Sampling Policies for Near-Optimal Device Choice in Parallel Simulations on CPU/GPU Platforms

Philipp Andelfinger*, Alessandro Pellegrini†, and Romolo Marotta†
*University of Rostock, Germany
†Tor Vergata University of Rome, Italy

***Abstract*—Heterogeneous hardware platforms comprised of CPUs, GPUs, and other accelerators offer the opportunity to choose the best-suited device for executing a given scientific simulation in order to minimize execution time and energy consumption. To this end, the recently proposed "Follow the Leader" approach dynamically selects a suitable device based on runtime performance measurements during speculative discrete-event simulations. A currently active "leader" device is periodically challenged by a "follower" device in order to negotiate the new leader. The optimality of the device choices and the associated overhead depends critically on the challenge frequency and timing. Here, we explore policies to schedule challenges with the goal of attaining Pareto-optimal combinations of execution time and energy consumption. Several heuristics are first evaluated in an abstract fashion using a "meta-simulation" by mimicking the progress and energy consumption of an idealized co-execution. In this setting, we optimize the heuristics' tuning parameters to assess their relative merits in near-optimal configurations when compared to challenge timings based on perfect knowledge. We find that under challenging stochastic workloads based on a class of mean-reverting random walks, the best heuristics can closely approximate the execution time and energy consumption achievable under an optimal device choice. Empirical support for this observation is given by measurements of a CPU/GPU co-execution of the Time Warp algorithm on physical hardware.**

***Index Terms*—GPU-based simulation, Time Warp, heterogeneous hardware, discrete-event simulation**

I. Introduction

Parallel Discrete Event Simulation (PDES) [1] is a powerful tool for what-if analysis and studying complex systems across fields like telecommunications, transportation, healthcare, and military applications. These simulations are becoming increasingly complex and require significant computational power. Modern heterogeneous exascale systems [2], which include a mix of CPUs, GPUs, and other accelerators, are well-suited for this task. However, running PDES on such systems is challenging because different hardware classes exhibit varying performance and energy profiles based on the workload [3]. As model dynamics can change frequently, determining the optimal deployment at runtime is difficult, necessitating adaptive strategies for efficient and energy-effective simulations.

Recently, the "Follow the Leader" (FTL) approach [4] has been proposed for dynamically selecting the optimal device based on real-time performance measurements during speculative discrete-event simulations based on the Time Warp synchronization protocol [5]. The approach involves periodically challenging the currently active "leader" device with a "follower" device to negotiate a potential new leader. The effectiveness and overhead of this method depend heavily on the frequency and timing of these challenges. This runtime scheduling problem is characterized by binary decisions (also known as "bang-bang control" [6]), such as choosing between running the model for a certain time window $[t_{VT}, t'_{VT}]$ on either CPUs or a GPU.

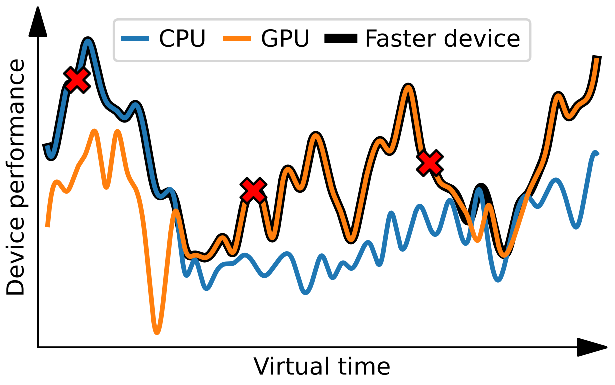

Fig. 1: The performance of different devices varies over time with the workload dynamics. Our goal is to heuristically choose appropriate times to collect performance observations ($\times$) in order to select the fastest device while minimizing cost. For instance, the third observation in the figure does not lead to a device change and should thus be omitted.

Our goal in this paper is to optimize the scheduling of observations (i.e., running a challenge), specifically exploring various sampling policies that can guide these decisions. The core problem we address involves two unknown performance trajectories, $f(t_{VT})$ and $g(t_{VT})$, one associated with the execution of the model on CPUs, the other on GPU. At specific times t_{VT}, we can observe these trajectories, albeit at a certain cost. Our goal is to approximate the function $h(t_{VT}) = \max(f(t_{VT}), g(t_{VT}))$ over the interval $[t_{VT}, t'_{VT}]$ by optimally selecting the times to observe the function values. This process is summarized in Figure 1.

We focus on developing and evaluating policies for scheduling these challenges to achieve Pareto-optimal combinations of execution time and energy consumption. We employ a "meta-simulation" framework based on [7] that abstractly represents the co-execution of simulations on CPUs and GPUs. This setup allows us to optimize the tuning parameters of various heuristics and compare their performance against challenge timings derived from perfect knowledge.

To this end, we systematically evaluate different sampling policies, providing insights into their efficacy under stochastic workloads that mimic real-world scenarios. The results help refine strategies for device selection in heterogeneous platforms, ultimately enhancing both performance and energy efficiency in parallel simulations. We validate the results obtained through the meta-simulation framework by exercising our policies on actual heterogeneous hardware.

The remainder of this paper is structured as follows. Related work is discussed in Section II. The heuristics considered to determine the best-suited executed hardware are presented in Section III. Section IV discusses the study conducted using the meta-simulation methodology. Results from the experimental assessment are reported in Section V.

II. Related Work

The effects of simulation models' varying workloads on general-purpose runtime environments have led the research community to dedicate great care to studying self-optimization policies that can improve performance and energy efficiency, although only a few have explicitly considered the underlying availability of heterogeneous computing architectures. Indeed, while generic DES engines running on coprocessors (e.g., GPU or FPGA) have appeared in the literature (see, e.g., [8]–[11]), the first attempts to optimize the simulation execution on heterogeneous systems are very recent [4], [12], [13], although only the work in [4] has considered generic DES systems.

In speculative simulation, particularly based on the Time Warp synchronization protocol [5], many proposals have tried to optimize runtime control parameters to reduce the memory footprint and housekeeping costs, based on explicit cost models [14]–[19], on temporal measurements [20]–[22], on heuristics [23]–[25] or on machine-learning approaches [26]–[29], sometimes also considering energy efficiency [30]. In our work, as we assume that the workload dynamics are unknown a priori and highly variable and that individual simulation runs can be brief, there is only limited scope for collecting enough observations to train a machine learning model, e.g., by reinforcement learning. Instead, we compare a number of heuristic policies that can be parameterized for given workloads.

Much effort has been dedicated to identifying the best-suited accelerator to run a certain workload in the general landscape of heterogeneous high-performance computing (HPC). Several performance models have been proposed to estimate the execution time of kernels on CUDA GPUs [31]–[33], also using machine-learning approaches [34]–[36] or considering program skeletons [37]. These approaches are specific to some specific accelerator or program organization. Some approaches also target particular applications, such as iterative stencil loops [38] or sparse matrices [39]. Other models are tailored to help developers optimize their code once performance bottlenecks have been identified [40]–[43]. Energy efficiency when selecting one particular family of accelerators has also been studied through models to aid in the identification of the best-suited HPC setup [44]–[46]. Unlike these works, we select the best-suited accelerator for a certain execution phase based on observing the model's behavior in a certain time window. This strategy avoids the simplifications introduced by modeling approaches. Conversely, we rely on models and heuristics to determine the best-suited instant when to run a challenge to limit performance impairment and energy wastage.

Our proposal to optimize execution on heterogeneous architectures is reminiscent of approaches for simulation algorithm selection [47], [48], or the more general n-version programming approach [49], [50] that leverages multiple versions of the same program to improve non-functional properties such as fault tolerance or execution timeliness. The main difference with our approach is that the different versions of the same model that we consider exercise different hardware instances in ways that are hard to predict due to the inherent architectural difference and because we do not assume the stability of the model. Therefore, we adopt explicit challenges to determine the best-suited architecture for execution, coupled with black-box heuristics to reduce the impact of such challenges on the overall performance and energy footprint.

Our approach also shares some objectives with co-simulation techniques [51], where the integrated simulation of a coupled system can be accomplished by combining the simulations of its components. In particular, we could consider the different instances of the simulation models executed on CPUs and GPUs as independent components that must be periodically synchronized (i.e., to support a correct execution of a challenge and to resume the simulation on the selected leader). Several synchronization methods have been proposed in the literature for this goal (see, e.g., [52], [53]). The High-Level Architecture (HLA) standard [54] also has dedicated significant space to this problem. Nevertheless, our main focus is on execution performance and energy efficiency rather than on interoperability, therefore our timeliness requirements are much more stringent.

Having multiple instances of the same simulation model running at the same time recalls the concept of simulation cloning [55]. This method dynamically initiates multiple simulation instances at decision points within a simulation, facilitating the concurrent exploration of different execution paths. By sharing common computation outcomes, cloning enables a more thorough investigation of what-if scenarios. However, our approach differs by focusing on a single simulation trajectory while utilizing multiple hardware devices. We also share computation results among instances, but since we concentrate on a single trajectory, we meticulously synchronize the state and event queues among the replicas to maintain consistent execution.

In the simulation field, various performance prediction approaches similar to our meta-simulation have been proposed. Critical path analysis [56] models the parallelism of a simulation model based on inter-event dependencies to highlight how parallel execution should be, although irrespective of the hardware used. This is overcome in [57], [58], where descriptive knowledge of the hardware or results from micro-benchmarking are considered. Characteristics of the executed model are also taken into account in [59], where both the characteristics of the model and the hardware environment are taken into account, although focusing on network simulations. In our work, a time-stepped meta-simulation generates predictions not just of execution times, but also of energy consumption.

III. Sampling Heuristics

In the following, we describe four heuristic for scheduling challenges throughout simulation runs, each associated with one or more tuning parameters. The heuristics are intended to be applicable without prior knowledge of the workload and without requiring a learning phase. Hence, the scheduling decisions directly rely on events and variables that can be easily observed at runtime.

A. Fixed Period

This baseline heuristic used in our previous work [4] schedules challenges at a constant period $d \in \mathbb{R}_+$ in virtual time, with d being a tunable parameter.

B. Additive Increase, Multiplicative Decrease (AIMD)

Inspired by TCP congestion control [60], the rationale behind this heuristic is to gradually extend the interval between challenges as long as no change in the leader occurs. When a new leader is chosen, the interval is sharply reduced, reflecting the assumption that further imminent leader changes are likely given the recent change in relative performance among the devices. On a challenge i, the delay d_{i+1} in wall-clock time to the next challenge is determined by

$$d_{i+1} = \begin{cases} d_i + a, \text{if a leader change occurred} \\ bd_i, \text{else.} \end{cases}$$

Here, $a \in \mathbb{R}_+$ and $b \in (0, 1]$ are tuning parameters representing the additive or multiplicative change at each challenge.

C. Energy Budget

This heuristic focuses on the benefit in energy consumption by running on the respective fastest device. New challenges are scheduled with a delay that balances the cost of the new challenge with the estimated energy savings through the current challenge.

We make the simplifying assumption that the energy consumption of the devices per second wall-clock time is approximately constant and roughly the same between the devices. The former holds approximately for the devices considered in our empirical evaluation of Section V, whereas the latter holds approximately as a consequence of the full hardware utilization produced by the speculative execution via Time Warp. Under these assumptions, a relative reduction in energy consumption is equivalent to the same relative reduction in wall-clock time.

In order to determine a new challenge delay d_{i+1}, the heuristic postulates $d_{i+1} r_{\text{WT}} \overset{!}{=} c_{\text{WT}}$, where r_{WT} is the relative reduction in wall-clock time and c_{WT} is the cost in units wall-clock time of a challenge. The latter represents the number of units wall-clock time running on a single device that are equivalent in energy use to a challenge. This equation leads to delays between challenges that correspond to the energy savings by a correct device choice. Adding a tuning parameter $a \in \mathbb{R}_+$, the new challenge delay is calculated as $d_{i+1} = ac_{\text{WT}}/r_{\text{WT}}$

D. Monitor Leader

Here, we rely on the fact that the leader's current performance can be monitored continuously at negligible cost. This allows for a scheduling policy based on the hypothesis that changes in the leader's performance are highly correlated with changes in the relative performance of the devices. If this hypothesis holds, minor changes in the leader's performance indicate that the leader is unlikely to be outperformed by the follower device. Given the leader's performance p_i during the previous challenge and its currently observed performance p', an immediate challenge is triggered if the relative change in performance exceeds a threshold $a \in \mathbb{R}_+$: $|p' - p_i| > ap_i$. A minimum delay d_{min} in wall-clock time between challenges is used as hysteresis to avoid rapidly repeated challenges during periods of highly varying workload. The threshold a and the minimum delay d_{min} are tuning parameters.

IV. Methodology

Our goal is to understand the degree to which the sampling heuristics described in the previous section are able to approximate an optimal placement of sampling points that would lead to Pareto-minimal execution time and energy consumption. To identify the fundamental behavior of the heuristics independently of implementation details and hardware specifics, we evaluate the heuristics in a form of "meta-simulation" that imitates the behavior of the CPU/GPU co-execution over the course of a simulation run. This experiment setup allows us to freely exercise the heuristics under arbitrary synthetic workloads and to optimize the heuristics' parameters in a controlled setting. Furthermore, we can identify near-optimal sampling points based on full knowledge of specific workloads to better understand the optimality gap of the heuristics. In the following, we describe the meta-simulation procedure, the synthetic workload generation, and the optimization approach.

A. Meta-simulation Procedure

The meta-simulation abstractly represents the progress and energy consumption of the CPU/GPU co-execution under a given workload and sampling heuristic. In the meta-simulation, two notions of time must be distinguished: *virtual time* (VT) represents the progress in processing FTL's workload, whereas *wall-clock time* (WT) models the wall-clock execution time. For simplicity, the meta-simulation advances time using a fixed step size, while the implementation of FTL targeting physical hardware follows the discrete-event paradigm over a continuous time base.

The key consideration for the device choice in FTL is the performance of the two devices, which we define here as a ratio:

$$r_{\text{device}}(t_{\text{WT}}) := \frac{t_{\text{VT}}(t_{\text{WT}}) - t_{\text{VT}}(t_{\text{WT}} - \Delta t_{\text{WT}})}{\Delta t_{\text{WT}}}.$$

This ratio, which we will refer to as a device's momentary "speed", is the progress in virtual time over a span Δt_{WT} in wall-clock time. Importantly, the meta-simulation allows us to regard the speed as an independent variable, i.e., we can directly generate speed trajectories for the devices and observe the effects on execution time and energy consumption depending on the chosen sampling heuristic. This is in contrast to our empirical experiments, where the given devices' speeds are dependent variables that can be varied only indirectly by adjusting the workload. The decisive factor for a device's momentary speed are the characteristics of the current workload, i.e., the dynamics of the simulation model at the current point in virtual time. Hence, we represent speed trajectories as sequences of speed values over discretized virtual time.

Algorithm 1 shows pseudo code of the meta-simulation. Its inputs are the termination point in virtual time, a set of device names, their speed trajectories over virtual time, and a function representing the sampling heuristic with associated tuning parameters. The meta-simulation loop advances the wall-clock execution time until the desired virtual time has been reached by iteratively incrementing the current virtual time by the current leader devices' speed, while recording the energy consumption associated with this process. Whenever a challenge is carried out, the energy consumption includes the base overhead of a challenge and the energy required to run all devices in parallel for the current step in wall-clock time. A new leader is chosen as the current fastest device and the progress in virtual time is throttled to account for the challenge overhead in terms of execution time. Finally, the wall-clock time of the next challenge is determined by supplying the configured sampling heuristic with its required arguments (see Section III). The return values of the meta-simulation are the overall wall-clock execution time and energy consumption required to satisfy the termination criterion.

B. Performance Trajectory Generation

The effectiveness of the sampling heuristics can only be evaluated with reference to device speed trajectories corresponding to the device performance under a workload. To emphasize the differences between the heuristics, we focus on speed trajectories that render the device selection particularly challenging. Hence, we aim to generate trajectories that are non-linear and stochastic, and in which the device speeds at a given point in virtual time are uncorrelated. The latter assumption leads to particularly challenging trajectories since on physical hardware, significant changes in workload dynamics are likely to affect all devices' speeds either positively or negatively. An additional requirement for the FTL approach to be applied gainfully is that the optimal leader, i.e., the fastest device, is likely to change over the course of a simulation run.

Algorithm 1 Pseudo code of the meta-simulation used to abstractly evaluate and optimize the sampling heuristics.

```
Require: $T$: end time; $D$: devices; $S$: device speeds over
    time; $E_d$: energy per step for each device; $E_c$: challenge
    energy cost; $C$: challenge speed coefficient; $H$: heuristic
    for next challenge delay; $P$: heuristic parameters
 1: $wt \leftarrow 0$                  # wall-clock time
 2: $wt_{challenge} \leftarrow 0$      # next challenge time
 3: $e \leftarrow 0$                   # energy consumption
 4: $t \leftarrow 0$                   # virtual time
 5: while $t < T$ do
 6:     if $wt_{challenge} = wt$ then
 7:         $e \leftarrow e + E_c + \sum_{d \in D} E_d[d]$
 8:         $leader \leftarrow \arg\max_{d \in D} S[d][t]$
 9:         $t \leftarrow t + S[leader] \cdot C$
10:         $wt_{challenge} \leftarrow wt + H(P)$
11:     else
12:         $e \leftarrow e + E_d[leader]$
13:         $t \leftarrow t + S[leader]$
14:     $wt \leftarrow wt + 1$
15: return $wt, e$
```

Given these desired properties, we generate the speed trajectories based on random walks. Each trajectory is an Ornstein-Uhlenbeck process [61], which is a mean-reverting random walk that allows us to generate uncorrelated trajectories with high variance, without the tendency for early and permanent divergence of simpler, e.g., plain Gaussian, random walks. By bounding the process to $[0.1, 1]$ to generate relative speeds within a factor of 10, setting the mean to 0.5, and varying the process' variance, we obtain speed trajectories as shown in Figure 2.

C. Optimization Approach

We intend to compare the heuristics when adjusted to the general characteristics of a workload, i.e., under near-optimal configurations of their tuning parameters. This is achieved by optimizing the heuristics' parameters over large sets of randomly generated trajectories. Although the heuristics are configured with only one or two parameters, substantial computational demands are created by the large number of trajectories on which the heuristics must be exercised to reliably identify well-performing parameter values.

Given that the meta-simulation returns both the wall-clock execution time and the energy consumption, we are faced with a multi-objective optimization problem. Here, we vary a weight coefficient in order to adjust the importance put on minimizing either execution time or energy. We normalize these values to represent percentage overheads over the minimal time or energy achieved by a hypothetical optimal run that always executes on the faster device without any challenges.

As an additional point of reference in between the oracle result and the sampling heuristics, we approximate the time and energy achievable by an optimal sampling heuristic that

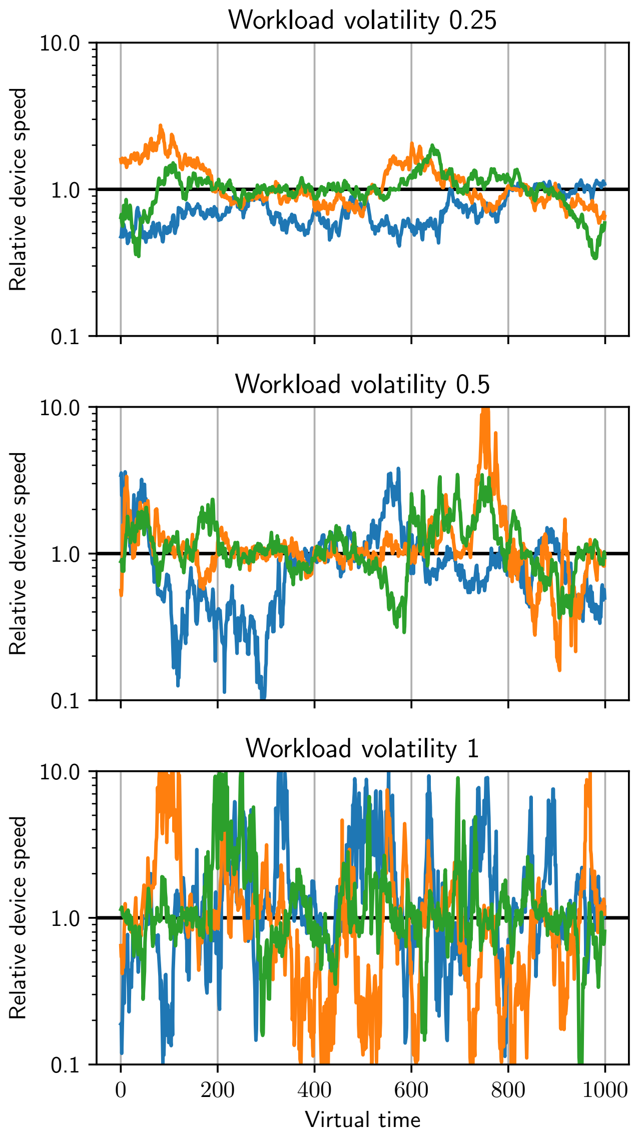

Fig. 2: Example trajectories of the relative device speed over virtual time, varying the workload's volatility by adjusting the standard deviation of the underlying mean-reverting random walks. The plots show three trajectories per volatility level, with the highest level generating frequent and aggressive changes in the relative device speed.

carries out challenges at the optimal points in time. We determine these values by solving a high-dimensional optimization problem for each set of speed trajectories, with one binary decision variable for each wall-clock time step representing the decision whether to carry out a challenge. In our experiments described in Section V, this "brute force" approach leads to an optimization problem with 2 000-dimensional input.

We tackle these optimization problems using DiscoGrad [7], a publicly available[1] framework for gradient estimation across programs with conditional branches. DiscoGrad applies source-to-source transformations and several forms of smoothing in order to efficiently compute gradients in the presence of function discontinuities and stochasticity. By implementing the meta-simulation as a DiscoGrad program in C++, we benefit from the capability of gradient descent to swiftly identify local optima of high-dimensional objective functions.

[1]https://www.github.com/DiscoGrad/DiscoGrad

V. Evaluation

The goal of our evaluation is to assess the execution time and energy consumption achievable via the different heuristics. We first consider the results when executing the heuristics abstractly and with optimized tuning parameters in our meta-simulation before assessing how the results for the best-performing heuristics carry over to physical hardware.

A. Optimized Heuristics

The meta-simulation requires several parameters reflecting assumptions about the modeled Time Warp simulation and hardware platform (cf. Section IV-A). Since we are only interested in relative results between the heuristics and idealized runs, energy and time are represented as unit-less values. For our experiments, we set the energy consumption per unit wall-clock time to 1, the energy consumption of a challenge to 2, and the coefficient slowing down the execution during challenges to 0.9.

For optimizing the heuristics' tuning parameters, we define an objective function $\alpha c_{\text{energy}} + (1-\alpha)c_{\text{WT}}$ that uses an energy coefficient α to weight the percentage overheads in energy and time over an oracle result in which the faster device is chosen without challenge overheads. In order to explore the Pareto frontier of trade-offs between execution time and energy, we optimized the heuristics' tuning parameters for each energy coefficient level in $\{0, 0.03125, 0.0625, 0.125, 0.1875, 0.25, 0.5, 0.75, 1\}$. The workload volatility as configured via the dynamics of the device speeds was set to the levels 0.25, 0.5, and 1 as in Figure 2. The heuristics' parameters were optimized over batches of 1000 random workloads at a time, whereas the "brute force" optimization to identify near-optimal challenge timings was executed on 100 separate fixed workloads for each combination of values for the workload volatility and energy coefficient.

We first assess whether the parameters identified via optimization, which are shown in Table I, align with our expectations (cf. Section III for the parameters' interpretation). In all heuristics, choosing larger parameter values increases the intervals between challenges. We observe that the chosen values consistently follow the expected patterns. With higher energy weight coefficients, longer periods between challenges at the cost of increase execution times can be tolerated. However, even with the weight coefficient set to 1, no extreme increases in in the parameter values can observed, since it is still necessary to take the execution time into account due to the devices' energy consumption over wall-clock time. Conversely, with higher workload volatility, more frequent

TABLE I: Optimized tuning parameters for the different heuristics (cf. Section III). High energy weights generate parameter settings that lead to long intervals between challenges.

Heuristic	Energy weight				
	0	0.125	0.25	0.5	1
Fixed Period	15.1	37.9	55.4	81.9	98.2
AIMD	16.1; 0.2	42.0; 0.9	66.3; 1.0	90.5; 0.4	100.0; 0.9
Energy Budget	0.7	2.7	3.1	8.4	10.0
Monitor Leader	0.1; 7.8	0.1; 18.7	0.2; 27.4	0.2; 39.2	0.2; 54.8

(a) Workload volatility 0.25

Heuristic	Energy weight				
	0	0.125	0.25	0.5	1
Fixed Period	9.4	22.4	31.6	46.8	71.7
AIMD	9.7; 0.0	24.6; 0.9	36.2; 0.8	53.5; 1.0	85.2; 0.1
Energy Budget	0.8	1.3	1.9	3.7	6.5
Monitor Leader	0.1; 4.2	0.2; 10.8	0.2; 11.3	0.3; 19.9	0.3; 28.2

(b) Workload volatility 0.5

Heuristic	Energy weight				
	0	0.125	0.25	0.5	1
Fixed Period	5.7	12.5	17.2	25.0	38.1
AIMD	5.8; 0.1	13.3; 0.5	19.9; 0.8	30.8; 0.9	46.0; 1.0
Energy Budget	0.9	1.9	2.3	2.6	4.2
Monitor Leader	0.2; 0.0	0.3; 0.4	0.3; 0.1	0.4; 9.1	0.5; 15.6

(c) Workload volatility 1

challenges, i.e., lower parameter values, are needed to react to the frequent and abrupt changes in device speeds.

Figure 3 shows the trade-offs between energy consumption and execution time achieved by the different heuristics. There are two points of reference: firstly, the axes show the percentage overheads over hypothetical oracle runs in which the faster device is always chosen without incurring the cost of challenges. Secondly, "Optimized Sampling" represents the results of scheduling challenges based on "brute-force" optimization across known device speed trajectories. Compared to these idealized results, the heuristics incur the cost of challenges and operate without prior knowledge of the device speeds.

At a high level, we see that with higher workload volatility, the difference in results among the sampling policies becomes vastly more pronounced, whereas their ranking remains the same. With workload volatility 1, achieving low energy consumption becomes extremely challenging, as indicated by the overhead of about 10% incurred even with Optimized Sampling. The Energy Budget heuristic is vastly outperformed by all others, and its Pareto frontier does not show the clear pattern visible with the other heuristics. Although Table I showed that its parameter follows the changes in the energy weight coefficient, the effect of this adjustment seems to be weak. Interestingly, the simple Fixed Period heuristic slightly outperforms AIMD, indicating that given our workloads, the time since the last leader change is not a sufficiently strong predictor of the next change for AIMD's rationale to apply. The Monitor Leader heuristic clearly dominates all others. We attribute this to its use of the leader's changes in speed, which directly affects the relative device speed. Encouragingly, the overhead of this heuristic is only a few percentage points above the Optimized Sampling result.

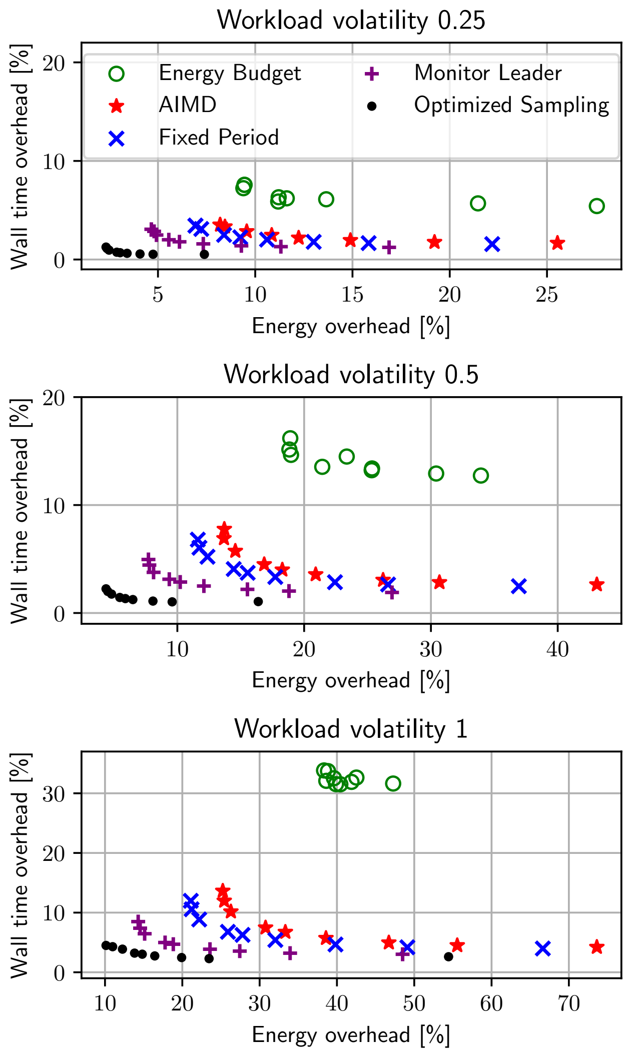

Fig. 3: wall-clock time/energy overheads of the different heuristics in percent over an oracle result without sampling and compared to optimized sampling points based on perfect knowledge.

B. Empirical Results

We now turn to experimental results using an implementation of FTL running on physical hardware based on the CPU-based ROme OpTimistic Simulator (ROOT-Sim) [62] and GPUTW [63]. ROOT-Sim is a CPU implementation of the Time Warp [64] synchronization protocol that supports self-optimized state saving for dynamically allocated memory [18] and load balancing [65]. GPUTW is a GPU implementation of Time Warp [64] and the conservative YAWNS [66] algorithm written in NVIDIA CUDA. Each GPU thread handles events pertaining to a number of simulation objects and holds its own event, state, and antimessage lists. Objects are aggregated dynamically to balance parallelism with the cost of event-list operations.

The benchmark model used in our experiments is a variant

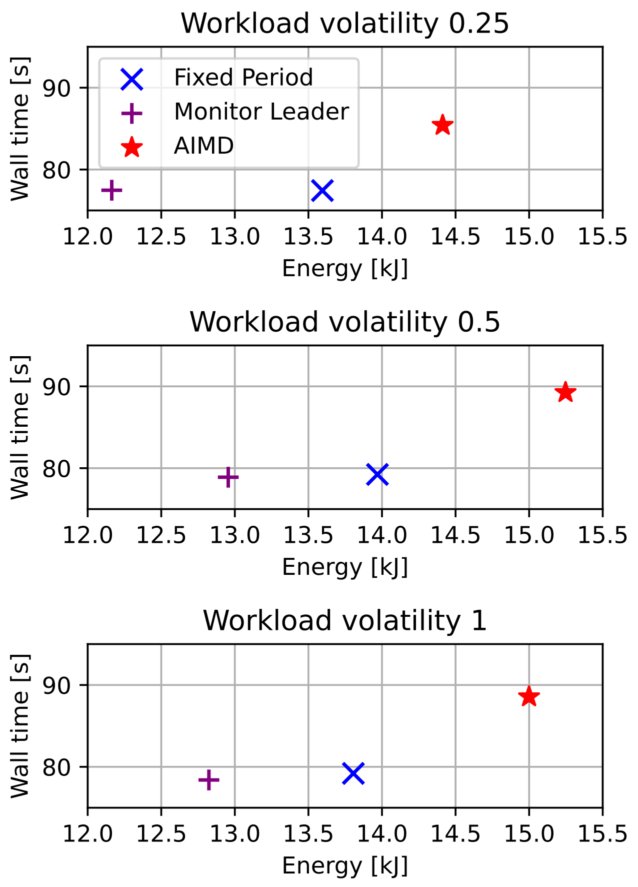

Fig. 4: wall-clock time and energy consumption of the different heuristics for different workload volatility values.

of the traditional PHold model [67]. In PHold, each involves a configurable number of no-operations to imitate processing time after which a new event is scheduled, with the destination object and the delay drawn from probability distributions. In our experiments, the model is extended so that the amount of parallelism in the model can be adjusted dynamically via the percentage of simulation objects that can be targeted by newly-injected events. The difference in the CPU and GPU simulators' performance depending on the degree of parallelism allows us to generate device speed trajectories in line with the random walks described in Section IV-B.

Given that our empirical results showed a strong correlation between energy consumption and execution time, we configured each heuristic with the parameters obtained by the meta-simulations with the weight coefficient set to 0. Since the Energy Budget heuristic was vastly outperformed by the others in the meta-simulations, we exclude it in these experiments.

Our measurements were conducted on a machine equipped with an Intel i7-13700F processor, 32GB RAM, and an NVIDIA RTX 4070 Super, running Debian Ubuntu 24.04. The energy measurements are based on the CPU's and GPU's own estimates.

Figure 4 shows the energy consumption and execution times obtained by running 40 different simulation traces for each volatility value and evaluated heuristic. Our results align with those from the meta-simulations. In particular, AIMD consumes both more energy and time than the other heuristics, which both complete the simulations using 12.2% shorter wall-clock time. As in the meta-simulations, the Monitor Leader heuristic dominates the others, with 15% energy savings over Fixed Period.

VI. Conclusions

The experiments with optimized parameters of the sampling heuristics established a consistent ranking across all levels of volatility in the workload. We observed that scheduling challenges based on changes in the leader device's speed clearly outperformed the other heuristics both in energy consumption and execution time. Remarkably, these results and the parameter choices identified via meta-simulation largely carried over to measurements on physical hardware.

The main limitation of our work lies in the dependence of the results on the chosen workload. Our experiments relied on synthetic workloads generated by mean-reverting random walks. While the results were consistent across instances of these workloads, experimentation across a broader set of representative workloads is needed to assess the robustness of the different heuristics. To enable the direct applicability to practical use cases, we intend to explore dynamic adjustments of the heuristic parameters to given hardware platforms and workloads.

Acknowledgment

This paper has been partially supported by the Italian MUR PRIN 2022 Project: Domain (Grant #2022TSYYKJ) financed by NextGenEu and partially by the Spoke 1 "FutureHPC & BigData" of the Italian Research Center on High Performance Computing, Big Data and Quantum Computing (ICSC) funded by MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di "campioni nazionali" di R&S (M4C2-19) - Next Generation EU (NGEU). Philipp Andelfinger is supported by Deutsche Forschungsgemeinschaft (DFG), German Research Foundation, under Grant 497901036.

References

[1] R. M. Fujimoto, "Parallel discrete event simulation," *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990.

[2] D. Schneider, "The exascale era is upon us: The frontier supercomputer may be the first to reach 1,000,000,000,000,000,000 operations per second," *IEEE Spectrum*, vol. 59, no. 1, pp. 34–35, Jan. 2022.

[3] U. Lopez-Novoa, A. Mendiburu, and J. Miguel-Alonso, "A survey of performance modeling and simulation techniques for accelerator-based computing," *IEEE transactions on parallel and distributed systems: a publication of the IEEE Computer Society*, vol. 26, no. 1, pp. 272–281, Jan. 2015.

[4] R. Marotta, A. Pellegrini, and P. Andelfinger, "Follow the leader: Alternating CPU/GPU computations in PDES," in *Proceedings of the 2024 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS '24. Atlanta, GA, USA: ACM, Jun. 2024.

[5] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, Jul. 1985.

[6] R. Bellman, I. Glicksberg, and O. Gross, "On the "bang-bang" control problem," *Quarterly of applied mathematics*, vol. 14, no. 1, pp. 11–18, 1956.

[7] J. N. Kreikemeyer and P. Andelfinger, "Smoothing Methods for Automatic Differentiation Across Conditional Branches," *IEEE Access*, vol. 11, pp. 143 190–143 211, 2023.

[8] K. S. Perumalla, "Discrete-event execution alternatives on general purpose graphical processing units (GPGPUs)," in *20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)*. IEEE, 2006.

[9] H. Park and P. A. Fishwick, "A GPU-based application framework supporting fast discrete-event simulation," *Simulation*, vol. 86, no. 10, pp. 613–628, Oct. 2010.

[10] X. Liu and P. Andelfinger, "Time warp on the GPU: Design and assessment," in *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS '17. New York, NY, USA: ACM, May 2017, pp. 109–120.

[11] S. Rahman, N. Abu-Ghazaleh, and W. Najjar, "PDES-a: Accelerators for parallel discrete event simulation implemented on FPGAs," *ACM transactions on modeling and computer simulation: a publication of the Association for Computing Machinery*, vol. 29, no. 2, pp. 1–25, Apr. 2019.

[12] J. Xiao, P. Andelfinger, W. Cai, P. Richmond, A. Knoll, and D. Eckhoff, "OpenABLext: An automatic code generation framework for agent-based simulations on CPU-GPU-FPGA heterogeneous platforms," *Concurrency and computation: practice & experience*, vol. 32, no. 21, Nov. 2020.

[13] A. Zhu, Q. Chang, J. Xu, and W. Ge, "A dynamic load balancing algorithm for CFD–DEM simulation with CPU–GPU heterogeneous computing," *Powder technology*, vol. 428, no. 118782, p. 118782, Oct. 2023.

[14] L. R. G. Auriche, F. Quaglia, and B. Ciciani, "Run-time selection of the checkpoint interval in time warp based simulations," *Simulation Practice and Theory*, vol. 6, pp. 461–478, 1998.

[15] Y.-B. Lin, B. R. Preiss, W. M. Loucks, and E. D. Lazowska, "Selecting the checkpoint interval in time warp simulation," *ACM SIGSIM Simulation Digest*, vol. 23, no. 1, pp. 3–10, Jul. 1993.

[16] A. C. Palaniswamy and P. A. Wilsey, "An analytical comparison of periodic checkpointing and incremental state saving," in *Proceedings of the 7th workshop on Parallel and Distributed Simulation*, ser. PADS '93. New York, New York, USA: ACM Press, 1993, pp. 127–134.

[17] H. M. Soliman and A. S. Elmaghraby, "An analytical model for hybrid checkpointing in time warp distributed simulation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 10, pp. 947–951, Oct. 1998.

[18] A. Pellegrini, R. Vitali, and F. Quaglia, "Autonomic state management for optimistic simulation platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, pp. 1560–1569, 2015.

[19] ——, "Di-DyMeLoR: Logging only dirty chunks for efficient management of dynamic memory based optimistic simulation objects," in *Proceedings of the 23rd Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS '09. Piscataway, NJ, USA: IEEE, Jun. 2009, pp. 45–53.

[20] R. Rönngren and R. Ayani, "Adaptive checkpointing in time warp," in *Proceedings of the 8th Workshop on Parallel and Distributed Simulation*, ser. PADS '94. New York, NY, USA: ACM, 1994, pp. 110–117.

[21] S. Sköld and R. Rönngren, "Event sensitive state saving in time warp parallel discrete event simulations," in *Proceedings of the 28th conference on Winter simulation*, ser. WSC '96. New York, New York, USA: ACM Press, 1996, pp. 653–660.

[22] F. Quaglia, "Event history based sparse state saving in time warp," in *Proceedings of the twelfth workshop on Parallel and distributed simulation*, ser. PADS '98. Piscataway, NJ, USA: IEEE Computer Society, 1998, pp. 72–79.

[23] D. West and K. Panesar, "Automatic incremental state saving," in *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*, ser. PADS. Piscataway, NJ, USA: IEEE, May 1996, pp. 78–85.

[24] F. Quaglia, "A cost model for selecting checkpoint positions in time warp parallel simulation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, pp. 346–362, 2001.

[25] F. Montesano, R. Marotta, and F. Quaglia, "Spatial/temporal locality-based load-sharing in speculative discrete event simulation on multi-core machines," in *Proceedings of the 2022 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 81–92. [Online]. Available: https://doi.org/10.1145/3518997.3531026

[26] J. Wang and C. Tropper, "Optimizing time warp simulation with reinforcement learning techniques," in *2007 Winter Simulation Conference*. IEEE, Dec. 2007.

[27] ——, "Selecting GVT interval for time-warp-based distributed simulation using reinforcement learning technique," in *Proceedings of the 2009 Spring Simulation Multiconference*, ser. SpringSim '09. San Diego, CA, USA: Society for Computer Simulation International, Mar. 2009, p. 49.

[28] ——, "Using genetic algorithms to limit the optimism in time warp," in *Winter Simulation Conference*, ser. WSC '09. Winter Simulation Conference, Dec. 2009, pp. 1180–1188.

[29] A. Pellegrini, "Optimizing memory management for optimistic simulation with reinforcement learning," in *2016 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, Jul. 2016.

[30] S. Conoci, M. Ianni, R. Marotta, and A. Pellegrini, "Autonomic power management in speculative simulation runtime environments," in *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS'20. New York, NY, USA: ACM, Jun. 2020, pp. 93–98.

[31] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *ACM SIGARCH Computer Architecture News*, vol. 37, p. 152, 2009.

[32] K. Kothapalli, R. Mukherjee, M. S. Rehman, S. Patidar, P. J. Narayanan, and K. Srinathan, "A performance prediction model for the CUDA GPGPU platform," in *2009 International Conference on High Performance Computing (HiPC)*. IEEE, Dec. 2009.

[33] G. García and C. Yenyxe, "Modelo de estimación de rendimiento para arquitecturas paralelas heterogéneas," Master's thesis, Universitat Politècnica de València, Valencia, Spain, Feb. 2013.

[34] A. Kerr, G. Diamos, and S. Yalamanchili, "Modeling GPU-CPU workloads and systems," *Computer Engineering and Design*, p. 31–42, 2010.

[35] S. Che and K. Skadron, "BenchFriend: Correlating the performance of GPU benchmarks," *The international journal of high performance computing applications*, vol. 28, pp. 238–250, 2013.

[36] K. Sato, K. Komatsu, H. Takizawa, and H. Kobayashi, "A history-based performance prediction model with profile data classification for automatic task allocation in heterogeneous computing systems," in *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications*. IEEE, May 2011.

[37] C. Nugteren and H. Corporaal, "The boat hull model: enabling performance prediction for parallel computing prior to code development," in *Proceedings of the 9th conference on Computing Frontiers*. New York, NY, USA: ACM, May 2012.

[38] J. Meng and K. Skadron, "Performance modeling and automatic ghost zone optimization for iterative stencil loops on GPUs," in *Proceedings of the 23rd international conference on Supercomputing*. New York, NY, USA: ACM, Jun. 2009.

[39] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," *SIGPLAN notices*, vol. 45, no. 5, pp. 115–126, May 2010.

[40] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, p. 65–76, 2009.

[41] J. Lai and A. Seznec, "Break down GPU execution time with an analytical method," in *Proceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*. New York, NY, USA: ACM, Jan. 2012.

[42] Y. Zhang and J. D. Owens, "A quantitative performance analysis model for GPU architectures," in *Proceedings of the 17th International Symposium on High Performance Computer Architecture*, ser. HPCA'11. IEEE, Feb. 2011, pp. 382–393.

[43] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-M. W. Hwu, "An adaptive performance modeling tool for GPU architectures," *SIGPLAN notices*, vol. 45, no. 5, pp. 105–114, May 2010.

[44] Y. Wang and N. Ranganathan, "An instruction-level energy estimation and optimization methodology for GPU," in *Proceedings of the 11th International Conference on Computer and Information Technology*. IEEE, Aug. 2011.

[45] Y. S. Shao and D. Brooks, "Energy characterization and instruction-level energy model of intel's xeon phi processor," in *International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, Sep. 2013.

[46] S. Hong and H. Kim, "An integrated GPU power and performance model," *Computer architecture news*, vol. 38, no. 3, pp. 280–289, Jun. 2010.

[47] R. Ewald, J. Himmelspach, and A. M. Uhrmacher, "An algorithm selection approach for simulation systems," in *2008 22nd Workshop on Principles of Advanced and Distributed Simulation*. IEEE, Jun. 2008.

[48] T. Köster, N. M. Drüeke, and A. Uhrmacher, "Latency optimized execution of sequential simulators by parallel parameter optimization," *Online World Conference on Soft Computing in Industrial Applications*, pp. 4230–4231, Dec. 2018.

[49] M. Castro, R. Rodrigues, and B. Liskov, "BASE: Using abstraction to improve fault tolerance," *ACM Transactions on Computer Systems*, vol. 21, no. 3, pp. 236–269, Aug. 2003.

[50] F. Quaglia, "Software diversity-based active replication as an approach for enhancing the performance of advanced simulation systems," *International Journal of Foundations of Computer Science*, vol. 18, pp. 495–515, 2007.

[51] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: A survey," *ACM Comput. Surv.*, vol. 51, no. 3, pp. 1–33, May 2018.

[52] H. El Tahawy, D. Rodriguez, S. Garcia-Sabiro, and J.-J. Mayol, "VHDeLDO: A new mixed mode simulation," in *Proceedings of EURO-DAC 93 and EURO-VHDL 93- European Design Automation Conference*. IEEE Comput. Soc. Press, 1993, pp. 546–551.

[53] P. Fey, H. W. Carter, and P. A. Wilsey, "Parallel synchronization of continuous time discrete event simulators," in *Proceedings of the 1997 International Conference on Parallel Processing*, ser. ICPP'97. IEEE, 1997, pp. 227–231.

[54] IEEE, "IEEE standard for modeling and simulation (M&S) high level architecture (HLA) federate interface specification," Piscataway, NJ, USA, pp. 1–480, 2008.

[55] M. Hybinette and R. M. Fujimoto, "Cloning parallel simulations," *ACM transactions on modeling and computer simulation: a publication of the Association for Computing Machinery*, vol. 11, no. 4, pp. 378–407, Oct. 2001.

[56] O. Berry and D. Jefferson, "Critical path analysis of distributed simulation," in *Proceedings of the 1985 SCS Multiconference on Distributed Simulation*, ser. PADS'85. San Diego, CA, USA: Society for Modeling and Simulation International, 1985, pp. 57–60.

[57] D. Gianni, G. Iazeolla, and A. D'Ambrogio, "A methodology to predict the performance of distributed simulations," in *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*. IEEE, May 2010.

[58] J. Liu, D. Nicol, B. Premore, and A. Poplawski, "Performance prediction of a parallel simulator," in *Proceedings Thirteenth Workshop on Parallel and Distributed Simulation*, ser. PADS'99. IEEE Comput. Soc, 2003, pp. 156–164.

[59] P. Andelfinger and H. Hartenstein, "Towards performance evaluation of conservative distributed discrete-event network simulations using second-order simulation," in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. New York, NY, USA: ACM, May 2013.

[60] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[61] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the Brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.

[62] A. Pellegrini, R. Vitali, and F. Quaglia, "The ROme OpTimistic simulator: Core internals and programming model," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS. Brussels, Belgium: ICST, Apr. 2012, pp. 96–98.

[63] X. Liu and P. Andelfinger, "Time warp on the gpu: Design and assessment," in *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ser. SIGSIM PADS '17. New York, NY, USA: ACM, 2017, pp. 109–120.

[64] D. Jefferson, B. Beckman, F. Wieland, L. Blume, and M. DiLoreto, "Time warp operating system," in *Proceedings of the eleventh ACM Symposium on Operating systems principles*. New York, NY, USA: ACM, 1987, pp. 77–93.

[65] R. Vitali, A. Pellegrini, and F. Quaglia, "A load-sharing architecture for high performance optimistic simulations on multi-core machines," in *Proceedings of the 19th International Conference on High Performance Computing*, ser. HiPC '12. IEEE, Dec. 2012, pp. 1–10.

[66] D. M. Nicol, "The cost of conservative synchronization in parallel discrete event simulations," *Journal of the ACM (JACM)*, vol. 40, no. 2, pp. 304–333, 1993.

[67] R. M. Fujimoto, "Performance of time warp under synthetic workloads," in *Distributed Simulation*, ser. PADS '90, D. Nicol, Ed. San Diego, CA, USA: Society for Computer Simulation International, 1990, pp. 23–28.