy floating point operations involving large matrices in some of the traditional scientific computing codes.

Simulation and Wall-clock Time Dynamics: In traditional time-stepped execution, simulation time is generally advanced via (fixed) time increments, and the system state is updated in a global fashion while the simulation time is frozen at a certain value after a time increment. In contrast, PDES execution is dominated by arbitrarily variable simulation time advances both within as well as across processing elements. Figure 3.1 shows the salient differences in the execution styles. PDES allows floating point simulation time values, offers highly asynchronous time leaps, and can generally complete the simulation faster. However, these also translate into highly variable network communication, congestion, and buffering demands that a PDES application places on the runtime system.

Computational Elements: Many scientific computing codes involve the solution of coupled partial differential equations, implicit/explicit solution methods, linear algebra solvers, memory management, and so on, which have been well studied in the high-performance computing domain. However, PDES is fundamentally built on less studied concepts such as fine-grained event codes, timestamp ordering, global virtual time computation, lookahead, rollbacks, state-saving, reversal, fossil collection, and so on, which directly determine the efficiency of PDES application as a parallel code (see Figure 3.2).

Modeling Elements: Table 3.1 shows the common modeling elements in PDES in comparison with those in high-performance computing applications that are broadly based on time-stepped simulations. While many scientific codes are built on concepts such as mesh elements of complex geometries, PDES applications are based on the notion of logical processes (LPs). Similarly, while messages and shared arrays dominate many scientific codes, communication in PDES applications is defined in terms of events which are time-stamped messages that constrain the global time-ordered execution of the entire parallel run. Modeled phenonema in PDES tend to be

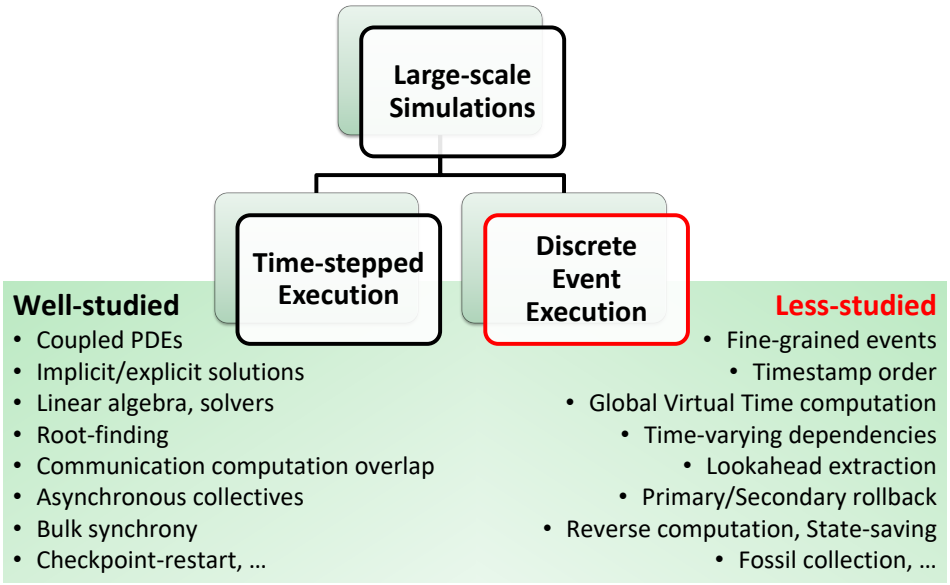


Figure 3.2: Illustration of differences in basic concepts underlying parallel discrete event from traditional high performance computing applications

	Traditional Supercomputing (Time-stepped) Simulations	Parallel Discrete Event Simulations
Modeling Units	Mesh elements, etc.	Logical Processes (LPs)
Inter-unit Communication Support	Messages, Shared arrays	Time-stamped Messages (Events)
Virtual time advances	Periodic, regular, mostly static	Highly dynamic, irregular
Computational concurrency	Across space, at frozen virtual time (time step)	Staggered across time and space (event timestamps, and LPs)
Modeled phenomena	Predominantly physical	Predominantly anthropogenic
Modeled inter-entity interactions	Often structured in space and time	Predominantly unstructured in both space and time
Scientific basis	Focused compositions of well-understood basic laws	Exploratory compositions of evolving views/insights

Table 3.1: Representative dimensions of traditional high performance computing applications and parallel discrete event applications

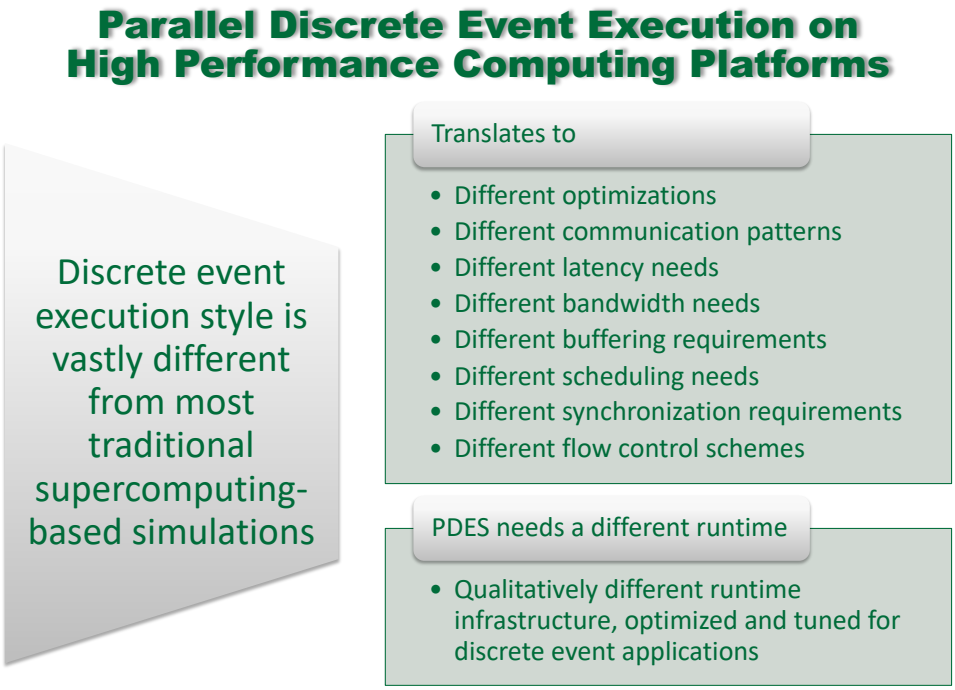


Figure 3.3: Implications of parallel discrete event execution on the underlying system features of the high performance computing platform

predominantly anthropogenic in nature, rather than defined by physical system laws. Inter-entity interactions are nearly always unstructured in nature, in direct contrast to the physically dimensioned structures arising in many scientific codes. Note that the unstructured nature is different from that encountered in scientific codes in terms of unstructured meshes. *Technically, the bandwidth of the graph formed by the network of entities (and their interdependencies) in many PDES applications tends to be proportional to the number of entities in the graph, in contrast to the relatively low bandwidth that can be obtained in a graph defined by unstructured meshes.*

From the standpoint of high-performance computing systems, PDES constitutes a qualitatively unique class of applications (see Figure 3.3). Its characteristics translate to a different set of optimizations, communication patterns, latency needs, buffering requirements, scheduling constraints, synchronization and flow control needs at the levels of hardware and middleware of the system. A layered schematic of the PDES software stack is shown in Figure 3.4.

3.2 Science Enterprise Design and Provisioning

The DOE high performance computing and networking facilities provide world-leading capabilities that act as the substratum to interconnect and enable geographically distributed, collaborative science using the most advanced scientific user facilities, leadership computing, and production computing (see Figure 2.3 on page 7). Some of the fastest and broadest networking superhighways tie them together as a seamless whole. The next leaps in scientific advancements are enabled by unprecedented levels of collaborative scientific workflows realized over this large scientific enterprise. An excellent portrayal of this vision can be found in the presentation [10] by Benjamin Brown et al titled “Vision of DOE Science/User Facility Eco-system” in the DOE Advanced Scientific Computing Advisory Committee (ASCAC) meeting on September 29, 2021.

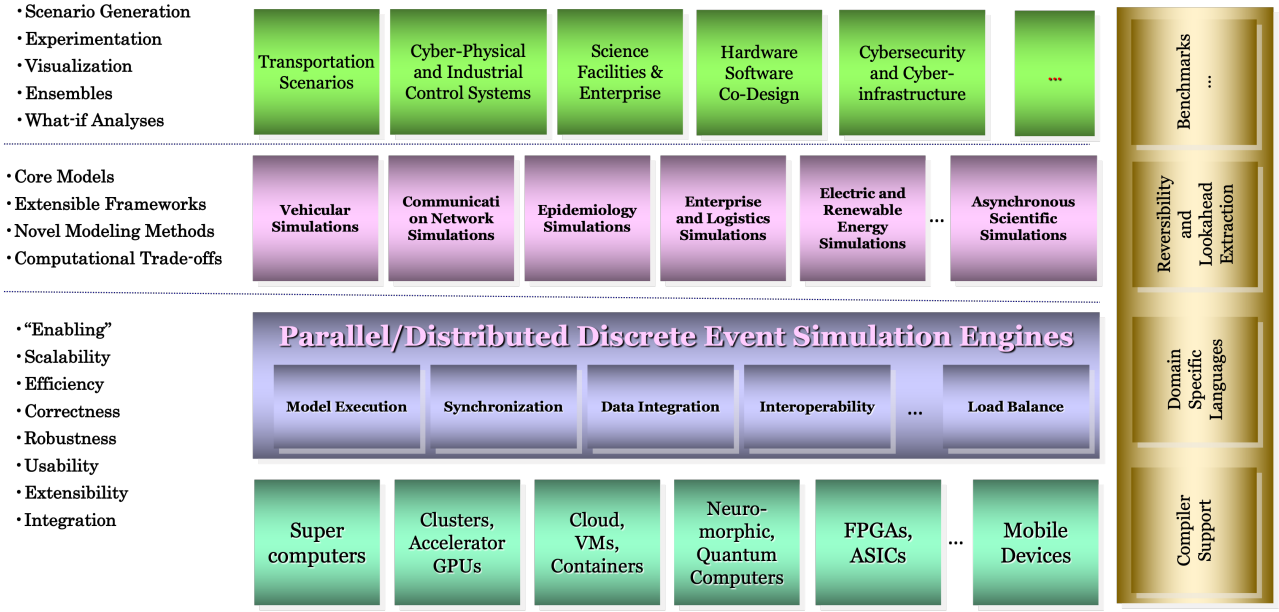


Figure 3.4: The hardware and software stack of PDES technology components and applications

Risk

A critical dimension that is emerging in this rapidly evolving scientific enterprise is the immense complexity of composition of all the elements at play. While the *physics* of the enterprise is made of the scientific instruments, computing, networking, and storage, the *behaviors* of the enterprise compound the complexity in terms of the collective scientific minds pursuing dynamic, insights-driven, temporally and geographically distributed scientific inquiries (see Figure 3.5).

Although it is relatively easier to conceive the concept of operations in a localized setting of scientific studies (say, at the level of a single facility within a laboratory), the newly expanded scientific enterprise is extremely difficult to conceive and design without a methodical, holistic, and thorough simulation-based exploration of solutions. The high level of risk in major decisions for the enterprise can be significantly mitigated and educated via a holistic PDES-based evaluation of the enterprise.

System-scale simulation is helpful in not only covering the scientific activity space in a thorough manner, but also most efficiently react and provision in an evolving landscape. It can be used in answering many types of questions such as: How does a new planned installation impact the rest of the enterprise? What effects do changes in scheduling policies or down-times have on different parts of the enterprise? This is akin to national-scale air traffic simulations that are necessary to help predict the design, planning and responses to dynamic air traffic scenarios, especially due to the sheer complexity and scale of the system. The risk of not relying on a holistic simulation of the science enterprise lies in missing less expensive solutions and operating with an inadequate understanding of the full design space, which leads to slower scientific advancement, reduced productivity, and a predominance of local optimalities,

Vision

In the broader vision for DOE's scientific enterprise, a new PDES-based system can provide a national-scale facility by which major additions, changes, or needs can be thoroughly evaluated using a large, system-scale simulation of the integrated operation of the entire enterprise. Feasibility can be confirmed, conformance

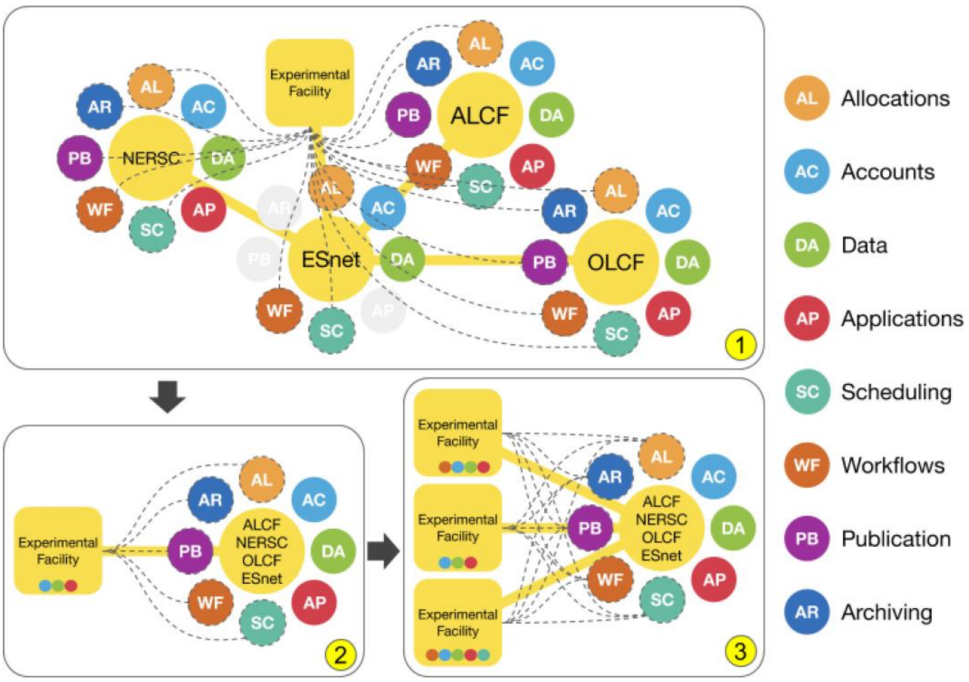


Figure 3.5: DOE Computing Facilities reproduced from Benjamin Brown's presentation to the Advanced Scientific Computing Advisory Committee (ASCAC), 2021 [10]

to constrains can be verified, investments can be vetted, cost trade-offs can be explored, expansions can be planned, effects can be stress-tested, and many additional analyses can be achieved, all in the holistic view of a nation-scale scientific enterprise system encompassing the behaviors of all infrastructural matter and scientific minds.

The problem of simulating the DOE science enterprise is fundamentally the challenge of discrete event modeling and simulation of its full complexity: scientific experiment models, buffers, staging, delays, jitters, storage, networks, volumes, velocities, insight generation, documentation, software, security, authentication, provenance, archiving, etc. In order to move towards realizing this vision, research is needed to develop effective discrete event models of the scientific facilities, computing and networking infrastructures, the scientific workflows, and the behavioral elements of scientific inquiry. Owing to the inherently large sizes and high speeds of operations, the result is a large PDES execution for which DOE will be directly responsible to establish and sustain as a critical service to the science community.

3.3 Transportation and Mobility Applications

Metropolitan-scale transportation systems are becoming increasingly complex and interconnected with the integration of ubiquitous sensor, communications, and control technologies such as real-time GPS and camera data feeds, 5G communications networks, and intelligent infrastructure and vehicle behavior controllers. The resulting system consists of a collection of distributed agents (e.g., vehicles and infrastructure controllers), each making different decisions such as how to optimize either their individual utility (e.g., user travel time) or the system efficiency (e.g., minimize overall congestion or system fuel use). Control strategies implemented through traffic management can include infrastructure modifications such as signal phase timing adjustments of signals to expedite congestion mitigation. With the introduction of autonomous vehicles, an even greater opportunity is expected for agents to make frequent decisions in response to system dynamics.

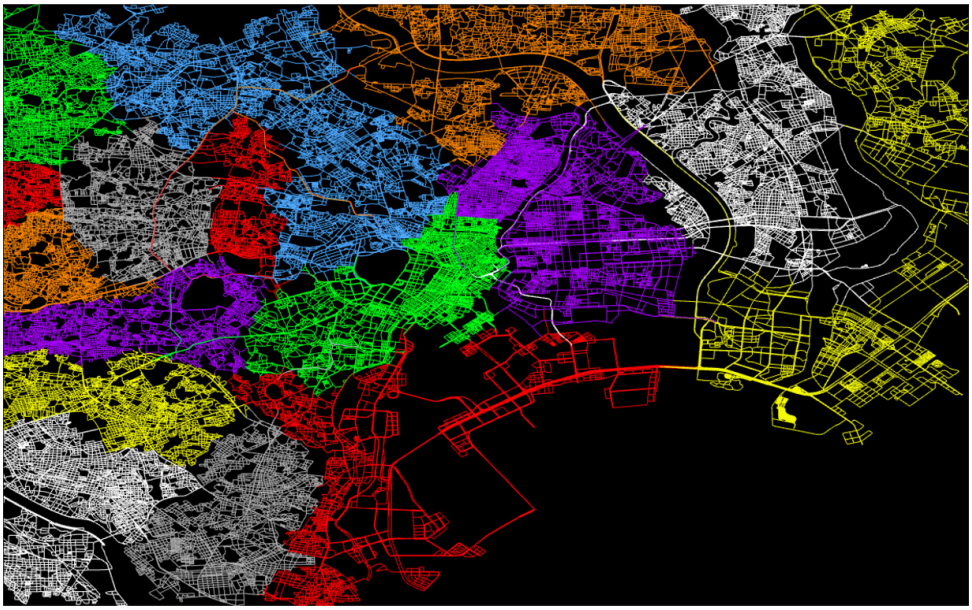


Figure 3.6: Tokyo road network partitioned into regions for mapping to a multi-processor parallel discrete event execution [36]

Transportation Simulation Systems

The rapid introduction of distributed, intelligent control technology makes it increasingly difficult for researchers and government transportation agencies to understand and predict the dynamics of congested transportation systems. Traffic network simulation is a key capability for these organizations to analyze different scenarios and predict the potential impacts of infrastructure changes, policies, or control strategies *at scale*. PDES is a key computational capability that enables modeling and analysis of these complex systems at metropolitan scale involving millions of agents, nodes, links, etc.

Transportation system simulators can be broadly classified as macroscopic, mesoscopic, and microscopic based on the level of detail of the behavior of individual agents simulated. Macroscopic models typically rely on continuous flow approximations of traffic rather than modeling the behavior of discrete vehicles. While macroscopic models can capture bulk, aggregate behavior of traffic networks, they are insufficient to model the detailed, discrete agent behavior that are characteristic of dynamic, intelligent control systems. In contrast, mesoscopic and microscopic models can resolve the behavior of individual vehicles, but their computational cost has traditionally limited their applicability to smaller geographical regions. Simulators such as TRANSIM [66], MATSim [59], POLARIS [2], DynaMIT [4], DynaSmart [58], SCATTER [79], SCATTER-OPT [124], Aimsun [14], SUMO [57], INTEGRATION [89], BEAM [101], Mobiliti [17], MANTA [123], and others [113] fall along the spectrum of different modeling approaches, target problem scales, modeling fidelities, and outputs. Many of these simulation tools are functionally sequential, and some of the parallel simulators utilize a fixed-time stepping method. The efficacy of parallel discrete event simulation has been demonstrated to varying degrees [2, 17, 66, 101, 124]. Some of these tools utilize conservative synchronization (e.g., TRANSIM, POLARIS, BEAM, SCATTER), while others utilize an optimistic approach (e.g., SCATTER-OPT and Mobiliti). Improving parallel scalability of these simulation tools is an important area of research, as domain scientists and engineers aim at simulating ever larger systems with significantly greater traffic volumes.

Challenges

There are a number of important computational challenges that need to be addressed to further enable transportation scientists to utilize PDES technology. These include:

- Partitioning for efficient load balancing
- Dynamic load balancing
- Efficient mapping of microscopic models and inter-entity shared states to new high-performance systems such as those that make use of accelerator-based computational kernels and memories.

As with other PDES domains, performance is dependent on the model representation as a collection of logical processes (LPs), their associated execution events (compute and communicate costs), and their mapping to compute resources. An important aspect of the mapping problem is the parallel domain decomposition to balance the work of the simulation across compute resources. Efficient partitioning involves recognition of modules of an application process that could run concurrently, with minimal, infrequent interaction between each other (see the illustration in Figure 3.6). In particular, zero-lookahead interaction is to be avoided for parallel simulation efficiency. A common domain decomposition of the traffic system partitions the road network into subgraphs and assigns each subgraph to a compute resource.

The domain decomposition and load balancing problem is challenging because traffic systems with complex control can contain a variety of heterogeneous agents, each with their own computational requirements and characteristics. Examples of agents in such a system include link actors that are responsible for computing congestion that occurs among vehicles simultaneously occupying a link, vehicle rerouting controllers that are responsible for computing new routes for vehicles based on current congestion patterns, and dynamic traffic signal controllers that are responsible for changing the signal timing of an intersection or set of intersections. The computational cost of executing events for these different agents may be highly variable both spatially and temporally, depending not only on the type of LP and event (e.g., status updates versus shortest-path calculations versus neural network inference) but also on the dynamic state of the system. As a result of this variability, the resulting simulation is difficult to statically load balance and requires novel dynamic load balancing methods to meet the scaling needs.

Another reason to utilize dynamic load balancing is due to the movement of vehicles across the road network. In some implementations, the vehicles flowing through the system may not need to be represented by LPs themselves, but can rather be represented as events that are passed between the road links [17, 124]. The advantage of such an approach is that the communication pattern between road link LPs has very good spatial locality and remains stable during the simulation, even as vehicles travel from one end of the road graph to the other. However, if vehicles need to serve as direct destinations of events (rather than indirectly through their current link), vehicles need to be instantiated fully as LPs. In this case, if the vehicle LPs were partitioned statically, it would result in much more non-local message traffic as the vehicles traverse the road network and need to communicate with road segments and vehicles far away from their initial locations. A dynamic load balancing approach takes the vehicle's changing simulated location into account when determining to which compute partition to migrate the vehicle's LP, thus helping minimize the physical distance that messages have to travel between LPs that interact with one another in the simulation.

In addition to partitioning for spatial locality and dynamic load balancing, there is a question of how to optimally leverage accelerators for transportation PDES. Several classes of agents process different types of events that benefit from hardware acceleration. For example, microscopic models may be mapped to GPUs, while traffic signal controllers that utilize deep reinforcement learning could leverage Deep Neural Network

(DNN) acceleration hardware. The efficient management and shared utilization of these hardware accelerators and their memories by multiple LPs in the context of a distributed PDES simulation is an important topic of investigation for future research. Also, mesoscopic models [86] that are suitable for computation on accelerators need to be explored for exploiting the power of the emerging hardware systems.

3.4 National Energy Grid Applications

Distributed energy resources with increasing solar and wind energy penetration introduce bi-directional electric power flows mixing the transmission and distribution functionalities of the energy grid. The large size and complexity of electrical power systems demands the application of high performance computing in simulating the hybrid models of the changing energy grid.

Hybrid Models for Grid

The simulation of hybrid models plays a leading role in the design of control systems for the power grid in the large. Historically, industries adopting highly automated systems have motivated and promoted research in hybrid models and their simulators; manufacturing and aerospace are prominent examples. The distributed, automatic controls that are intrinsic to smart grids and renewable energy grids are the motivators for a new wave of research in modeling and simulation of hybrid systems.

Hybrid models emerge from the study of interactions between a system's digital and analog components. The continuous dynamics of analog components are modeled with differential-algebraic equations. Discrete event models are used to describe the dynamics of digital components. The interaction of these discrete event and differential-algebraic models plays a central part in simulations of the complete system.

A simulation of frequency regulation by loads provides one example of a model containing continuous and discrete event components [61, 63, 111]. Consider, in particular, a model used to select design requirements for the sensors. At each load, a digital controller watches the frequency of the power system where it is installed. The sensor in the controller has finite precision, and so acts upon only discrete changes in frequency. A typical sensitivity is $\Delta f = 0.005$ Hz [112]. When the sensor reports a change in frequency, the controller changes the impedance of the load (e.g., by turning electrical equipment on or off) at its location. In the time between these control events, the electro-mechanical aspects of the power system evolve continuously as described by their differential-algebraic model. One variable in this model is the frequency observed by the sensors. *Simulation of the interacting dynamics of these two models — one discrete event and the other continuous — is necessary to predict the effectiveness of the control scheme.*

Discontinuity locking is a technique central to the discrete event simulation of hybrid systems [16]. With this approach, interaction between the discrete event and continuous model occurs at the roots of state event functions. In the preceding example, the continuous model contains a variable f_k that is the frequency at the k^{th} bus; the sensor has a variable n_k that is the frequency level at which the most recent control event took place at the k^{th} bus; and the next discrete event at the k^{th} bus happens when $f_k - (n_k \pm 1)\Delta f = 0$. This function is an example of a state event function; by construction, it is zero at the event and changes sign upon crossing zero. During the simulation of the differential algebraic model, all the discrete variables are read at their present values. In this case, the n_k terms are kept constant while simulating the electro-mechanical dynamics. However, at each step of the numerical scheme that solves the differential-algebraic equations, the values of the state event functions are calculated. If the sign of any these functions has changed, then the simulator missed an event (or,

in general, at least one event). It then looks with a root-finding procedure for the precise location in time where the state event function changed sign.

At best, the location of the event is found in just one extra evaluation of the differential-algebraic equations. More often, several solutions to these equations must be calculated even as the root finding procedure narrows the time interval containing the event. When the event is found, the discrete event is applied — in our example, the admittance at the bus is modified — and the numerical algorithm restarted at the time of the event.

Computational Needs for Discrete-Continuous Models

The example just described highlights three key aspects of discontinuity locking that contribute to a high computational cost for simulating hybrid models. The first is that the root finding procedure necessitates a large number of evaluations of the differential-algebraic equations. Secondly, the frequency of events places a potentially severe constraint on the step size of the numerical scheme that is used to simulate the continuous model; this also contributes to a very large number of evaluations of the differential-algebraic systems.

Third, frequent discontinuities exhibited by discrete events prohibit in practice the use of multi-step numerical methods. Although a number of multi-step numerical methods are available for optimized simulation of electric networks, they cannot be used for discrete event execution, owing fundamentally to their closed-system treatment and/or continuity at irregular points along the system trajectory. While the ability of multi-step methods to reuse calculations from previous time steps is useful to improve the accuracy of the numerical solution in next steps, they cannot be used to advance the simulation along irregular time steps. Instead, single step methods that are easy to restart are preferable; for example, those in the Runge-Kutta family. These require multiple evaluations of the differential-algebraic equations to calculate a single point in their trajectory; once again, this increases the number of evaluations of the differential algebraic equations.

To examine wide area control of electro-mechanical transients by smart devices, simulation of electro-mechanical dynamics at the transmission level are essential. Though previous work in power system simulation has addressed this computational problem in a classical setting [41, 44, 102], the introduction of significant discrete event dynamics necessitates a new approach.

With regard to the differential algebraic equations that model the electro-mechanical dynamics, it is the solution of the linear system relating voltages and currents of the transmission network that poses the greatest computational challenge. In particular, we must address two problems: (1) the frequent refactorization of the admittance matrix as required to model some types of control events, and (2) the large numbers of back-substitutions imposed by discontinuity locking.

Challenges

New algorithms are needed for solving this problem in the context of PDES. The broad framework for discrete event-based hybrid simulation is shown in Algorithm 1. The new algorithms need to be designed to fit into such simulation frameworks and reduce execution times for large-scale models from hours to a few minutes as demanded by practical design tools. By large-scale is meant models with thousands to tens of thousands of buses, each with its own discrete sensors and actuators.

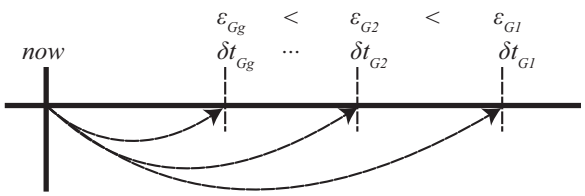
The simulation of large-scale, discrete event-based hybrid models of the evolving grid raises challenges and opportunities in mapping them to high performance computing.

- How can the hybrid discrete-continuous models be mapped for efficient execution on accelerators such as GPUs?

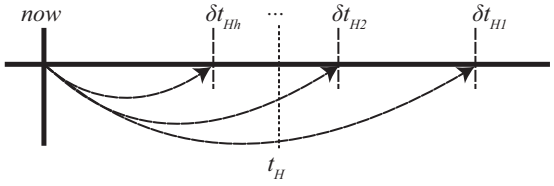
Algorithm 1 Discrete event-based execution of energy grid simulation [83]

1: $now \leftarrow 0$
2: **while** $now < end\,time$ **do**
3: $\delta t_{min} \leftarrow \min(\delta t_{integrator}, \delta t_{threshold}, \delta t_{external})$
where, the δt values are determined as follows:

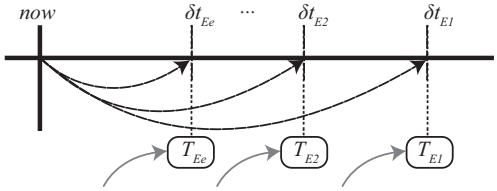
(a) $\delta t_{integrator} \leftarrow \delta t_{Gg}$ for the largest time leap, δt_{Gg}, that gives an acceptable error ε_{Gg} in numerical integration of the grid system state from now to $now + \delta t_{integrator}$



(b) $\delta t_{threshold} \leftarrow \delta t_{Hh}$ for the earliest time leap δt_{Hh} near the earliest time t_H at which the system state (e.g., voltage) crosses a threshold value specified by a model component (*e.g.*, automated control from smart devices for voltage shedding)



(c) $\delta t_{external} = T_E - now$ for $T_E = \min_i(T_{Ei})$, which is the earliest of all times $\{T_{Ei}\}$ at which an externally-specified system change is scheduled to occur (*e.g.*, outages due to non-electrical causes)



4: Advance the electric grid state from now by δt_{min}
5: Incorporate electric device control effects in the interval now and $now + \delta t_{min}$
6: Incorporate effects of external events, if any exist with time stamp $\leq now + \delta t_{min}$
7: Advance the simulation time: $now \leftarrow now + \delta t_{min}$
8: **end while**

- How can the models be efficiently distributed across multi-accelerator, multi-node architectures while achieving correct, time-synchronized evaluation for large-scale grid scenarios?
- How can linear algebra solvers be efficiently integrated into the discrete event execution in order to advance the grid state across multiple distributed discrete events?
- How can the graph network of the electric grid be efficiently partitioned and mapped to the parallel system?
- What is the most effective incorporation of sparse matrix algorithms within the discrete event-dominant hybrid simulations?
- How can the granularity of discrete event-initiated updates be increased so that the overheads of discontinuous state updates of the grid model states from events are lowered.
- What novel discrete event-based mathematical solution techniques (such as quantized state solvers) can be designed and implemented for the fastest PDES-based grid simulations? How can such novel methods be best realized on accelerated computing platforms?

3.5 Internet and Cybersecurity Simulations

The complexity of communication networks and their associated protocols has grown to the point where an accurate understanding or analysis of all but the most simple aspects and behaviors requires simulation. Network simulators replicate the interactions of protocols, applications, and equipment, including the propagation of electrical signals through a wire, or radio signals through an environment. Accurate network simulation enables testing various applications and protocols at assorted scales, under a variety of conditions, yielding reproducible results. No large-scale network testbed can offer all of that, thus large-scale high-fidelity network simulation is a critical need.

Fundamental Discrete Event Nature

Nearly all current network simulation environments use discrete event simulation methods to simulate the behavior of a communication network. Only certain states of the system being simulated (the network) need to be tracked as they evolve over time. The state of the system is sampled at discrete points in time, and one can safely ignore the state of the system at times in between those discrete points. As an example, consider simulating the transmission of a network packet from one router across a wired point-to-point link to another router connected to the same link. In the actual physical system, this seemingly simple act is in fact quite complicated, including encoding the individual bits into electrical or optical waveforms, calculating error detection and correction information, transmitting these waveforms on the actual communications link, detecting the waveforms at the receiving end, verifying the correctness of the received signal, and reconstructing the packet at the receiving router. However, in a network simulator, the overall behavior of this activity is captured by simply noting that the transmitting router starts sending the packet at some time T, and the packet is completely received by the receiving router at time $T + \Delta t$. The Δt in this case represents the time it takes the sender to transmit all of the individual bits in the packet, plus the the time it takes for the signal to propagate from the sender to the receiver, the signal propagation delay. The state of the system at any time during the interval Δt varies in the physical system, but can be safely abstracted out for this simulation. Using this approach, the state of the system only needs to be changed twice in this example; once when the transmitter has completed sending the packet, and once when the receiver has received the entire packet. Each state change is realized as an event in the discrete event simulation.

The main motivation for implementing PDES in a network simulator is to reduce run-times for large-scale, and/or complex simulations. Without parallelization, some large-scale high-fidelity network simulations require weeks (or longer) to simulate the traffic occurring over short periods of time. Another motivation for PDES is to enable the simulation of large or complex models that require more memory than is available on a single computing element. Yet another reason for employing PDES is the need to integrate with each other multiple simulators, each of which is specialized in a specific networking technology (e.g., wired and wireless network simulators).

Time-stepped versus Discrete Event Network Simulation

In regard to advancement of simulation time, network simulators generally use discrete event simulation (DES) rather than discrete time simulation (DTS). Discrete time simulations divide the simulated time into thin uniform slices. All the events that occur during a time slice are processed, and then the simulation proceeds to the next time slice. In contrast, in discrete event simulations, after processing one event, the simulation time

advances to the time of the next event, which is not constrained in any way to be a fixed time increment from the previous event. One reason communication network simulators favor this next-event time progression over a fixed-increment time progression is that determining an effective fixed step size would be tricky. Because events can generate new events, one would need to determine a time step that would ensure any new events are scheduled in the next time slice. The time increment would likely be very small, and would need to be fine-tuned to optimize each different simulation. The times of the events in many network simulations are not uniform, and are often sparse. As such, a discrete time simulation of a network may incur a large penalty when it faithfully steps through each time slice in a simulation where there is a large gap between events. There would effectively be no change in the simulation state between the gaps, yet every time slice would be processed. In contrast, a discrete event simulation would simply advance to the next event, and instantly skip over the large gap. Discrete time simulations work better for applications where a time step value is obvious, where many independent events can be processed at each time step, and where event distribution is somewhat uniform without large variability in time gaps.

One of the widely-used open source network simulators is ns-3 [32, 68, 87, 90], a discrete event-based network simulator, written in C++ with Python bindings, popularly used for research and education in any networked or Internet systems. A discrete-event network simulator such as ns-3 maintains a priority queue, where events are sorted by their times such that an event with the earlier time will be processed before an event with a later time. Events are usually associated with specific objects. In ns-3, the events are essentially function calls on objects in memory. The simulator dequeues the earliest event from the priority queue and advances its current simulation time to the time associated with that event. It processes the event, changing the system state and typically adding one or more new events to the queue (set to be processed at specified future simulation times). Then, the process repeats, once again advancing the simulation time, processing the next event, and so on. Events generally relate to a simulated object, such as a network device. Network device A, for example, may add an event to the queue representing the receipt of data on network device B, at a future time calculated to mimic the time required to transmit and propagate the data.

PDES-based Network Simulation

In PDES-based network simulators, the simulated network devices are each assigned to one of the multiple computing elements (or to logical processes which are in turn mapped to computing elements). Each computing element maintains its own event queue, processing events for the simulated nodes allocated to it. Each computing element runs in parallel, asynchronously advancing simulation time at its own pace. The simple example of a network device A now becomes more complicated if network devices A and B are managed by different computing elements. Suppose network device A is simulated on computing element a and network device B is simulated on computing element b. Network device A cannot just add an event intended for network device B to computing element a's event queue because network device B relies on a different event queue. Instead, a (virtual timestamped) message must be sent from computing element a to computing element b, requesting that the new event be added to computing element b's queue. Only in this way, will network device B be able to act on the event (and simulate the receipt of the data) at its ordained virtual time in the correct temporal order. However, this raises another potential issue. Since computing element a and computing element b are proceeding at their own pace, it is possible that the message (containing the event for device B) may arrive at computing element b after it has already advanced beyond the time specified in the event. In that case, the simulation remains no longer accurate, as it has missed an event. This example illustrates the need for

synchronization between the computing elements in the parallel discrete event simulation. Two main styles of synchronization can be used: conservative and optimistic (also known as speculative).

Conservative PDES-based Network Simulation

In one of the conservative synchronization schemes called "granted time window," assume that there was a guarantee that no remote events (from other computing elements) would arrive at a computing element before a specific virtual time in the future. The computing element would then be free to process all the events on the event queue up to the given virtual time, with the assurance that it will not receive an event for a simulation time in the "past." Suppose further, that once all computing elements had reached the given simulation time, they could synchronize, exchanging events that need to flow between computing elements, and then determine a new future time with the same guarantee until when they could safely run. This is the granted time window synchronization scheme, and the time to safely run until is called the granted time. But how is the granted time determined? The catch is that, to use this scheme, a lookahead value must be determined which represents the minimum delay between the sender and the receiver for inter-processor events. In other words, lookahead must be set to a value smaller than the shortest amount of time it could ever take to physically communicate between simulated entities allocated to different computing elements. In the case of a network of point-to-point links, the lookahead time can be set to the smallest latency of the point-to-point links plus the transmission time for smallest packet that the simulation sends. If network device A on computing element a, for example, sends a packet across the point to point link connecting it to network device B on computing element b, where the link latency (or delay) is 0.006 ms and the transmission time of the smallest packet expected to be sent is 0.005 ms, the lookahead could be set to any value less than the sum $0.006 + 0.005 = 0.011$. In this case, even if network device A sent an event to network device B immediately after the last synchronization, it would not be scheduled for receipt on B for 0.011 ms, which is after the next synchronization, ensuring it will be placed on the queue, preventing the simulation time from advancing past it. Rather than just advancing in time slices, the earliest next event time across all computing elements (or logical processes) is computed. This value, also called the Lower Bound on incoming Time Stamp (LBTS), is added to the lookahead at each iteration to produce the new granted time. Since synchronizing across all the computing elements is expensive, the key to good performance using this scheme is to maximize the lookahead.

Optimistic PDES-based Network Simulation

In the other main style synchronization called optimistic synchronization, all the computing elements run in parallel, optimistically assuming there will be no issues with events arriving from other computing elements after they should have been processed. However, as they run, they store all the details of the previous messages received and processed, and the resulting state changes. When an event is received for a simulation time in the future, it is simply added to the event queue as usual. When an event is received for a simulation time in the past, the computing element uses a restoration mechanism (such as memory log or reversible computation) to undo all the changes incorrectly made by events to the state of the simulation until the state matches what it was just before the simulation time of the incoming event. The event is then placed on the event queue, and simulation restarts, processing this event, and then re-processing all that followed it. This process is repeated, thus advancing the simulation optimistically and correcting the state on the fly as required. Rollback is the fundamental method of synchronization in this case, rather than blocking the computing elements for guarantees of correct state before events are processed. Optimistic PDES does not require a lookahead value and overcomes

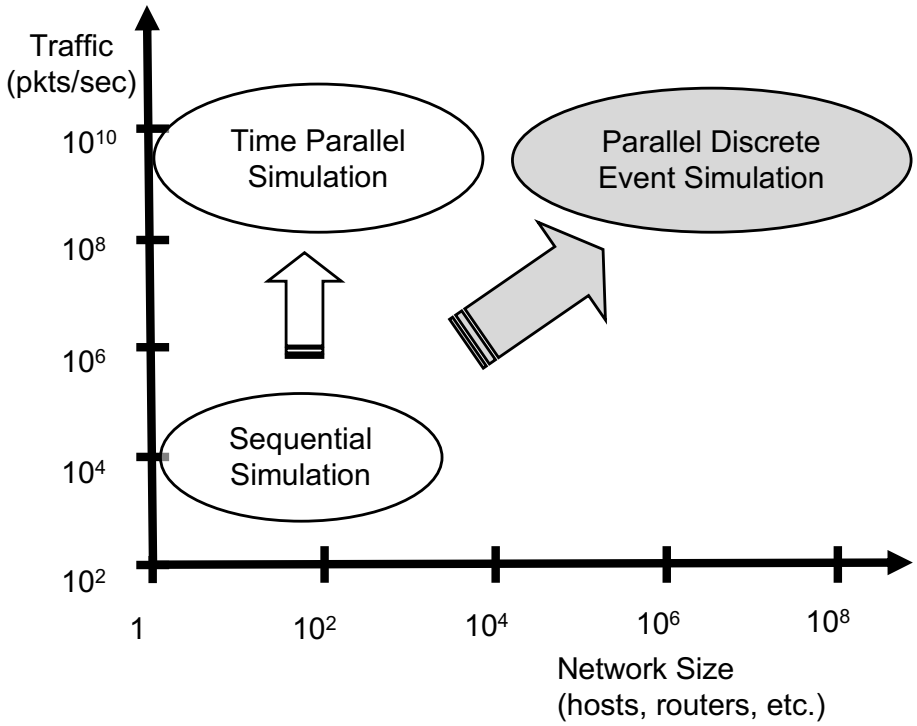


Figure 3.7: Scope of parallel discrete event simulation in network modeling and simulation problems at large scale [32]

the restrictions of conservative PDES. However, implementing the ability to rollback is extremely complicated.

In particular, simulating a wireless network link between mobile nodes using PDES raises specific challenges. Unlike a wired point-to-point link, which has an appreciable and predictable lower bound on the inter-link delay, the wireless link propagation time is proportional to the distance between the mobile nodes, which is not only highly variable but also essentially lower-bounded by zero. Setting a lookahead would require knowing or ensuring that the mobile node separation distance never fell below a certain level, which in most scenarios is not admissible. For this and other reasons, many simulators such as ns-3 limit support for PDES on wired point-to-point links. There is a critical need for an open source network simulator that supports PDES for wireless connections, enabling large-scale, high-fidelity wireless network simulations possibly in combination with wired network models. Enhancement of simulators like ns-3 to support optimistic PDES is one way to achieve this goal. Such an undertaking, however, may require a large effort by teams from multiple institutions (both government and academic) working together to succeed.

Network simulators can be evaluated in a spectrum of speed versus scale. The speed is represented by the aggregate network traffic simulated in terms of packet transfers per second, and the scale is represented by the number of network entities simulated in terms of thousands or millions of hosts and routers active in the simulated network. In this spectrum shown in Figure 3.7, sequential simulation is situated in the bottom left corner while specialized techniques such as time parallel simulation could be used to increase the speed. However, PDES represents the best approach that provides the most increase simulatneously in speed and scale needed to study large networks that, for example, arise in the DOE's science enterprise (see Section 3.2).

To incorporate very complex software-defined network behaviors, a fully virtualized PDES-based network simulation approach is needed [125].

Challenges

There are several challenges in PDES for network simulation, some of which are highlighted in the following.

- **Partitioning** How should the network be partitioned into subnetworks/LPs? It is usually advantageous to partition the network so there are more numerous LPs than processors to allow some flexibility in mapping LPs to processors for load balancing purposes (described next). On the other hand, too many LPs reduce the execution efficiency of the simulation because of the overhead in managing many LPs and because event communication between LPs is generally less efficient than intra-LP event exchanges within a single LP. Another very important question concerns which communication links are split across processing elements when partitioning the network. For synchronization reasons, it is much more advantageous to let communication links that have a high latency span processing elements. In other words, it is better to partition low-bandwidth links that span long distances than high bandwidth links connecting nodes that are close together.
- **Load balancing** A related question concerns the distribution of logical processes among processors. The load distribution algorithm attempts to simultaneously ensure each processor has approximately the same amount of computation to perform, while also trying to minimize the communication that takes place between processors. These two goals are sometimes conflicting. Static load distribution strategies map the logical processes to computing elements prior to the execution of the simulation, and do not attempt to redistribute workload during the course of the execution. Dynamic load distribution strategies do redistribute the mapping of processes to processors after the execution begins in order to try to maintain an effective load balance.
- **Synchronization** A synchronization algorithm is needed that performs well on wireless links and ensures that the parallel execution of the simulator yields the exact same results as a sequential execution. In particular, synchronization algorithms are needed to ensure that events are processed in the correct global timestamp order and ensures that repeated executions of a simulation with the same inputs produce exactly the same results.
- **Virtualization** To reduce the modeling effort in producing high-fidelity behaviors of network entities, the integration of virtual machines needs to be achieved with virtual time-aware synchronization built into the frameworks [125]. Enabling discrete event execution in para-virtualization-based network simulation systems such as Mininet [48] is a major challenge towards enabling novel applications of network simulation. Also critical is the advancement of transparent and easily portable support for virtual time-synchronized, unmodified software behaviors in networked environments executed using discrete event simulation. Such a support is needed for a wide range of application scenarios spanning Internet of Things (IoT), cyber-physical systems (CPS), and smart distributed devices.

3.6 Simulations for Hardware Co-Design

Simulations for hardware co-design involves the accurate exercise and exploration of software and hardware configurations in tight conjunction for targeted advances in high-end computing. Co-design research uses a wide variety of simulators spanning the space from individual component design to models of full HPC systems. These simulators share similar approaches, trade-offs and challenges as described for network simulations in the previous section, and the simulators reflect this design space. For example, one of the primary simulators used for

the design of advanced processor features is `gem5` [6, 109], which offers multiple modes of operation. However, it is commonly limited to serial execution because of the challenges of running in parallel that incurs modeling and accuracy limitations when synchronizing cores to improve the performance of the simulator. CODES [25, 62] is a simulator focused on modeling the system-level network, which supports parallel execution with both optimistic and conservative synchronization across parallel threads of operation. The Structural Simulation Toolkit (SST) [91, 106] supports simulations at both the node- and system-level and uses a conservative lookahead interval for synchronization. One of the greatest challenges of using PDES for hardware co-design is developing workflows that can tie these various levels of simulations together to inform the design of all the components across the system stack.

Mixed Fidelity Models

The co-design of large-scale HPC systems requires understanding how the individual components of a system will work in conjunction with several thousands of other components. Because it is impossible at present to simulate all components at a high level of detail, it is necessary to develop techniques for simulating systems at multiple levels of fidelity. For example, one could simulate a handful of nodes in the system at full fidelity (including cycle-accurate CPU and memory system models), while simulating the rest of the system at a higher level of abstraction. This allows the designer to analyze the node-level architecture performance as a part of the larger system.

The greatest challenge for these types of simulations lies in the development of multiple compatible models operating at the different levels of fidelity. The challenge lies in ensuring that the different fidelity levels properly represent the application being modeled, as well as the network and system level protocols running on the platform. For example, consider the problem of simulating large scale HPC system running a complex MPI application. In the high fidelity models, the MPI stack would be running on the simulated processor. In the low fidelity models, the MPI stack would be emulated using simplified abstractions. For the system to be successfully simulated at a reliable level of detail, the choice of algorithms used for the various MPI operations would have to be closely comparable between the two models.

Using Hardware to Simulate Hardware

With the increased availability of highly parallel computing architectures (FPGAs, CGRAs, etc.), it is desirable to exploit these hardware units as new simulator co-processors in co-design simulation frameworks. These types of systems could be used to greatly increase the simulation performance of cycle-approximate/accurate models for CPUs, GPUs and custom computational components. This methodology has been explored in the past [19, 20] and was shown to improve instruction simulation rates for microprocessors by over an order of magnitude. Further research is needed to understand how to generalize this approach to a wider range of processor and accelerator architectures. This technique could be applied in conjunction with mixed fidelity models to further increase the flexibility of the modeling capability.

There are challenges inherent with integrating any type of co-processor into the discrete event simulation. The first is synchronizing the execution of the co-processor with the discrete event core. Another is finding generic ways to translate the data between the software realm and hardware emulation realm.

Event Scheduling Optimizations

A typical characteristic of cycle-accurate hardware simulation is the use of fine-grained event-driven models for the constituent hardware components such as cores and caches. For example, each hardware component may have an associated clock handler that executes every clock cycle. Furthermore, depending on the workload, the clock handlers may not have much computational work to perform in the execution, as in simply checking to see if any requests have arrived over a bus and returning without the need to do any additional work. If the amount of computational work per clock handler is very small, then a significant fraction of the simulation time may be spent scheduling and running such fine-grained tasks. In these cases, there is an opportunity to improve simulation performance by reducing task scheduling overheads using advanced event scheduling techniques such as event aggregation. For example, SST consolidates all clock handlers into a single event that is scheduled in the event queue. Additionally, the model is able to remove its clock handlers from the clock list when there is no work to do. It would be beneficial to investigate more advanced techniques for enhancing the PDES runtimes to help automatically determine when and how events can be aggregated to reduce these scheduling overheads (also see Section 4.1).

Hardware Simulation with Optimistic PDES

In some cases, the performance of conservative PDES may be constrained by low latency links that span multiple processing elements during the parallel domain decomposition. Because each component LP can only execute events if it can prove that no remote LPs will produce events before its timestamp, the minimum latency of the spanning links determines the frequency of synchronization and event exchange between components assigned to different partitions. The lower the minimum link latency among all such links, the higher the synchronization overheads, reducing the scalability of the simulation. It would be worth investigating the degree to which optimistic methods of PDES can help improve the performance of these simulations by reducing or eliminating the need for frequent synchronization, and how the speculation and rollback overheads can be mitigated through effective load balancing (to reduce clock skew across simulator cores) and how new blocked-waiting heuristics (such as based on event prediction) can be enabled through PDES programming model and runtime advancements (see Section 4.1).

Simulation of Emerging Alternative Technologies

There is a range of new hardware technologies whose simulation models are naturally discrete event in nature, which include neuromorphic computers [100] and quantum computing/networking devices [26, 122]. Computational needs are highly intensive in these simulations, requiring parallel execution. Previous attempts at using PDES for these technologies have shown basic feasibility but also exposed immense challenges with respect to model development and parallel scalability. Major advances are needed in PDES technologies to enable the effective designs of neuromorphic and quantum hardware solutions. Event granularity optimization, model entity aggregation, entity-to-processor mapping, load balancing, and event concurrency enhancement are some of the major dimensions along which computer science needs arise in PDES-based design of these emerging hardware technologies.

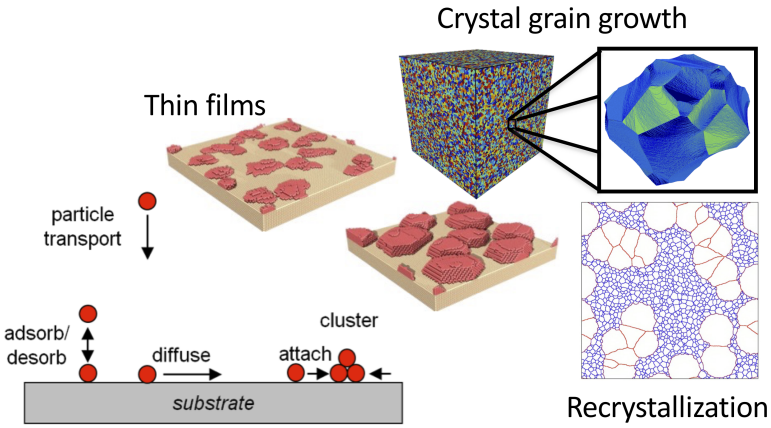


Figure 3.8: Illustration of salient phenomena in material science for application of discrete event-based kinetic Monte Carlo methods

3.7 Material Science Applications using Kinetic Monte Carlo

For more than half a century Kinetic Monte Carlo (KMC) simulations have been used successfully to model many processes including crystal grain growth, thin film growth, dopant migration in semiconductors, and material microstructure evolution due to radiation damage, accounting for the requisite time scales while retaining atomistic resolution. The application range of KMC is exceptionally wide in the scientific computing domain and includes diffusion-controlled chemical and biochemical reactions, which even intersects with studies in epidemiology and population dynamics.

In KMC, only events significantly changing the system state are tracked, such as a vacancy diffusing from one lattice site to a neighboring one in a solid. The simulation effectively integrates over unimportant faster (stiff) time-scales, like atomic vibrations around their crystal lattice sites. Therefore, a KMC simulation belongs to the class of discrete event models and simulations in general.

On a single processor, KMC may be relatively easy to implement. However, getting it to execute correctly and with notable speed-up from parallel computers is *considerably* more difficult. Often, the instants and locations of events are selected stochastically, which renders any parallel KMC simulation asynchronous and irregular in nature. In addition, it is not certain that there is any upper bound on how fast information can travel, making it difficult or nearly impossible to compute a meaningful lookahead for synchronization. To further complicate things, interactions among objects in the simulation can have long-range effects, leading to significant inter-object communication.

Given the tight synchronization constraints on KMC, optimistic simulation appears to be the most practical and general approach ahead. A Time Warp-based KMC simulator, SPOCK, demonstrated [71] that KMC can be executed accurately in parallel, without needing approximations as compared to a sequential simulation. Further, the same implementation showed that the execution can scale linearly to 400,000 processor cores, indicating that parallel KMC can be attempted even at the scale of the largest supercomputers.

The design and development of a general, parallel, and scalable KMC simulator would be groundbreaking in many fields since there is a significant number of KMC applications across the sciences. There is a clear need for such a simulator that allows researchers to formulate a model as a sequential simulation, which is automatically enabled by PDES to employ the model on large supercomputers. However, much research and development remain to accomplish this goal. Necessary components include automatic support for roll-back and reverse computing of events and dynamic load-balancing. The feasibility of using automated incremental

state-saving for roll-back support for the SPOCK simulator has been demonstrated lately [96, 97], but a large amount of work remains to make the approach more practical and effective.

3.8 Epidemiological Planning, Response, Policy, and Decision-making

The significance of gaining better insights into the dynamics of large-scale epidemics is well known. The enormity of epidemic outbreak effects has become common knowledge with world-wide attention directed towards controlling them. In addition to the non-technical factors that come into play in the process of effectively dealing with epidemics, an important technical aspect continues to be elusive and remains to be explored, namely, gaining a good understanding of epidemic dynamics and the ways and means by which various contributing factors affect the propagation phenomena. Public health planners and policy makers use epidemiological simulations to study a variety of factors that influence epidemic dynamics within a population. The set of all factors called non-pharmaceutical intervention (NPI) mechanisms plays an extremely important role in the planning, prevention, and response phases for many epidemics. These NPI considerations represent extremely high levels of financial effects, in addition to many other non-technological repercussions such as political and societal impacts.

Simulation continues to be an important tool to augment analytical models that are based on simplifications. In contrast to numerical integration-based analysis of analytical (differential equation-based) epidemic models, simulation often provides tremendous flexibility in incorporating many factors. Large spatial scales and high resolution of behavioral detail propel the challenge of sustaining simulations of epidemic propagation dynamics at increasing scales from cities and states to countries. Certain epidemics with global spans serve to motivate simulations at even world-scale.

Decisions and policies are typically based on statistical inferences from results of multiple simulation runs that attempt to explore the model's associated parameter space as exhaustively as possible. This requires very fast turnaround times for each run so that enough statistics can be gathered within a reasonable duration of wall clock time on which actionable decisions can be based. In light of this, the need for parallel execution of such epidemiological models becomes evident. Large-scale computational epidemiology is an important area of active research in an era when larger and more powerful parallel platforms are becoming increasingly common. Scalable algorithms are therefore imperative for large scale realistic epidemiological simulations that can exploit the computing resources offered by today's state-of-the-art parallel platforms.

PDES is an excellent match to simulating NPI at large scales without sacrificing fidelity of representation in geography, population sizes, behavioral complexity, and so on. Ideally, epidemiological models should be sufficiently detailed to capture realistic models of the underlying phenomena and produce actionable insights. Realistic models tend to be very complex and the associated parameter space very large. In turn, this implies that the resulting computational problem becomes very large and unsuitable for sequential execution.

The basic processes underlying the non-pharmaceutical parts of epidemiological phenomena are discrete interactions and transitions that can be mapped to events at which most pertinent state evolution happens. Therefore, they are naturally amenable to discrete event modeling and simulation. Mobility and interactions in proximity can be mapped to events of interest at which key modeling elements are invoked and incorporated, whose composition and aggregation unravels all epidemiological variables of interest. Similarly, interventions such as time-spanned vaccination campaigns, curfews, bio-bubbles, and so on are all readily treated as discrete state transitions introduced into the base simulation of the epidemiological propagation. Feature-specific models of health conditions and disease states are easily and efficiently modeled using discrete-event transitions

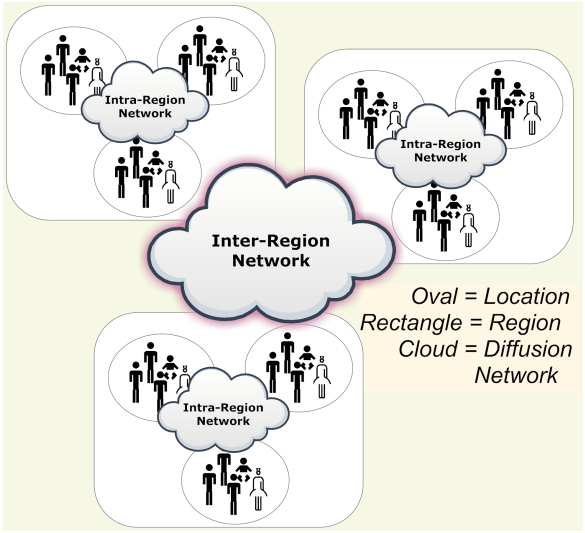


Figure 3.9: System abstraction for discrete event-based modeling of epidemic propagation [82, 84, 85]

of probabilistically timed state transition machines at the level of each individual. These are discretized generalizations that relax the classical SIR (susceptible-infected-recovered) or SEIR (susceptible-exposed-infected-recovered) or many other similar classes of models into a single high-fidelity framework of models that can be customized for various types of epidemics. While classical analytic models such as SIR are commonly used with simplifying assumptions, the PDES-based approaches can accommodate a rich variety of complexities in specifying and capturing the realities of modern epidemics [82, 85]. A wide range of spatial scales can be supported using a hierarchy of the same basic discrete event framework of individuals diffusing and reacting across intra-region and inter-region networks (see Figure 3.9).

Challenges

Although discrete event models have been developed as proofs-of-concept, their parallel execution to sustain large scales remains a significant challenge. The challenge is further amplified by an inadequate understanding about the effective ways to realize these complete, discrete event models on the latest hardware platforms including GPU and FPGA accelerators. Research is needed to enable new PDES modeling and execution approaches to enable assimilation of dynamic data (such as rapidly arriving ground truth data streams) into discrete event models of epidemic propagation and intervention phenomena for simulation-based predictions, verification, and what-if decision-making, especially in the presence of rapidly changing environments in real-time. Towards this end, new PDES-internal technologies are needed to increase the speed of model execution using many modern parallel systems. Similarly, new PDES-external technologies are needed to optimize the multi-scenario evaluation of PDES models for decision-making in the high-dimensional space of possible scenarios that only differ from each other in minor increments [82]. Usability of PDES environment is a key challenge in enabling the epidemiological domain experts to develop their own customized instances of the next generation of PDES-based epidemiological simulation models.

Chapter 4 Challenges and Opportunities in Core Discrete Event Technologies

4.1 Virtual Time-Aware Load Balancing and Event Scheduling

One of the key requirements for PDES engines to achieve speedup is determining a good partitioning of the state of the system and scheduling of work across compute resources. This is an issue that appears in many other parallel computing paradigms outside of PDES, and is relatively well studied in previous literature. Because finding optimal partitions and event execution schedules is generally intractable [114], practitioners have resorted to heuristic techniques that span from iterative graph partitioning [46] to dynamic work stealing [1] and many other approaches. A critical feature of the load balancing algorithm is managing the trade-off between data migration costs, compute load imbalance, and communication and synchronization overheads. Furthermore, there are also related considerations, such as how to select a good computational granularity for each partition, task, and LP that exposes sufficient concurrency and flexibility to balance the loads adequately while still amortizing the costs of event scheduling.

Load balance is critical to the performance of both conservative and optimistic PDES techniques. In conservative approaches, under-loaded compute cores will advance their local virtual clocks faster than their peers and then must wait for their peers to catch up before they can continue doing useful work. In optimistic approaches, the under-loaded cores may execute too far into the future and then roll back events when straggler events arrive, sometimes leading to cascading rollbacks. Having a well load balanced system ensures that local virtual clocks of the LPs stay relatively aligned with one another, thus reducing time wasted in waiting, misspeculation, and rollback overheads.

Load balancing specifically for PDES is not as well studied as for general-purpose, bulk-synchronous and task-based execution paradigms; nevertheless, many techniques and lessons can carry over from those domains. At a high level, load balancing approaches can be categorized into two types: static and dynamic. Static approaches make fixed assignments of partitions/tasks/LPs to compute cores relying only on knowledge about the system prior to simulation execution. These are often based on geometric or graph partitioning methods [46, 78, 93], which may be simple to implement and yield acceptable results for systems where task/event loads and communication patterns are predictable and stable. However, this is not always the case: systems that tend to be the most relevant to domain scientists are often those that have unpredictable and irregular behavior, which cannot be ascertained before execution starts. Dynamic approaches allow partitions, tasks, or LPs to migrate across compute cores during the simulation, resulting in load balance that is more robust to dynamic changes in the distribution of event loads across the simulated system [15, 45, 53, 131]. The trade-off is that dynamic load balancers are typically much more complicated to implement and require additional capabilities from the application and runtime, such as support for runtime introspection, dynamic state migration, and directory updates and indirection. Furthermore, semi-static schedulers constitute a class of load balancers that offer some of the benefits of fully dynamic load balancing with less complexity than fully dynamic (asynchronous) load balancers [9].

A research program is needed to investigate and address many PDES-specific considerations and challenges to enable scalable PDES for dynamic applications. The design space for load balancing and event scheduling algorithms is large and complex. While previous work has applied various load balancing techniques to PDES [7,

8, 29, 54, 60, 105, 110, 121], the community needs to do more to investigate techniques that take into account how PDES-specific application load and communication characteristics interact with emerging hardware trends.

Synchronization-aware Partitioning

An example of a PDES-specific consideration is the latency of events that traverse links that are cut at an inter-processor partition boundary. In many window-based conservative approaches, the minimum among all latencies of links cut at processor boundaries determines the lookahead interval. The shorter the cut link latency, the lesser the event concurrency exposed between processor cores for synchronizations, and the more frequent the required synchronization. Similarly, in optimistic PDES, lower latencies of cut link increase the probability that an event transmitted across a partition boundary causes a rollback, as the receiver may speculatively execute events while the straggler events are in flight. As data movement and synchronization costs are becoming increasingly dominant compared to compute costs on modern architectures, load balancers need to place a greater emphasis on the costs of LP data migration, the subsequent communication costs, and impacts on synchronization overhead, rather than focusing primarily on balancing local load metrics (e.g., event count). Another example of a PDES-specific concern is managing the overheads of scheduling fine-grained events. In a fixed-timestep computing paradigm, one might simply change the degree to which components are aggregated to amortize overheads; however, in PDES each member component processes events at its own individual event timestamps, complicating the selection of an optimal spatial aggregation strategy.

Application-Runtime Linkage

Another important question to be answered is how load balancing and event scheduling capabilities can be productively enabled by the supporting software infrastructure. A highly productive PDES framework should provide support in terms of the language, library, and runtime so that the simulator (with some assistance from the application developer) can automatically determine when load balancing should occur, how LPs can be migrated between processing elements, and how events to those LPs should be scheduled to the underlying compute resources. Complicating the problem, the introduction of new hardware accelerators presents heterogeneous resources to which the load balancer needs to assign work. The PDES runtime needs support for automated introspection about the LP event execution and rollback and communication costs, as well as algorithms and heterogeneous performance models to determine how to use that information to formulate a good load balancing strategy. Finally, since application developers have domain-specific knowledge about the characteristics of their system, it would be beneficial to allow developers to expose a kind of relevant information that the runtime would not be able to determine by itself.

Elastic Execution

Strongly coupled to the load balancing problem is the challenge of making PDES execution dynamically scalable. With elastic computing platforms providing the mechanisms to add and remove hardware resources on-the-fly during execution, PDES runtimes should be able to adopt their mapping and scheduling to the resource changes. This dynamism may be even more critical for real-time execution in which the PDES runtime must adopt not only to dynamically changing computational demands from the event processing but also to fluctuations in the externally induced real-time constraints from the application.

4.2 Compiler Support

Optimistic PDES has the fundamental requirement of reverting on demand to previously (speculatively) executed states of an event function. This aspect has attracted a significant amount of compiler work regarding reversible computation, broadly addressed by two approaches: (1) generating reverse code which allows the computation backwards from a given program state, and (2) different variants of checkpointing (also called incremental state-saving), including copy checkpointing, differential checkpointing, and incremental check-pointing (Chapter 9 "Adding Reversibility to Irreversible Programs," in Introduction to Reversible Computing [80]).

The advantage of having reverse code for reversible programs is that no run-time overhead is incurred in the forward direction. For irreversible programs a run-time overhead occurs, by storing the destroyed information, making them reversible. Checkpointing approaches usually incur a higher runtime overhead than reverse code approaches in the forward direction, but can be faster when restoring a previous state depending on the checkpointing frequency and scope of a checkpoint. Reversibility can also be addressed at the language level, such as in Janus [126, 127] a reversible programming language. For example, in a reversible program language a program for lossless compression needs to be implemented only for compression. The decompression of data can be achieved by simply calling the inverse (or undo) operation.

Compiler transformations have been focusing on automating these techniques at the source and binary level. The scope of the language supported constructs is another dimension, where various forms of parallelism have attracted most attention recently.

Reverse code generation has previously been discussed [80] for the `C` language and with exact measures of how much information is destroyed by certain operations and how code can be generated to store and recover this information in the backward code. Techniques for incremental check-pointing are also discussed in the same context, and automated compiler-based techniques for reverse code generation have previously been investigated [12, 49, 96, 116].

An illustration of this approach is Low Overhead Runtime Assisted Instruction Negation (LORAIN) [49] that accounts for, and in many cases reverses, the computation without resorting to state-saving techniques. Based on the Rensselaer's Optimistic Simulation System (ROSS), it is coupled with the LLVM compiler to generate the reverse code. The reverse code generation is limited as it cannot handle more sophisticated `C++` language features such as virtual functions and exceptions, but since it operates on the LLVM IR, it is independent of the source level language. Loops require user-provided information. An alternative fine-grain time-sharing Time Warp system [77] runs on multi-core Linux machines and makes systematic use of event preemption in order to dynamically reassign the CPU to higher priority events/tasks.

Another approach [96] automates incremental state-saving for `C/C++` at the source-level by instrumenting all write operations such that any irreversible `C/C++` program becomes reversible by storing a trace of all memory modifications as a sequence of address-value pairs which are used when restoring a previous state. The approach is implemented in the tool called Backstroke and its use has been demonstrated for Kinetic Monte-Carlo simulations.

There also exist methods for autonomic state management for optimistic simulation platforms such as an autonomic system that can utilize both an incremental and a full checkpointing mode [76]. At run time both code variants are available and the system switches between the two variants, trying to intelligently select the more efficient checkpointing version. An instrumentation technique has been previously applied [23] to relocatable object files, operating on the Executable and Linkable Format (ELF) using the tool Hijacker [75] to instrument

the binary code to generate a cache of disassembly information. This avoids disassembly of instructions at run time. At run time, the reverse instructions are built on-the-fly also using pre-compiled tables of instructions. This approach has been further refined to also operate on shared libraries [24] and bears similarity to other methods [96] paying an instrumentation overhead. The information that it extracts from instructions is similar to address-value pairs, pairing the target address with the size of a memory write. Some progress has been made also in utilizing hardware transactional memory for further optimizing single node performance [95].

A focused research program is warranted to build on these preliminary, scattered attempts into a coordinated and cohesive whole to bridge the critical gap of a streamlined, PDES-oriented compiler technology.

4.3 Lookahead Extraction

An important challenge in lookahead-based advancements relates to application-agnostic and application-specific approaches for automated lookahead extraction [67, 104]. Application-agnostic methods include compiler-based analyses of application event handler codes and random number stream presampling. Application-specific methods include domain-specific analyzers, application programming interfaces, and domain-specific languages to enable exposure of information needed for extraction at configuration time or runtime. All lookahead extraction techniques work in such a way that there is no burden on the modeler. The most ideal techniques ensure that there is no additional information that needs to be supplied on top of a sequential model.

Lookahead-extraction methods have been previously studied in a limited context at small scales of parallelism and applications such as mobile ad-hoc network simulations and multi-processor simulation [22, 28]. However, modern parallel processing and supercomputing architectures need new research to apply those techniques or invent new techniques.

Research is needed to investigate how to efficiently achieve lookahead extraction from pre-sampling random number generators particularly on accelerated platforms. Because of the hardware-specific nature of libraries on accelerators, the implementations need to be revisited to (1) add new interface methods to provide pre-sampling functionality, (2) provide new, efficient implementations for pre-sampling, and (3) provide balance between memory, pre-computation time, and parallel execution facilities.

New compiler-based methods are needed to perform code path analyses to track the virtual time delays backwards from each event scheduling operation within the application event handlers. Generalize methods as well as domain/application-specific methods need to be investigated to intelligently extract the maximum amount of lookahead from the applications' event codes.

Through a concerted research program, advancements made in lookahead extraction can have far-reaching impacts and fundamentally change PDES at its core, to consequently benefit a large number of applications and their adoption.

4.4 Benchmarks

Benchmarks for evaluating the performance and correctness of simulators form an important ingredient in the development of PDES technology. They allow comparing and ordering different simulation approaches and implementations and they also aid in identifying flaws in implementations.

Correctness Evaluation of Complex Dynamics

PDES benchmark models have been developed [99] for evaluating simulators that goes beyond the classic PHOLD benchmark model. This is enabled by including extra state variables and matrix computations that aid in detecting simulator errors. In such models, each event involves non-commutative matrix algebra, and the matrix that results from the simulation of the model serves as a checksum or hash of the simulation and is sensitive to the order of events. The size of this matrix can be controlled by the user, as can the number of bits in its elements. This new benchmark is particularly useful for debugging simulators that are based on the Time Warp Algorithm as its mathematical properties allow for checking of various assertions. In this model, the amount of arithmetic operations can be tuned relative to the amount of memory modifying operations.

The state of each LP contains two square matrices of integers: an accumulation matrix A, and a transformation matrix T, each of size $n \times n$, where n is an integer constant chosen by the user. Each event message contains the transformation matrix of the sender. Upon execution of an event, the receiving LP multiplies its accumulation matrix to the right with the received transformation matrix. When an event is executed, the receiving LP schedules a new event for a randomly selected LP at an exponentially distributed virtual time delay.

At the end of the simulation, the matrices of all LP's are multiplied together, in LP ID (rank) order. The resulting matrix is the output of the simulation. Since matrix multiplication is in general non-commutative, the output depends on the individual events being executed in the correct order. The output serves as a check sum or hash of the simulation, and its size can be controlled by choosing the matrix size and the number of bits in the matrix elements.

A first evaluation of this benchmark [97] used a reversible variant of matrix multiplication and evaluated multiple system variants: the Backstroke tool [96, 98] for generating reversible C++ code, the (adapted) Janus compiler [128] for generating forward/reverse functions in C, and hand-written reverse code. These constitute a first step that needs to be methodically expanded and refined in a PDES research initiative for the success of PDES-enabled scientific advancements. Many additional benchmarks need to be developed similar to the preceding examples for the automatically evaluating the correctness of complex dynamics in PDES execution.

Characterizing PDES Model Properties

For any given simulation application or model implementation, there are a number of metrics that can have a large influence on the execution speed, relatively independent of the simulator engine used. Classic parallel computing measures such as ratios of computation to communication, computation to I/O, computation per memory access, etc. remain directly relevant. Nevertheless, to gain the additional insight necessary for PDES execution, the conventional measures often have to be recast in terms relevant for PDES execution: computation per event, ratio of local to remote events, number of events (or computation) per lookahead period (for conservative simulation) or between remote events (for optimistic simulation), size of event data per event, etc. In addition, there are PDES-specific metrics characterizing the simulation engine itself that bear importance, such as synchronization overhead, run time cost for sending remote events as a function of event size, overhead for time synchronization, and so on.

Typically, simulation engines are benchmarked for performance using standard benchmark models, such as the classical PHOLD bechmark or the later LA-PDES benchmark. There is also considerable amount of literature on performance of various algorithms to manage the event priority queues. However, other than a few rare efforts [27], there is a dearth of application model characterization across different application domains. As a result, the field lacks a sound basis to weigh trade-offs between computation, communication,

global synchronization, I/O, partitioning, load-balancing, etc. This characterization can be useful in answering questions as follows. How important is model-space event retraction/cancellation? What fraction of events are rolled back for a given partitioning? Even for questions where the options are well characterized, such the choice of priority queue algorithm, practitioners have only anecdotal evidence or experience to decide which algorithm might best suit a given model. There is a general lack of a formal basis for choosing one synchronization or partitioning algorithm over another, beyond domain experience, familiarity, or phenomenological approaches.

A comprehensive research program to define and characterize model properties across many application domains would have a broader impact on the field. The results of such a program would inform practitioners' judgement on a host of implementation choices when addressing a new application, by enabling direct comparison to existing model properties, and identifying truly novel behavior or regimes requiring new simulation algorithms. At the same time such a program would enable meta-insights into equivalence classes of PDES applications, in terms of performance properties, and generic performance predictors based on application properties. These insights would move PDES application performance from phenomenology to the level of an engineering discipline. All the "Inner Technologies" topics in PRO Category 1 would benefit from this program.

Assessing Communication, Computation, and Memory Performance

The LA-PDES benchmark [74] is a parameterized benchmark application for measuring parallel and serial discrete event simulation (PDES) performance. Applying a holistic view of PDES system performance, LA-PDES tests the performance factors of (i) the PDES engine in terms of event queue efficiency, synchronization mechanism, and load-balancing schemes; (ii) available hardware in terms of handling computationally intensive loads, memory size, cache hierarchy, and clock speed; and (iii) interaction with communication middleware (often MPI) through message buffering. LA-PDES defines seven scenarios for individual performance factors and an comprehensive stress evaluation scenario. Each scenario is simply a set of input parameter values such as number of entities and events, end time, inter-send time distributions, computational and event load distributions, memory use distributions, cache-friendliness, and event queue sizes.

At its core, each simulation entity in LA-PDES communicates sends messages at randomly chosen time intervals to a set of randomly chosen destination entities; it receives messages from sending entities, which cause it to execute a computational load before proceeding to sending out a next batch of messages. Input parameters control all the distributions of receiving entities, times, work load, and locality.

LA-PDES has been implemented on several PDES engines with a particular emphasis [37] on comparing Just-in-time PDES engines, such as Simian [94], to traditional engines, such as ROSS [11], and comparing work-stealing [120] or load-balancing schemes [30].

Without a PDES research program, such initial efforts leave a wide gap in our scientific understanding of the basic interactions of PDES workloads on the high-performance computing systems of DOE and other agencies.

4.5 Accelerated Computing Systems

Given the dominance of accelerators in the high performance computing domain, new methods are needed to fully exploit the accelerators in PDES systems. Currently, the known technology is severely limited in taking traditional PDES models and porting them to the unique hardware capabilities of accelerators, such as the Single Instruction Multiple Data (SIMD) mode of GPUs and the Bulk Synchronous Processing mode of Intelligent

Processing Units. Fundamentally new algorithms are needed to realize efficient event storage, representation, scheduling, and execution of events on the GPUs for main event processing. New synchronization methods are needed to efficiently compute global virtual time in these architectures. It is possible that novel definitions, data structures, and implementations can be invented specifically for these architectures. Also, the integration of event processing needs to be carefully resolved with the new AI/ML algorithms that are becoming part of the application codes. Without these advancements, it becomes nearly impossible to use the current leadership-class accelerated computing systems of DOE user facilities in all the emerging applications in the DOE and scientific mission spaces.

Chapter 5 Challenges and Opportunities in Discrete Event Eco-System and User Needs

In delivering the scientific and technological outcomes across applications, an important need that arises is in regard to how the end-user can effectively use the PDES applications to solve the important problems of their domain. This involves interfaces, modeling languages, usability features, ready-to-use solvers, interoperability standards and implementations, ensembles, steering and visualization.

5.1 Domain-Specific Languages and Modeling Environments

Although advancement in PDES is necessary in many applications, the bar is high for application developers to move their DES models to PDES for effective parallel execution. Therefore, the barriers to moving from DES to PDES need to be significantly lowered for success in PDES applications, especially for rapidly porting to DOE's leadership class computing at a low development cost. Similary, it would be highly desirable for PDES application developers to easily move their PDES codes from laptops to supercomputers. Petri-Nets, Agent-based Simulations (ABMS), and Cellular Automata (CA) models are instances where domain-specific languages and modeling environments are effective in briding the gap. However, scaling techniques are necessary in PDES-based execution for achieving provably scalable parallel models in areas such as ABMS and CA.

To illustrate, Cellular Automata (CA) form the basis for models in domains as diverse as forest fires [39], materials [88], transportation, chemistry, and countless other domains. It is less well known that CA constitute a special case of discrete event simulation and that significant advantages in computational efficiency and, in some cases, numerical accuracy can be obtained by transforming a cellular automaton into its event driven form [69, 115, 118, 130]. Typically, the advantages of event driven CA derive from two aspects.

The first is natural activity tracking that emerges from an event oriented simulation method [64, 65, 117, 129]. Surprisingly, the possibilities of activity tracking are only now being rediscovered, separately, in the domains sciences that often favor CA. Work on activity tracking in event driven CA anticipates, indeed encompasses, recent work on front tracking for grain growth models [108] by offering efficient simulation algorithms and conceptually simple modeling constructs to focus computational effort on dynamic portions of the cell space. The second aspect is the use of continuous time in the simulation model. This improves efficiency by avoiding delay steps to simulate velocities in the physical space being modeled, and it improves numerical accuracy by not truncating random variables used to model motion through time. This continuous in time simulation is achieved by progressing from event to event, rather than step to step, and allowing real valued intervals between events.

The mathematical concepts and techniques that make cellular automata into a special case of discrete event systems have been well know for decades [129]. Significant, practical improvements in the handling of collisions in discrete event models simplified this original approach [21], and these improvements have become commonplace in event driven simulations in general and cellular spaces in particular [69, 130]. The usefulness of these basic techniques have been validated by a growing array of modeling and simulation techniques aimed specifically at event driven simulation of cellular spaces [118, 119].

Where event driven CA can offer improvements in numerical accuracy and computational efficiency, these improvements can only be enhanced by using PDES algorithms to further reduce execution times or to improve

the resolution of a model by simulated a larger cell space. Initial forays into this research area [55] have been promising, but the topic has yet to gain significant traction in the broader PDES community. Specifically, the peculiar role of simultaneous events in many cellular models imposes a need for specific modeling constructs that are less useful, and hence less familiar, in some other application domains [69, 70]. An expanded interest in the application of PDES models to problems in scientific computing would spur rapid and profound advancements at the intersection of PDES algorithms, modeling methods, and application domain that are unlikely to emerge with the present, less integrated approach to research and development.

5.2 Discrete Event-based Mathematical Solvers

Discrete event-based mathematical solvers arise as the underlying execution model behind integrators such as quantized-state integrators, variational integrators, and adaptive time-step integrators, and applications of the transmission line matrix methods. Their application areas span a wide range such as radio signal propagation, multi-scale physics (fusion, plasma, magnetosphere), electric grid transient dynamics, and cyber-physical system dynamics, to name just a few.

PDES can also help enable more efficient execution of simulations where tasks are irregular (e.g., temporally and spatially non-uniform), and their dependencies are only fully known when the *global* state has reached a certain iteration or simulated time. Problems that exhibit these properties are termed *unstable-source* problems [38]. For these unstable-source problems, both *static* and most *dynamic* task graph frameworks are insufficient since the dependencies of a task are not necessarily fully known at task creation time and can be altered by subsequent task executions. Additional checks must be made to determine whether a set of tasks in the dependence graph are safe to execute (e.g., based on lookahead). For these reasons, many applications resort to using a mechanism of global fixed time-step per iteration to enforce task dependencies. However, such an approach can leave a lot of performance untapped due to global synchronization overheads and because it prohibits the use of non-uniform time-step updates. PDES provides a way to re-capture some of that performance by enabling asynchronous, spatially irregular update schemes (e.g., adaptive local time-stepping) while at the same time enforcing all dynamic task dependencies that arise.

Fluid simulation codes with large discrepancies in wave speeds are a prime target for these techniques since state dynamics tend to be more localized, and non-uniform time-stepping techniques can provide a large potential work-reduction opportunity. Examples of these types of applications of interest to DOE include: storm surge simulation, plasma physics, and climate simulation. Recent work has shown significant speedup potential from adaptive, locally time-stepped simulations using PDES [8], where an improvement of $4\times$ per-node compute throughput was demonstrated compared to a synchronous, fixed time-stepped formulation of a shallow water equation simulation. Furthermore, a $15\times$ work reduction was predicted for full-scale storm surge simulations using the adaptive time stepping PDES techniques.

Still, there are many challenges and opportunities in this space, including implicit systems that utilize iterative solver methods with non-uniform convergence behavior. For problems with locally non-smooth solutions, they tend to exhibit very poor convergence around these regions. Synchronous methods handle these cases by simply iterating globally until convergence of all regions of the mesh. However, similar to the adaptively time-stepped simulations described above, there is a large work-reduction opportunity in these iterative methods that can be enabled by PDES techniques by avoiding computationally unnecessary updates in the locally converged regions. We could then explore dynamic load balancing to achieve a faster overall time to solution in these cases. Implicitly time-stepped systems may be able to combine the local time-stepping and non-

uniform iterative convergence optimizations together within a PDES framework to maximize work-reduction opportunities.

Furthermore, an interesting possibility that arises when applying optimistic PDES approaches to iterative convergence problems is the possibility of avoiding rollback operations. So long as the end solution still satisfies the convergence criteria, speculative updates that temporarily change the evolution of the non-converged state may not be harmful to the quality of the resulting solution. The performance and accuracy impact of optimizations such as these would be an interesting topic of future investigation. In addition to these technical research questions, practical challenges in this area include consolidating and characterizing local time-stepping and iteration algorithms, packaging high-performance implementations of these algorithms together and making them available through existing DOE computing software infrastructure, and demonstrating their impact on large real-world applications.

The aforementioned methods also bear relevance to ways in which PDES could be used to help solve implicit systems: (1) work reduction for systems with irregular spatial convergence properties, and (2) eliding rollback when considered not harmful to quality of converged solution.

5.3 Interoperability, Federated Simulations and Ensembles

PDES encompasses an important mode of simulations, namely, federated simulations in which multiple disparate simulators cooperate to achieve a combined, larger function. Interoperability at various levels is key in this approach, which includes continuous-discrete model interfaces and multi-scale model interfaces.

The federated simulation services address two important components: (1) overall event processing order by each federate, and (2) synchronized event delivery to each federate. While enabling event processing order and synchronized event delivery, all in a single encompassing standard framework, the interface needs to accommodate a large variety of individual types of simulators. In general, there is a plethora of different simulator types — event-stepped vs. time-stepped, sequential vs. parallel, real-time vs. as-fast-as-possible, conservative vs. optimistic, etc. A federation might include any combination of any of these simulator types — this combination results in a PDES execution at its core. Moreover, the exact combination of the types is not always made known a priori to the federation, and hence the interface as well as the implementation must be sufficiently general to accommodate any/all of them. The interface must accommodate any arbitrary combinations of, and any number of instances of, different types of simulators, all in one core, seamless interface.

While federated simulation has been previously well studied in the context of live, virtual, and constructive simulators in the defense sector [33, 81], relatively little is known in the non-defense applications. To realize federated execution of large DOE applications on high-performance computing platforms, new federated approaches are necessary that are specialized and custom-built for runtime performance using the unique computational and networking infrastructure of the DOE science enterprise.

What-if scenario evaluations, real-time interfaces, and PDES-aware design of experiments are also key enablers of user-oriented functionalities needed for effective application of PDES advancements. Advances in this aspect of PDES ensembles can have tangible impacts on non-PDES applications as well, by presenting new ways of exploiting computational and memory overlaps in any ensemble of computational experiments.

5.4 Visualization

Visualization is key to successful PDES, playing an important role in understanding of the underlying behavior (e.g., communication, rollback) of simulations as they unfold and in understanding the behavior of the system being simulated.

Visual Analysis of Simulation Performance

Large-scale PDES can simulate millions of elements when predicting complex real-world infrastructure and phenomena. To deliver on the potential of these parallel simulations, users need to understand the execution behavior of the large simulation to ensure its efficiency. Optimistic, or speculative, executions ensure local coherence by using event rollbacks while conservative executions employ event blocking, both of which reduce simulation efficiency. The fact that rollbacks and blocking are impacted by several factors, such as the organization of processors and the simulated communication patterns, makes it difficult to identify efficient simulation configurations. The volume of multivariate time-series performance data also poses a challenge to understanding the execution dynamics of different configurations. However, interactive visual analysis of the simulator's components and behaviors at varying granularities can provide insights into bottlenecks and inefficiencies, thereby improving the capabilities of the simulator. Such visualizations present high-level summaries of performance that highlight areas of interest and allow for the low-level explorations of these areas to discover the sources of performance inefficiencies.

Previous efforts [13] showed that sampling and visualizing metrics related to rollback events can be used to identify problems with executing simulations in a multiuser environment better than numerical methods without visual analysis [31, 73]. Additionally, visual analysis of simulation performance using aggregates of performance metrics can further expose communication patterns between processors that cause more rollbacks, show the individual performance of processors relative to the global time, and help to identify non-performant simulation configurations [92].

To further improve the capabilities of analysis tools for PDES, more advanced data processing and exploration methods are required. Along this line, recent efforts have been made using advanced time-series methods for simulation performance analysis. Specifically, clustering and dimensionality reduction have been used to automatically detect and correlate changes in communication patterns and identify similarities in the dataset [51]. Coupled with interactive visualizations, users can explore complex temporal behaviors of simulations and gain insights into event execution dynamics. Such an understanding can guide the mapping of processes across the system to mitigate bottlenecks and improve performance.

There still exist barriers to be overcome when analyzing simulation performance. The lack of support for streaming data from simulations for in-situ data processing with lowered network bandwidth requirement is one such barrier. Additionally, allowing for more visual interactivity and highlighting significant features while maintaining low visual and cognitive loads remain challenging. Employing machine learning in the data analysis will also provide opportunities for greater insights to be gained with little effort from the user.

On-line Data Analysis, Reduction, and External Interfaces

During a PDES simulation it is often desirable to interface with external hardware and software, such as for saving restart files, on-the-fly visualization, and communicating with databases. In these applications, as well as internally within a PDES simulation, the relevant quantities are often sought as reduced data. An example is

the temperature in a materials simulation, which may depend on the state of many or all LPs, and which may be needed for saving output files, looking up material parameters in a database, or for visulalizing the current state. Although the implementation of data reduction inside a PDES model is possible, it is much more convenient to have an Application Programming Interface (API) for the same provided by the simulator. This is especially relevant for interfacing the PDES execution with the external world, which requires intimate knowledge of the simulator itself to be accomplished correctly and without any causality violations. Here too, the simulator can provide an appropriate API, alleviating the model developer from the predicament of creating fragile codes.

Research is required on how to best implement the suggested APIs in PDES in a way that is practical to the user while not adversely affecting parallel performance. It is expected that such simulator-provided APIs for on-the-fly data reduction and external world interaction will make PDES an attractive approach for more applications, and to make implementation of many existing models more practical and maintainable. The proposed APIs would provide the necessary mechanisms for data-streaming and on-the-fly analysis also relevant to on-line performance analysis and visualization.

Domain Visualization of Simulated Workloads

Extracting the most insightful output from the results of a simulation will positively impact user productivity and efficiency. Simulated systems are fundamentally endowed with the capability to produce metrics that are of a higher level of fidelity and granularity, as compared to real-world systems. However, this additional capability complicates performance analysis. To resolve this conflict, domain-based visualizations capturing the structural representation of the simulated system along with time-series data can efficiently expose important insights from complex, system-wide performance data. While visualizations integrated with time-series analysis are widely used to analyze and improve the performance of large-scale scientific applications [40], visual analysis solutions for simulated performance have been limited and lagging their scientific counterparts.

Domain-based visualizations can show the impact of domain-specific configurations, such as the placement of jobs on a simulated network [47], and the interactions between elements of the systems, such as nodes and links in the network [103]. Previously, certain co-design and design space exploration efforts using PDES have been supported by domain-visualization [13, 50, 72, 107]. Novel work has shown how well-selected visualization design and interaction methods can enable the exploration of a simulated HPC application running Dragonfly systems [5, 52, 56]. These visualizations capture the physical topology of the system for intuitive analysis, exposing temporal correlations as well as visual similarities and differences. Interaction methods such as creating connections between attributes and filtering allows users to apply expert knowledge in guiding their analysis.

Notwithstanding such recent forays, much remains to be researched and developed in a coherent research program. Due to the volume of data that can be generated by simulations, several challenges regarding data collection, management, and processing remain. For example, the use of in-situ analysis should increase as it would negate the need for storing large traces for post-processing and provide more information than offline, profile-based analysis. However, in-situ visualization requires integrating the analysis tools with the simulation without significant perturbation of the simulation performance. There are also opportunities to expose novel views of the correlation between the simulated workload and the behavior of the simulator, such as showing local state and metrics, synchronization, and global states and event passing. Finally, feedback from the simulated systems can be used to steer simulations to increase execution efficiency and for more intelligent design space exploration.

Bibliography

[1] Kunal Agrawal, Charles E Leiserson, and Jim Sukha. "Executing task graphs using work-stealing". In: *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE. 2010, pp. 1–12.

[2] Joshua Auld et al. "POLARIS: Agent-based modeling framework development and implementation for integrated travel demand and network and operations simulations". In: *Transportation Research Part C: Emerging Technologies* 64 (2016), pp. 101–116.

[3] Peter D. Barnes et al. "Warp Speed: Executing Time Warp on 1,966,080 Cores". In: *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. SIGSIM PADS '13. Montr©al, Québec, Canada: Association for Computing Machinery, 2013, pp. 327–336. ISBN: 9781450319201. DOI: 10.1145/2486092.2486134. URL: https://doi.org/10.1145/2486092.2486134.

[4] Moshe Ben-akiva et al. *Real Time Simulation of Traffic Demand-Supply Interactions within DynaMIT*.

[5] Abhinav Bhatele et al. "Analyzing Network Health and Congestion in Dragonfly-Based Supercomputers". In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016, pp. 93–102. DOI: 10.1109/IPDPS.2016.123.

[6] Nathan Binkert et al. "The Gem5 Simulator". In: 39.2 (Aug. 2011), pp. 1–7. ISSN: 0163-5964. DOI: 10.1145/2024716.2024718. URL: https://doi.org/10.1145/2024716.2024718.

[7] Azzedine Boukerche and Sajal K Das. "Dynamic load balancing strategies for conservative parallel simulations". In: *Proceedings 11th Workshop on Parallel and Distributed Simulation*. IEEE. 1997, pp. 20–28.

[8] Maximilian Bremer et al. "Speculative Parallel Execution for Local Timestepping". In: *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 2021, pp. 83–94.

[9] Maximilian H Bremer, John D Bachan, and Cy P Chan. "Semi-static and dynamic load balancing for asynchronous hurricane storm surge simulations". In: *2018 IEEE/ACM Parallel Applications Workshop, Alternatives To MPI (PAW-ATM)*. IEEE. 2018, pp. 44–56.

[10] Benjamin Brown. *A Vision for the ASCR Facilities Enterprise*. Tech. rep. Presented to the Advanced Scientific Computing Advisory Committee, Sept. 2021. URL: https://sc.osti.gov/-/media/ascr/ascac/pdf/meetings/202109/Brown_ASCR_Facilities_Vision_202109.pdf (visited on 09/29/2021).

[11] Christopher D Carothers, David Bauer, and Shawn Pearce. "ROSS: A high-performance, low-memory, modular Time Warp system". In: *Journal of Parallel and Distributed Computing* 62.11 (2002), pp. 1648–1669.

[12] Christopher D. Carothers, Kalyan S. Perumalla, and Richard M. Fujimoto. "Efficient Optimistic Parallel Simulations Using Reverse Computation". In: *ACM Trans. Model. Comput. Simul.* 9.3 (July 1999), pp. 224–253. ISSN: 1049-3301. DOI: 10.1145/347823.347828. URL: http://doi.acm.org/10.1145/347823.347828.

[13] Christopher D. Carothers et al. “Visualizing Parallel Simulations in Network Computing Environments: A Case Study”. In: *Proceedings of the 29th Conference on Winter Simulation*. WSC ’97. Atlanta, Georgia, USA: IEEE Computer Society, 1997, pp. 110–117. ISBN: 078034278X. DOI: 10.1145/268437.268459. URL: https://doi.org/10.1145/268437.268459.

[14] Jordi Casas et al. “Traffic Simulation with Aimsun”. In: *Fundamentals of Traffic Simulation*. Ed. by Jaume Barceló. New York, NY: Springer New York, 2010, pp. 173–232. ISBN: 978-1-4419-6142-6.

[15] Umit V Catalyurek et al. “A repartitioning hypergraph model for dynamic load balancing”. In: *Journal of Parallel and Distributed Computing* 69.8 (2009), pp. 711–724.

[16] François E. Cellier and Ernesto Kofman. *Continuous system simulation*. Springer, 2006.

[17] Cy Chan et al. “Mobiliti: Scalable Transportation Simulation Using High-Performance Parallel Computing”. en. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Maui, HI: IEEE, Nov. 2018, pp. 634–641. ISBN: 978-1-72810-321-1. DOI: 10.1109/ITSC.2018.8569397. URL: https://ieeexplore.ieee.org/document/8569397/ (visited on 08/12/2020).

[18] K. M. Chandy and J. Misra. “Asynchronous Distributed Simulation via a Sequence of Parallel Computations”. In: *Commun. ACM* 24.4 (Apr. 1981), pp. 198–206. ISSN: 0001-0782. DOI: 10.1145/358598.358613. URL: https://doi.org/10.1145/358598.358613.

[19] Derek Chiou et al. “FPGA-Accelerated Simulation Technologies (FAST): Fast, Full-System, Cycle-Accurate Simulators”. In: *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. 2007, pp. 249–261. DOI: 10.1109/MICRO.2007.36.

[20] Derek Chiou et al. “The FAST methodology for high-speed SoC/computer simulation”. In: *2007 IEEE/ACM International Conference on Computer-Aided Design*. 2007, pp. 295–302. DOI: 10.1109/ICCAD.2007.4397280.

[21] A.C. Chow, B.P. Zeigler, and Doo Hwan Kim. “Abstract simulator for the parallel DEVS formalism”. In: *Fifth Annual Conference on AI, and Planning in High Autonomy Systems*. 1994, pp. 157–163. DOI: 10.1109/AIHAS.1994.390488.

[22] Moo-Kyoung Chung and Chong-Min Kyung. “Improving Lookahead in Parallel Multiprocessor Simulation Using Dynamic Execution Path Prediction”. In: vol. 2006. Feb. 2006, pp. 11–18. ISBN: 0-7695-2587-3. DOI: 10.1109/PADS.2006.20.

[23] Davide Cingolani, Alessandro Pellegrini, and Francesco Quaglia. “Transparently Mixing Undo Logs and Software Reversibility for State Recovery in Optimistic PDES”. In: *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. SIGSIM PADS ’15. London, United Kingdom: ACM, 2015, pp. 211–222. ISBN: 978-1-4503-3583-6. DOI: 10.1145/2769458.2769482. URL: http://doi.acm.org/10.1145/2769458.2769482.

[24] Davide Cingolani et al. “Dealing with Reversibility of Shared Libraries in PDES”. In: *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. SIGSIM-PADS ’17. Singapore, Republic of Singapore: ACM, 2017, pp. 41–52. ISBN: 978-1-4503-4489-0. DOI: 10.1145/3064911.3064927. URL: http://doi.acm.org/10.1145/3064911.3064927.

[25] *Co-Design of Exascale Storage Architectures and Science Data Facilities*. https://github.com/CODES-org. Accessed: 2022-02-22.

[26] Tim Coopmans et al. “NetSquid, a NETwork Simulator for QUantum Information using Discrete events”. In: *Communications Physics* 4.1 (July 2021), p. 164. ISSN: 2399-3650. DOI: 10.1038/s42005-021-00647-8. URL: https://doi.org/10.1038/s42005-021-00647-8.

[27] Patrick Crawford et al. “Some properties of communication behaviors in discrete-event simulation models”. In: *2017 Winter Simulation Conference (WSC)*. 2017, pp. 1025–1036. DOI: 10.1109/WSC.2017.8247852.

[28] E. Deelman et al. “Improving lookahead in parallel discrete event simulations of large-scale applications using compiler analysis”. In: *Proceedings 15th Workshop on Parallel and Distributed Simulation*. 2001, pp. 5–13. DOI: 10.1109/PADS.2001.924616. URL: https://ieeexplore.ieee.org/document/924616.

[29] Ewa Deelman and Boleslaw K Szymanski. “Dynamic load balancing in parallel discrete event simulation for spatially explicit problems”. In: *Proceedings. Twelfth Workshop on Parallel and Distributed Simulation PADS’98 (Cat. No. 98TB100233)*. IEEE. 1998, pp. 46–53.

[30] Ali Eker et al. “Load-Aware Dynamic Time Synchronization in Parallel Discrete Event Simulation”. In: *Proceedings of the 2021 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. 2021, pp. 95–105.

[31] Roland Ewald et al. “A Simulation Approach to Facilitate Parallel and Distributed Discrete-Event Simulator Development”. In: *2006 Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications*. 2006, pp. 209–218. DOI: 10.1109/DS-RT.2006.5.

[32] Richard Fujimoto, Kalyan Perumalla, and George Riley. *Network Simulation: Synthesis Lectures on Communication Networks*. USA: Morgan and Claypool, 2006. ISBN: 978-1598291100.

[33] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*. 1st. USA: Wiley, 2000. ISBN: 978-0-471-18383-9.

[34] Richard M. Fujimoto. “Parallel Discrete Event Simulation”. In: *Commun. ACM* 33.10 (Oct. 1990), pp. 30–53. ISSN: 0001-0782. DOI: 10.1145/84537.84545. URL: https://doi.org/10.1145/84537.84545.

[35] Richard M. Fujimoto et al. “Parallel discrete event simulation: The making of a field”. In: *2017 Winter Simulation Conference (WSC)*. 2017, pp. 262–291. DOI: 10.1109/WSC.2017.8247793.

[36] Masatoshi Hanai et al. “Exact-Differential Simulation: Differential Processing of Large-Scale Discrete Event Simulations”. In: *ACM Trans. Model. Comput. Simul.* 29.3 (June 2019). ISSN: 1049-3301. DOI: 10.1145/3301499. URL: https://doi.org/10.1145/3301499.

[37] Christopher Hannon et al. “Just-in-time parallel simulation”. In: *2018 Winter Simulation Conference (WSC)*. IEEE. 2018, pp. 640–651.

[38] Muhammad Amber Hassaan, Donald D. Nguyen, and Keshav Pingali. “Kinetic Dependence Graphs”. In: *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2015, Istanbul, Turkey, March 14-18, 2015*. Ed. by Özcan Özturk, Kemal Ebcioglu, and Sandhya Dwarkadas. ACM, 2015, pp. 457–471. DOI: 10.1145/2694344.2694363. URL: https://doi.org/10.1145/2694344.2694363.

[39] Xiaolin Hu, Yi Sun, and Lewis Ntaimo. “DEVS-FIRE: design and application of formal discrete event wildfire spread and suppression models”. In: *SIMULATION* 88.3 (2012), pp. 259–279.

[40] Katherine E Isaacs et al. “State of the Art of Performance Visualization.” In: *EuroVis (STARs)*. 2014.

[41] V. Jalili-Marandi and V. Dinavahi. “SIMD-Based Large-Scale Transient Stability Simulation on the Graphics Processing Unit”. In: *IEEE Transactions on Power Systems* 25.3 (2010), pp. 1589–1599.

[42] David R. Jefferson. “Virtual Time”. In: *ACM Trans. Program. Lang. Syst.* 7.3 (July 1985), pp. 404–425. ISSN: 0164-0925. DOI: 10.1145/3916.3988. URL: https://doi.org/10.1145/3916.3988.

[43] David R. Jefferson and Peter D. Barnes. “Virtual Time III: Unification of Conservative and Optimistic Synchronization in Parallel Discrete Event Simulation”. In: *Proceedings of the 2017 Winter Simulation Conference*. WSC ’17. Las Vegas, Nevada: IEEE Press, 2017. ISBN: 9781538634271.

[44] Shuangshuang Jin et al. “A novel application of parallel betweenness centrality to power grid contingency analysis”. In: *International Parallel and Distributed Processing Symposium/International Parallel Processing Symposium*. 2010, pp. 1–7. DOI: 10.1109/IPDPS.2010.5470400.

[45] Hartmut Kaiser et al. “Hpx: A task based programming model in a global address space”. In: *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*. 2014, pp. 1–11.

[46] George Karypis and Vipin Kumar. “A fast and high quality multilevel scheme for partitioning irregular graphs”. In: *SIAM Journal on scientific Computing* 20.1 (1998), pp. 359–392.

[47] Aaditya Landge et al. “Visualizing Network Traffic to Understand the Performance of Massively Parallel Simulations”. In: *IEEE Trans. on Visualization and Computer Graphics* 18 (Dec. 2012), pp. 2467–2476. DOI: 10.1109/TVCG.2012.286.

[48] Bob Lantz, Brandon Heller, and Nick McKeown. “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks”. In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: Association for Computing Machinery, 2010. ISBN: 9781450304092. DOI: 10.1145/1868447.1868466. URL: https://doi.org/10.1145/1868447.1868466.

[49] Justin M. LaPre, Elsa J. Gonsiorowski, and Christopher D. Carothers. “LORAIN: A Step Closer to the PDES ’Holy Grail’”. In: *Proceedings of the 2Nd ACM SIGSIM/PADS Conference on Principles of Advanced Discrete Simulation*. SIGSIM-PADS ’14. Denver, Colorado, USA: ACM, 2014, pp. 3–14. ISBN: 978-1-4503-2794-7. DOI: 10.1145/2601381.2601397. URL: http://doi.acm.org/10.1145/2601381.2601397.

[50] Chee Wai Lee, Terry L Wilmarth, and Laxmikant V Kalé. “Performance visualization and analysis of parallel discrete event simulations with projections”. In: *Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign, Tech. Rep.* (2005), pp. 05–19.

[51] Jianping Kelvin Li et al. “A Visual Analytics Framework for Analyzing Parallel and Distributed Computing Applications”. In: *2019 IEEE Visualization in Data Science (VDS)*. 2019, pp. 1–9. DOI: 10.1109/VDS48975.2019.8973380.

[52] Jianping Kelvin Li et al. “Visual Analytics Techniques for Exploring the Design Space of Large-Scale High-Radix Networks”. In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. 2017, pp. 193–203. DOI: 10.1109/CLUSTER.2017.26.

[53] Jonathan Lifflander, Sriram Krishnamoorthy, and Laxmikant V Kale. "Work stealing and persistence-based load balancers for iterative overdecomposed applications". In: *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing*. 2012, pp. 137–148.

[54] Zhongwei Lin and Yiping Yao. "Load Balancing for Parallel Discrete Event Simulation of Stochastic Reaction and Diffusion". In: *2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*. IEEE. 2015, pp. 609–614.

[55] Qi Liu and Gabriel Wainer. "Parallel Environment for DEVS and Cell-DEVS Models". In: *SIMULATION* 83.6 (2007), pp. 449–471. DOI: 10.1177/0037549707085084. eprint: https://doi.org/10.1177/0037549707085084. URL: https://doi.org/10.1177/0037549707085084.

[56] Yarden Livnat et al. *DragonView: Toward Understanding Network Interference in Dragonfly-based Supercomputers*. Tech. rep. Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2016.

[57] *Lopez, P.A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Fl¨otter¨od, Y.P., Hilbrich, R., Lu¨cken, L., Rummel, J., Wagner, P., Wießner, E.: Microscopic traffic simulation using sumo. In: The 21st IEEE International Conference on Intelligent Transportation Systems. IEEE (2018). URL https://elib.dlr.de/124092/ - Google Search*.

[58] Hani Mahmassani. "Dynamic Traffic Simulation and Assignment: Models, Algorithms and Application to ATIS / ATMS Evaluation and Operation". In: Jan. 1998, pp. 104–135. ISBN: 978-3-642-08428-7. DOI: 10.1007/978-3-662-03514-6_5.

[59] *MATSim.org*. URL: https://www.matsim.org/ (visited on 08/12/2020).

[60] Eric P Mikida. "Adaptive techniques for scalable optimistic parallel discrete event simulation". PhD thesis. University of Illinois at Urbana-Champaign, 2019.

[61] A. Molina-Garcí, F. Bouffard, and D.S. Kirschen. "Decentralized Demand-Side Contribution to Primary Frequency Control". In: *IEEE Transactions on Power Systems* 26.1 (Feb. 2011), pp. 411–419.

[62] Misbah Mubarak et al. "Enabling parallel simulation of large-scale HPC network systems". In: *IEEE Transactions on Parallel and Distributed Systems* 28.1 (2016), pp. 87–100.

[63] Sara Mullen and Getiria Onsongo. "Decentralized agent-based underfrequency load shedding". In: *Integrated Computer-Aided Engineering* 17.4 (2010), pp. 321–329.

[64] Alexandre Muzy and Bernard P Zeigler. "Introduction to the activity tracking paradigm in component-based simulation". In: *Open Cybernetics & Systemics Journal* 2 (2008), pp. 30–38.

[65] Alexandre Muzy et al. "The Activity-tracking paradigm in discrete-event modeling and simulation: The case of spatially continuous distributed systems". In: *Simulation* 87.5 (2011), pp. 449–464.

[66] K Nagel et al. *Dynamic traffic assignment on parallel computers*. Tech. rep. Los Alamos National Lab., NM (United States), 1998.

[67] David M. Nicol. "Principles of Conservative Parallel Simulation". In: *Proceedings of the 28th Conference on Winter Simulation*. WSC '96. Coronado, California, USA: IEEE Computer Society, 1996, pp. 128–135. ISBN: 0780333837. DOI: 10.1145/256562.256591. URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.414.7539.

[68] Sergei Nikolaev et al. “Pushing the Envelope in Distributed Ns-3 Simulations: One Billion Nodes”. In: *Proceedings of the 2015 Workshop on Ns-3*. WNS3 ’15. Barcelona, Spain: Association for Computing Machinery, 2015, pp. 67–74. ISBN: 9781450333757. DOI: 10.1145/2756509.2756525. URL: https://doi.org/10.1145/2756509.2756525.

[69] James Nutaro. *Building software for simulation*. Wiley, 2010.

[70] James Nutaro. “Toward a Theory of Superdense Time in Simulation Models”. In: *ACM Trans. Model. Comput. Simul.* 30.3 (May 2020). ISSN: 1049-3301. DOI: 10.1145/3379489. URL: https://doi.org/10.1145/3379489.

[71] Tomas Oppelstrup et al. “SPOCK: Exact Parallel Kinetic Monte-Carlo on 1.5 Million Tasks”. In: *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. SIGSIM-PADS ’16. Banff, Alberta, Canada: ACM, 2016, pp. 127–130. ISBN: 978-1-4503-3742-7. DOI: 10.1145/2901378.2901403. URL: http://doi.acm.org/10.1145/2901378.2901403.

[72] N. Osawa, K. Hisano, and T. Yuba. “A visual performance debugging system for parallel programs”. In: *Proceedings of HICSS-29: 29th Hawaii International Conference on System Sciences*. Vol. 1. 1996, 300–309 vol.1. DOI: 10.1109/HICSS.1996.495475.

[73] Benno Overeinder and Peter Sloot. “Parallel Discrete Event Simulation Performance Modeling and Evaluation”. In: (May 2001).

[74] Eunjung Park et al. “Parameterized benchmarking of parallel discrete event simulation systems: Communication, computation, and memory”. In: *2015 Winter Simulation Conference (WSC)*. IEEE. 2015, pp. 2836–2847.

[75] A. Pellegrini. “Hijacker: Efficient static software instrumentation with applications in high performance computing: Poster paper”. In: *High Performance Computing and Simulation (HPCS), 2013 International Conference on*. July 2013, pp. 650–655. DOI: 10.1109/HPCSim.2013.6641486.

[76] A. Pellegrini, R. Vitali, and F. Quaglia. “Autonomic State Management for Optimistic Simulation Platforms”. In: *IEEE Transactions on Parallel and Distributed Systems* 26.6 (June 2015), pp. 1560–1569. ISSN: 1045-9219. DOI: 10.1109/TPDS.2014.2323967.

[77] Alessandro Pellegrini and Francesco Quaglia. “A Fine-Grain Time-Sharing Time Warp System”. In: *ACM Trans. Model. Comput. Simul.* 27.2 (May 2017), 10:1–10:25. ISSN: 1049-3301. DOI: 10.1145/3013528. URL: http://doi.acm.org/10.1145/3013528.

[78] François Pellegrini and Jean Roman. “Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs”. In: *International Conference on High-Performance Computing and Networking*. Springer. 1996, pp. 493–498.

[79] Kalyan S. Perumalla. “A Systems Approach to Scalable Transportation Network Modeling”. In: *Proceedings of the 2006 Winter Simulation Conference*. 2006, pp. 1500–1507. DOI: 10.1109/WSC.2006.322919.

[80] Kalyan S. Perumalla. *Introduction to Reversible Computing*. CRC Press Book, 2013, p. 325.

[81] Kalyan S. Perumalla. “Parallel and Distributed Simulation: Traditional Techniques and Recent Advances”. In: *Proceedings of the 2006 Winter Simulation Conference*. 2006, pp. 84–95. DOI: 10.1109/WSC.2006.323041.

[82] Kalyan S. Perumalla and Maksudul Alam. "Mesoscopic Modeling and Rapid Simulation of Incremental Changes in Epidemic Scenarios on GPUs". In: *Journal of the Indian Institute of Science* 101.3 (July 2021), pp. 357–370. ISSN: 0019-4964. DOI: 10.1007/s41745-021-00253-1.

[83] Kalyan S. Perumalla, James J. Nutaro, and Srikanth B. Yoginath. "Towards High Performance Discrete-Event Simulations of Smart Electric Grids". In: *Proceedings of the First International Workshop on High Performance Computing, Networking and Analytics for the Power Grid*. HiPCNA-PG '11. Seattle, Washington, USA: Association for Computing Machinery, 2011, pp. 51–58. ISBN: 9781450310611. DOI: 10.1145/2096123.2096135. URL: https://doi.org/10.1145/2096123.2096135.

[84] Kalyan S. Perumalla, Alfred J. Park, and Vinod Tipparaju. "Discrete Event Execution with One-Sided and Two-Sided GVT Algorithms on 216,000 Processor Cores". In: *ACM Trans. Model. Comput. Simul.* 24.3 (June 2014). ISSN: 1049-3301. DOI: 10.1145/2611561. URL: https://doi.org/10.1145/2611561.

[85] Kalyan S. Perumalla and Sudip K. Seal. "Discrete event modeling and massively parallel execution of epidemic outbreak phenomena". In: *SIMULATION* 88.7 (2012), pp. 768–783. DOI: 10.1177/0037549711413001.

[86] Kalyan S. Perumalla et al. "Interactive, graphical processing unit-based evaluation of evacuation scenarios at the state scale". In: *SIMULATION* 88.6 (2012), pp. 746–761. DOI: 10.1177/0037549711425236. eprint: https://doi.org/10.1177/0037549711425236. URL: https://doi.org/10.1177/0037549711425236.

[87] Kalyan S. Perumalla et al. "Scalable RTI-Based Parallel Simulation of Networks". In: *Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation*. PADS '03. USA: IEEE Computer Society, 2003, p. 97. ISBN: 0769519709.

[88] Dierk Raabe. "Cellular Automata in Materials Science with Particular Reference to Recrystallization Simulation". In: *Annual Review of Materials Research* 32.1 (2002), pp. 53–76.

[89] Hesham A. Rakha, Kyoungho Ahn, and Kevin Moran. "INTEGRATION Framework for Modeling Eco-routing Strategies: Logic and Preliminary Results". en. In: *International Journal of Transportation Science and Technology* 1.3 (Sept. 2012), pp. 259–274. ISSN: 2046-0430. DOI: 10.1260/2046-0430.1.3.259. URL: http://www.sciencedirect.com/science/article/pii/S2046043016301629 (visited on 08/12/2020).

[90] George F. Riley and Thomas R. Henderson. "The ns-3 Network Simulator." In: *Modeling and Tools for Network Simulation*. Ed. by Klaus Wehrle, Mesut Günes, and James Gross. Springer, 2010, pp. 15–34. ISBN: 978-3-642-12330-6. URL: http://dblp.uni-trier.de/db/books/collections/Wehrle2010.html#RileyH10.

[91] Arun F. Rodrigues et al. "The structural simulation toolkit". In: *ACM SIGMETRICS Performance Evaluation Review* 38.4 (2011), pp. 37–42.

[92] Caitlin Ross et al. "Visual Data-Analytics of Large-Scale Parallel Discrete-Event Simulations". In: *2016 7th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. 2016, pp. 87–97. DOI: 10.1109/PMBS.2016.014.

[93] Hans Sagan. *Space-filling curves*. Springer Science & Business Media, 2012.

[94] Nandakishore Santhi, Stephan Eidenbenz, and Jason Liu. "The simian concept: Parallel discrete event simulation with interpreted languages and just-in-time compilation". In: *2015 Winter Simulation Conference (WSC)*. IEEE. 2015, pp. 3013–3024.

[95] E. Santini et al. "Hardware-Transactional-Memory Based Speculative Parallel Discrete Event Simulation of Very Fine Grain Models". In: *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*. Dec. 2015, pp. 145–154. DOI: 10.1109/HiPC.2015.45.

[96] Markus Schordan et al. "Automatic Generation of Reversible C++ Code and Its Performance in a Scalable Kinetic Monte-Carlo Application". In: *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. SIGSIM-PADS '16. Banff, Alberta, Canada: ACM, 2016, pp. 111–122. ISBN: 978-1-4503-3742-7. DOI: 10.1145/2901378.2901394. URL: http://doi.acm.org/10.1145/2901378.2901394.

[97] Markus Schordan et al. "Generation of Reversible C++ Code for Optimistic Parallel Discrete Event Simulation". In: *New Generation Comput.* 36.3 (2018), pp. 257–280. DOI: 10.1007/s00354-018-0038-2. URL: https://doi.org/10.1007/s00354-018-0038-2.

[98] Markus Schordan et al. "Reverse Code Generation for Parallel Discrete Event Simulation". English. In: *Reversible Computation*. Ed. by Jean Krivine and Jean-Bernard Stefani. Vol. 9138. Lecture Notes in Computer Science. Springer, 2015, pp. 95–110. ISBN: 978-3-319-20859-6. DOI: 10.1007/978-3-319-20860-2_6. URL: http://dx.doi.org/10.1007/978-3-319-20860-2_6.

[99] Markus Schordan et al. "Reversible Languages and Incremental State Saving in Optimistic Parallel Discrete Event Simulation". In: *Reversible Computation: Extending Horizons of Computing: Selected Results of the COST Action IC1405*. Ed. by Irek Ulidowski et al. Cham: Springer International Publishing, 2020, pp. 187–207. ISBN: 978-3-030-47361-7. DOI: 10.1007/978-3-030-47361-7_9. URL: https://doi.org/10.1007/978-3-030-47361-7_9.

[100] Catherine D. Schuman et al. "Simulating and Estimating the Behavior of a Neuromorphic Co-Processor". In: *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*. PMES'17. Denver, CO, USA: Association for Computing Machinery, 2017, pp. 8–14. ISBN: 9781450351263. DOI: 10.1145/3149526.3149529. URL: https://doi.org/10.1145/3149526.3149529.

[101] Colin Sheppard et al. *Modeling plug-in electric vehicle charging demand with BEAM: the framework for behavior energy autonomy mobility*. en. Tech. rep. 1398472. May 2017, p. 1398472. DOI: 10.2172/1398472. URL: http://www.osti.gov/servlets/purl/1398472/ (visited on 08/12/2020).

[102] Jiwu Shu, Wei Xue, and Weimin Zheng. "A parallel transient stability simulation for power systems". In: *IEEE Transactions on Power Systems* 20.4 (2005), pp. 1709–1717.

[103] Carmen Sigovan et al. "A Visual Network Analysis Method for Large-Scale Parallel I/O Systems". In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 2013, pp. 308–319. DOI: 10.1109/IPDPS.2013.96.

[104] Viliam Solcany and J. Safarík. "The lookahead in a user-transparent conservative parallel simulator". In: *Proceedings 16th Workshop on Parallel and Distributed Simulation* (2002), pp. 9–14.

[105] Tapas K Som and Robert G Sargent. "Model structure and load balancing in optimistic parallel discrete event simulation". In: *Proceedings Fourteenth Workshop on Parallel and Distributed Simulation*. IEEE. 2000, pp. 147–154.

[106] *Structural Simulation Toolkit*. http://www.sst-simulator.org. Accessed: 2022-02-22.

[107] Tim Süß et al. "A system for aggregated visualization of multiple parallel discrete event simulations". In: *2008 IEEE International Symposium on Parallel and Distributed Processing with Applications*. IEEE. 2008, pp. 587–593.

[108] D.S. Svyetlichnyy. "Modelling of the microstructure: From classical cellular automata approach to the frontal one". In: *Computational Materials Science* 50.1 (2010), pp. 92–97. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2010.07.011. URL: https://www.sciencedirect.com/science/article/pii/S0927025610004210.

[109] *The gem5 Simulator*. http://www.gem5.org. Accessed: 2022-02-22.

[110] Sunil Thulasidasan et al. "Explicit spatial scattering for load balancing in conservatively synchronized parallel discrete event simulations". In: *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*. IEEE. 2010, pp. 1–8.

[111] D. Trudnowski, M. Donnelly, and E. Lightner. "Power-System Frequency and Stability Control using Decentralized Intelligent Loads". In: *Proceedings of the 2005 IEEE Power Engineering Society T & D Conference and Expo*. 2006, pp. 1453–1459.

[112] Shu-Jen Tsai et al. "Frequency Sensitivity and Electromechanical Propagation Simulation Study in Large Power Systems". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 54.8 (2007), pp. 1819–1828.

[113] Juliette Ugirumurera et al. "A unified software framework for solving traffic assignment problems". In: *arXiv:1804.11026 [cs]* (Apr. 2018). arXiv: 1804.11026. URL: http://arxiv.org/abs/1804.11026 (visited on 08/12/2020).

[114] Jeffrey D. Ullman. "NP-complete scheduling problems". In: *Journal of Computer and System sciences* 10.3 (1975), pp. 384–393.

[115] Hans Vangheluwe and Ghislain C. Vansteenkiste. "The Cellular Automata Formalism and Its Relationship to DEVS". In: *Proceedings of the 14th European Simulation Multiconference on Simulation and Modelling: Enablers for a Better Quality of Life*. SCS Europe, 2000, pp. 800–810. ISBN: 1565552040.

[116] George Vulov et al. "The Backstroke Framework for Source Level Reverse Computation Applied to Parallel Discrete Event Simulation". In: *Proceedings of the Winter Simulation Conference*. WSC '11. Phoenix, Arizona: Winter Simulation Conference, 2011, pp. 2965–2979. URL: http://dl.acm.org/citation.cfm?id=2431518.2431870.

[117] Gabriel A Wainer. "The Cell–DEVS formalism as a method for activity tracking in spatial modelling and simulation". In: *International Journal of Simulation and Process Modelling* 10.1 (2015), pp. 19–38.

[118] Gabriel A. Wainer. "Cellular Modeling with Cell–DEVS: A Discrete-Event Cellular Automata Formalism". In: *Cellular Automata*. Ed. by Jarosław Wąs, Georgios Ch. Sirakoulis, and Stefania Bandini. Cham: Springer International Publishing, 2014, pp. 6–15.

[119] Gabriel A. Wainer and Norbert Giambiasi. "Application of the Cell-DEVS paradigm for cell spaces modeling and simulation". In: *Simulation* 71.1 (Jan. 2001), pp. 22–39. URL: http://cell-devs.sce.carleton.ca/publications/2001/WG01.

[120] Tang Wenjie et al. "A work-stealing based dynamic load balancing algorithm for conservative parallel discrete event simulation". In: *2017 Winter Simulation Conference (WSC)*. IEEE. 2017, pp. 798–809.

[121] Linda F Wilson and Wei Shen. “Experiments in load migration and dynamic load balancing in speedes”. In: *1998 Winter Simulation Conference. Proceedings (Cat. No. 98CH36274)*. Vol. 1. IEEE. 1998, pp. 483–490.

[122] Xiaoliang Wu et al. *SeQUeNCe: A Customizable Discrete-Event Simulator of Quantum Networks*. 2020. arXiv: 2009.12000 [quant-ph].

[123] Pavan Yedavalli, Krishna Kumar, and Paul Waddell. “Microsimulation Analysis for Network Traffic Assignment (MANTA) at Metropolitan-Scale for Agile Transportation Planning”. In: *Transportmetrica A: Transport Science* just-accepted (2021), pp. 1–19.

[124] Srikanth B. Yoginath and Kalyan S. Perumalla. “Reversible discrete event formulation and optimistic parallel execution of vehicular traffic models”. en. In: *International Journal of Simulation and Process Modelling* 5.2 (2009), p. 104. ISSN: 1740-2123, 1740-2131. DOI: 10.1504/IJSPM.2009.028624. URL: http://www.inderscience.com/link.php?id=28624 (visited on 08/12/2020).

[125] Srikanth B. Yoginath, Kalyan S. Perumalla, and Brian J. Henz. “Virtual machine-based simulation platform for mobile ad-hoc network-based cyber infrastructure”. In: *The Journal of Defense Modeling and Simulation* 12.4 (2015), pp. 439–456. DOI: 10.1177/1548512915591050. eprint: https://doi.org/10.1177/1548512915591050. URL: https://doi.org/10.1177/1548512915591050.

[126] Tetsuo Yokoyama, Holger Bock Axelsen, and Robert Glück. “Principles of a Reversible Programming Language”. In: *Proceedings of the 5th Conference on Computing Frontiers*. CF ’08. Ischia, Italy: ACM, 2008, pp. 43–54. ISBN: 978-1-60558-077-7. DOI: 10.1145/1366230.1366239. URL: http://doi.acm.org/10.1145/1366230.1366239.

[127] Tetsuo Yokoyama and Robert Glück. “A Reversible Programming Language and Its Invertible Self-interpreter”. In: *Proceedings of the 2007 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-based Program Manipulation*. PEPM ’07. Nice, France: ACM, 2007, pp. 144–153. ISBN: 978-1-59593-620-2. DOI: 10.1145/1244381.1244404. URL: http://doi.acm.org/10.1145/1244381.1244404.

[128] Tetsuo Yokoyama and Robert Glück. “A reversible programming language and its invertible self-interpreter”. In: *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2007, Nice, France, January 15-16, 2007*. Ed. by G. Ramalingam and Eelco Visser. ACM, 2007, pp. 144–153. DOI: 10.1145/1244381.1244404. URL: https://doi.org/10.1145/1244381.1244404.

[129] B.P. Zeigler. “Discrete event models for cell space simulation”. In: *International Journal of Theoretical Physics* 21 (1982), pp. 573–588.

[130] Bernard P. Zeigler, Alexandre Muzy, and Ernesto Kofman. *Theory of Modeling and Simulation, 3rd edition*. Elsevier, 2018.

[131] Gengbin Zheng et al. “Hierarchical load balancing for charm++ applications on large supercomputers”. In: *2010 39th International Conference on Parallel Processing Workshops*. IEEE. 2010, pp. 436–444.