Follow the Leader: Alternating CPU/GPU Computations in PDES

Romolo Marotta
r.marotta@ing.uniroma2.it
Tor Vergata University of Rome
Rome, Italy

Alessandro Pellegrini
a.pellegrini@ing.uniroma2.it
Tor Vergata University of Rome
Rome, Italy

Philipp Andelfinger
philipp.andelfinger@uni-rostock.de
University of Rostock
Rostock, Germany

ABSTRACT

Despite the successes of graphics processing units (GPUs) in accelerating simulations in several research fields, their use is largely restricted to domain-specific workloads that consistently offer the large degree of inherent parallelism and computational intensity at which GPUs excel. When targeting generic discrete-event simulations, whose dynamics can vary wildly over time, a static choice between a GPU-based and traditional CPU-based execution is likely to be suboptimal. Here, we explore a parallel discrete-event (PDES) execution scheme for CPU-GPU platforms that aims to approximate an optimal dynamic device choice. Starting from an intermediate model state, a current "leader" device running the simulation is periodically challenged by a brief concurrent run on another device starting from an intermediate model state. Based on the gathered performance measurements, a forecasting scheme determines the leader for the next period. The execution time and power consumption of this scheme hinge on 1) an efficient mechanism for providing the "follower" device with a consistent model state, and 2) robust performance forecasting to justify the device choices. We present these building blocks, their implementation combining the existing CPU and GPU simulators ROOT-Sim and GPUTW, and measurement results demonstrating substantially reduced execution time without increasing energy consumption over a static device choice.

CCS CONCEPTS

• **Computing methodologies** → **Discrete-event simulation**; **Massively parallel and high-performance simulations**; • **Computer systems organization** → *Heterogeneous (hybrid) systems.*

KEYWORDS

Parallel simulation, Speculative simulation, GPU, Time Warp

ACM Reference Format:
Romolo Marotta, Alessandro Pellegrini, and Philipp Andelfinger. 2024. Follow the Leader: Alternating CPU/GPU Computations in PDES. In *38th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS '24), June 24–26, 2024, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3615979.3656056

1 INTRODUCTION

Largely driven by the highly regular workloads induced by machine learning applications, GPUs enjoy continued popularity as accelerators for data-parallel tasks. Beyond these applications dominated by linear algebra operations, GPUs are commonly used for scientific simulations, particularly when discretizing in regular steps over time or space (e.g., [18]). While manufacturers offer convenient thread-based programming models [22], programs must involve large numbers of parallel operations on dense data structures to make best use of the underlying vectorized operations supported by the hardware, This makes discrete-event simulations (DES) a less obvious fit for GPUs as events can be sparse both among simulation objects and across time. Nevertheless, when using suitable data structures and typically synchronous algorithms, GPU-based DES has been shown to outperform CPUs substantially given large numbers of simulation objects and high event densities in simulation time [15, 23]. Specialized GPU-based simulators have emerged targeting specific model classes such as spiking neural networks [7, 17] that typically exhibit these properties.

Generic DES engines, however, must support arbitrary user-specified models, which may not benefit from GPU acceleration. The device choice becomes further complicated by variability in the model dynamics. Fluctuations in event density affect the relative performance of CPU and GPU simulation, suggesting a dynamic selection at runtime. For this purpose, we present the "follow-the-leader" approach (cf. Figure 1), which approximates an optimal dynamic device selection between a CPU and GPU for generic DES based on measurement-driven time series forecasting and efficient state migration. In addition to execution time reductions compared to a static device choice, the approach has the potential to reduce energy consumption by accelerating the "race-to-idle" [10].

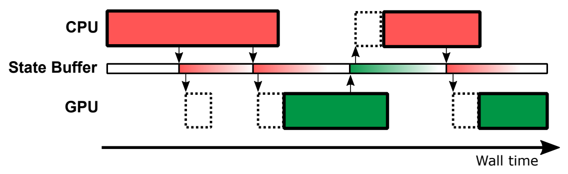


Figure 1: Our approach: The current "leader" device (bold) periodically saves a consistent model state in a buffer (arrows) and is challenged by brief runs of the "follower" device (dotted) starting from the saved state. The devices' predicted relative performance determines the new leader.

2 RELATED WORK

Initial efforts towards tapping the massively parallel hardware of modern GPUs for DES largely focused on offloading specific computationally intensive events, e.g., in wireless network simulations [3], or specialized simulators targeting application domains such as spiking neural networks [7, 17], electronics design automation [5], or agent-based simulations [30]. With increasing general-purpose programmability, generic DES engines entirely executed on a GPU appeared in the literature [1, 2, 15, 23, 26].

Due to the GPU architecture's emphasis on exploiting data parallelism, the relative performance between GPUs and CPUs differs vastly depending on various program characteristics [16], which the time-varying and hard-to-predict dynamics of simulations make particularly difficult to capture. Relying on all available devices [20] or statically employing only one device becomes inefficient when one of the devices is severely outperformed by the other. Our proposed dynamic device selection to attack this issue bears similarities to approaches for simulation algorithm selection [6, 14].

At a glance, a binary device selection can be seen as an extreme case of simulator workload balancing on heterogeneous platforms, which has been considered for certain classes of time-stepped simulations [29, 31]. However, to the best of our knowledge, neither has this problem been explored in the DES context, where assumptions on the model cannot easily be made, nor have the performance and energy implications of a binary device selection been assessed.

In the context of fault tolerance, the work in [4] proposes an opportunistic *n*-version programming approach to enhance Byzantine fault tolerance in software systems. The approach leverages different service implementations to reduce the likelihood of common failure modes. This approach has been applied to PDES in [27], where an Active Replication Management Layer (ARML) within the High-Level Architecture (HLA) framework was proposed. ARML enables transparent execution of multiple active replicas of simulation packages on SMP systems, improving simulation output timeliness without programmer intervention. Our proposal goes a step further by employing implementations targeting different devices to reduce the time-to-completion.

The idea of having multiple simulations of the same model was also explored in "simulation cloning" [12]. This technique dynamically creates multiple simulation instances at decision points within a simulation, enabling concurrent exploration of various execution paths. By sharing common computation results, cloning supports more in-depth explorations of what-if scenarios. Our proposal is different in that we explore a single simulation trajectory, but exploit multiple hardware devices. We also share computation results between the instances, but since we are exploring a single trajectory, we carefully synchronize the state and event queues among the replicas in order to ensure consistent execution.

3 THE APPROACH

The follow-the-leader approach dynamically migrates a simulation's execution between a CPU and a GPU. In the following, we describe the performance forecasting and state migration approach by which we approximate an optimal device selection.

3.1 Device Selection

Evidently, an ideal execution would consistently favor the device that makes faster progress towards the simulation's termination, which we assume to occur at a predefined point in virtual time. To formally capture the speed difference between devices based on the relationship between the elapsed times t_{GVT} and t_{WCT} in virtual and wall-clock time, we first define the ratio

$$r_{\mathrm{device}}(t_{\mathrm{WCT}}) := \frac{t_{\mathrm{GVT}}(t_{\mathrm{WCT}} + \Delta t_{\mathrm{WCT}}) - t_{\mathrm{GVT}}(t_{\mathrm{WCT}})}{\Delta t_{\mathrm{WCT}}}. \quad (1)$$

This is the progress in virtual time over a span Δt_{WCT} in wall time, which will serve as a measure of the momentary simulation speed.

In practice, the interval $[t_{\mathrm{WCT}}, t_{\mathrm{WCT}} + \Delta t_{\mathrm{WCT}}]$ must include at least one GVT calculation in order to observe simulation progress. Assuming that the CPU- and GPU-based simulator are both at the same GVT, we now consider the speed difference $r_{\mathrm{CPU-GPU}}(t_{\mathrm{WCT}}) := r_{\mathrm{CPU}}(t_{\mathrm{WCT}}) - r_{\mathrm{GPU}}(t_{\mathrm{WCT}})$. An ideal execution would select the CPU when the difference is positive, and the GPU when it is negative. However, the difference is not known a priori and must be estimated, for which we rely on time series forecasting based on measurements gathered on the fly. Since the measurements entail non-negligible overhead, we discretize wall-clock time into a series of equidistant epochs of length $\Delta t_{\mathrm{WCT,epoch}}$. At the beginning of each epoch, both devices execute in parallel for a short challenge period $\Delta t_{\mathrm{WCT,challenge}}$ in order to collect a new data point. Based on the time series defined by all previously collected data points, we forecast $r_{\mathrm{CPU-GPU}}$ using Holt's method with damping [9, 11], which combines simple exponential smoothing with a gradually decaying forecast of the current trend: $S_t = \alpha X_t + (1-\alpha)(S_{t-1} + \phi T_{t-1})$; $T_t = \gamma(S_t - S_{t-1}) + (1-\gamma)\phi T_{t-1}$; $\hat{X}_t(m) = S_t + \sum_{i=1}^{m} \phi^m T_t$, where S_t are the data points in the time series, T_t is the trend, α and γ are smoothing parameters in $[0, 1]$, and ϕ, which is set to 0.9 in our experiments, governs the trend's damping over time. This forecasting model is equivalent to an ARIMA (1, 1, 2) process [9].

Once the challenge at the start of an epoch has completed, we fit a model of the above form to the previously observed data points of $r_{\mathrm{CPU-GPU}}$ by adjusting α and γ using Nelder and Mead's method [19]. The model is then exercised to obtain a forecast $\hat{r}_{\mathrm{CPU-GPU}}$ of the speed difference up to the end of the epoch of length $\Delta t_{\mathrm{WCT,window}}$. The sign of the forecast's value decides which device is active for the next epoch. Assuming normally distributed forecasting errors, we use the prediction error's variance over the existing data points to additionally estimate the probability that the chosen device will in fact be faster. Given the error's variance σ^2_{err}, the estimated probability of $r_{\mathrm{CPU-GPU}} \leq 0$ is determined via the cumulative density function of the normal distribution as $\Phi(\frac{r_{\mathrm{CPU-GPU}}}{\sigma err})$. When $\mathbb{P}(r_{\mathrm{CPU-GPU}} \leq 0)$ is consistently close to $\frac{1}{2}$, i.e., the forecasts are too noisy to support a reliable device selection, the challenge overhead can be avoided by falling back to a traditional execution on a single device.

Naturally, our approach can only approximate the execution time and energy consumption of an optimal device selection. The sources of deviations from an optimal execution are as follows: 1) Discretization error: When $r_{\mathrm{CPU-GPU}}$ changes its sign throughout an epoch, the device selection is suboptimal for parts of the epoch. 2) Misprediction: With a certain probability, the forecast suggests the slower device to be selected. 3) Challenge overhead: Prior to each challenge, a consistent snapshot of the current leader simulator's state and events must be gathered, requiring a GVT calculation that may otherwise be postponed. 4) An additional overhead in terms of energy consumption is given by the period of length $\Delta t_{\mathrm{WCT,challenge}}$ during which both devices are active.

Key tradeoffs exist in configuring $\Delta t_{\mathrm{WCT,epoch}}$ and $\Delta t_{\mathrm{WCT,challenge}}$. A longer epoch duration increases the discretization error as well as the length of the forecasting horizon and thus the probability for mispredictions, yet decreases the frequency of incurring the overhead for the challenges. A longer challenge duration may provide more accurate data points to be used in the forecasting but increases the durations in wall-clock time during which both devices are active. Note that the overhead incurred by an increased challenge duration is in energy consumption only.

3.2 State Migration

An essential feature required by our approach is to resume the simulation on the "follower" device whenever a new challenge phase must be carried out. In fact, we cannot simply restart the simulation on the follower because its state lags behind the leader device's in simulation time. Consequently, we need to realign the state of the simulation across the different devices.

Since GPU/CPU DRAM transfers prefer aligned and large buffers, we introduced the *state buffer* (see Figure 1), which is a contiguous memory region in CPU DRAM where each simulator reads/writes from/to a simulation snapshot. The latter includes states and pending events of each simulation object.

Since the different simulators might implement different kinds of PDES engines (e.g., conservative vs. speculative), our proof-of-concept implementation imposes that a simulation snapshot is guaranteed to belong to the correct simulation trajectory. On the one hand, this allows the coexistence of conservative and speculative engines. On the other hand, if a speculative DES engine is running on the leader device, it needs to switch from forward execution to snapshot collection. The latter selects a consistent state and pending events for each simulation object at the same committed virtual time, which is achieved by aborting any speculative computation.

Once a committed snapshot has been constructed, the leader serializes it into the state buffer, and the follower engine deserializes the snapshot to resume the computation.

4 EXPERIMENTS

4.1 Setup

The reference CPU and GPU implementations that we have adopted and integrated to evaluate our proposal are The ROme OpTimistic Simulator (ROOT-Sim) [24] and GPUTW [15].

ROOT-Sim is an optimized implementation of a PDES runtime environment based on the Time Warp [13] synchronization protocol, supporting self-optimized state saving for dynamically allocated memory [25] and load balancing [28]. It is designed to implement simulation models through event handlers based on standard ANSI-C, providing a simple and reduced API for ease of use. ROOT-Sim is built on a set of paradigms focused on performance and scalability, offering facilities for parallelizing execution transparently.

GPUTW is a GPU implementation of the Time Warp [13] and YAWNS [21] algorithms written in NVIDIA CUDA. The implementation follows a synchronous approach in which each GPU thread is responsible for events pertaining to a dynamic number of model entities and holds its own event, state, and antimessage lists. The entity aggregation level is automatically tuned to obtain sufficiently dense parallelism while limiting the cost of event-list operations.

As a benchmark, we have used a variation of the traditional PHold model [8]. In PHold, simulation objects mimic real-world models by means of busy loops, which eat processing cycles with no-operations for a specified amount of time. After each event's execution, a new event is scheduled to some other destination object. In our experiments, we used two different execution phases. In a *balanced* phase, each object targets any other object in the model with a uniform probability. Conversely, in an *unbalanced* phase, only a small subset of the simulation objects are targeted by newly-injected events. This imbalance creates adversarial dynamics for performance of Time Warp simulations, due to increased likelihood of rollbacks and reduces parallelism. Since the CPU and GPU simulators fare differently with balanced and unbalanced workloads, alternating between phases allows us to showcase the follow-leader-leader approach's ability to dynamically switch devices.

Our measurements were conducted on a machine equipped with a AMD Ryzen 9 7950x processor, 64GB RAM, and an NVIDIA RTX 3090 Ti, running Debian GNU/Linux 11. Energy measurements are based on the CPU's and GPU's self-reported energy estimates.

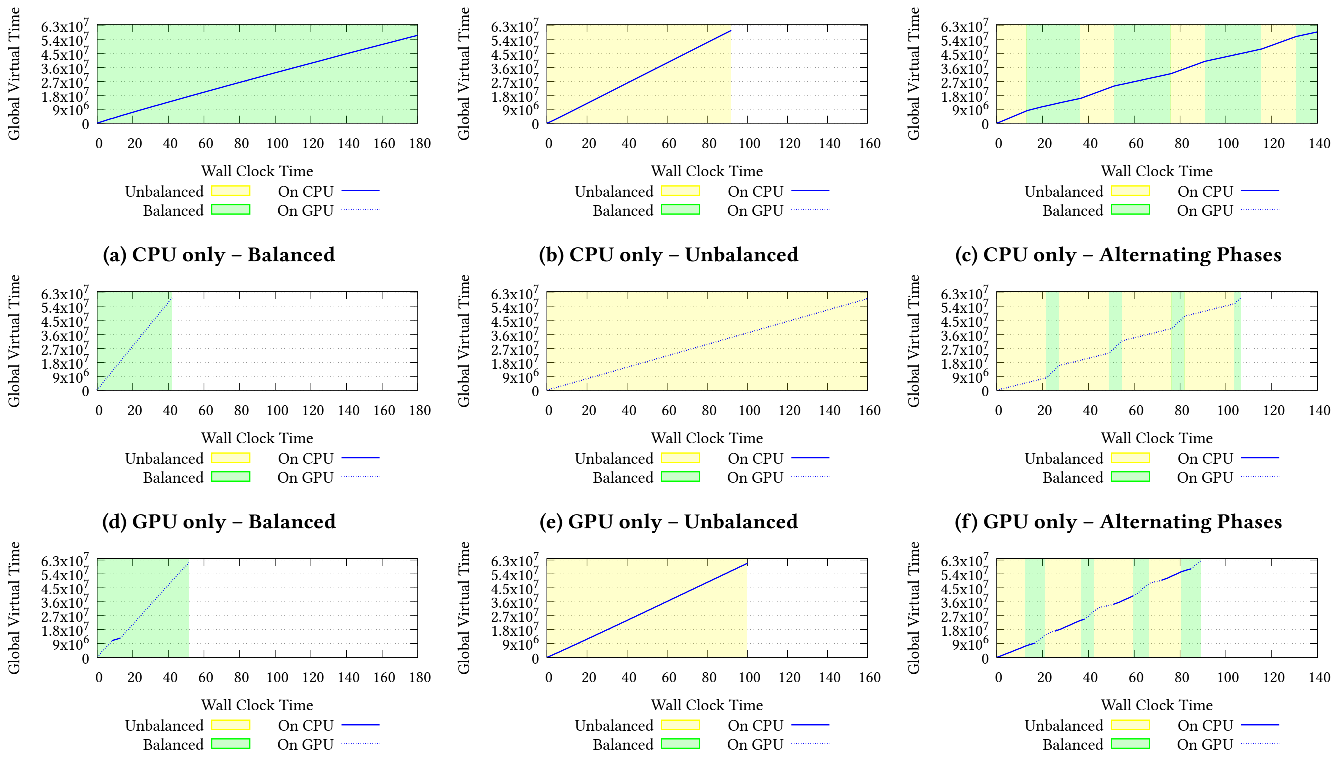
4.2 Results

Figure 2 shows the simulation progress in GVT up to the end time of 6.4×10^7 over wall clock time. The background color indicates the current workload balance generated by the model, and the line types indicate the device the simulation is running on.

Considering a purely "balanced" model configuration (left-hand side), we observe that the GPU vastly outperforms the CPU, finishing the simulation in about 43s wall clock time, compared to 195s on the CPU. The "unbalanced" case yields the opposite trend, with 163s execution time on the GPU and 95s on the CPU. In both cases, it is optimal to use just one of the devices without dynamic switching, i.e., the follow-the-leader approach can only produce overhead. In line with this expectation, a moderate overhead is observed. Most of the overhead is generated by the rollbacks and migration steps induced by the challenges (cf. Section 3.2). In the balanced run, the effect of an erroneous forecast is visible, likely caused by measurement noise. As a result, the CPU briefly becomes the leader although the GPU would be faster.

Finally, in the plots on the right-hand side, the model alternates between phases of balanced and unbalanced workload. Here, the benefits of the follow-the-leader approach are evident: The execution successfully switches between the devices in accordance with the model's phases, reducing the execution time from 108s or 147s on the individual devices to only 92s.

Figure 3 summarizes the energy in Joules required for the three model configurations. The trend for the balanced and unbalanced cases largely follows that of the performance curves, the least energy being consumed on the respective faster device. Note that we include the energy consumption of the passive device: When the GPU is active, one CPU thread orchestrates the GPU's execution via CUDA API calls while when the CPU is active, the GPU still consumes a non-negligible amount of energy. For the alternating model, the lowest energy consumption is achieved when running on the CPU, while the GPU and follow the leader consumed similar amounts of energy. In other words, our approach produced results faster without adding significantly to the energy consumption.

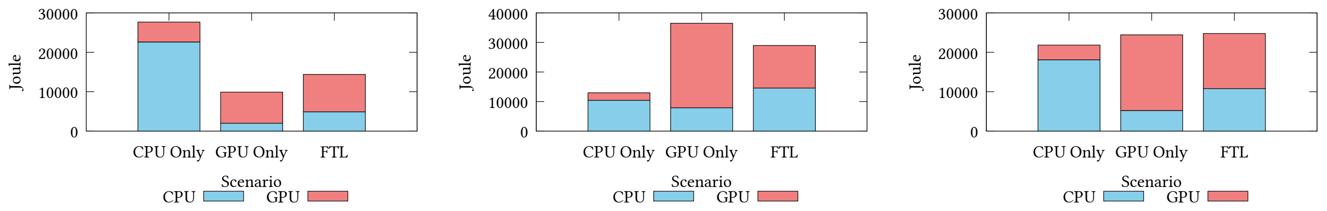


(a) CPU only – Balanced (b) CPU only – Unbalanced (c) CPU only – Alternating Phases

(d) GPU only – Balanced (e) GPU only – Unbalanced (f) GPU only – Alternating Phases

(g) Follow the leader – Balanced (h) Follow the leader – Unbalanced (i) Follow the leader – Alternating Phases

Figure 2: Simulation progress in terms of GVT over wall time on the CPU, GPU, and our combined approach. The benchmark model alternates between phases of balanced and unbalanced workload favoring either the CPU or the GPU.



(a) Balanced. (b) Unbalanced. (c) Alternating Phases.

Figure 3: Comparison of the energy consumption when running on a single device or when switching devices dynamically.

5 CONCLUSIONS

Our approach can reduce the time needed for simulations with fluctuating computational demands by about 20% over CPU/GPU, while maintaining GPU-level energy use. With worst-case time-invariant workloads, execution times increased by about 10–25%.

If changes in model dynamics are rare, the periodic challenges generate unnecessary overhead that more elaborate schemes could reduce. For instance, challenges could be scheduled when a change in the devices' relative speed is predicted, or the forecast's fidelity is expected to drop below a threshold.

Technically, the key building block is an efficient facility to transfer events between CPU and GPU. This will allow future work to explore more deeply intertwined modes of co-execution between otherwise self-contained and generic PDES realizations.

ACKNOWLEDGMENTS

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).

REFERENCES

[1] Philipp Andelfinger and Hannes Hartenstein. 2014. Exploiting the parallelism of large-scale application-layer networks by adaptive GPU-based simulation. In *Proceedings of the Winter Simulation Conference 2014*. IEEE, Piscataway, NJ, USA, 3471–3482.

[2] Philipp Andelfinger, Jens Mittag, and Hannes Hartenstein. 2011. GPU-based architectures and their benefit for accurate and efficient wireless network simulations. In *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE, Piscataway, NJ, USA, 421–424.

[3] Scott Bai and David M Nicol. 2008. Gpu coprocessing for wireless network simulation. In *Symposium on Application Accelerators in High Performance Computing (SAAHPC'08)*. IEEE Computer Society, Washington, DC, USA, 3 pages.

[4] Miguel Castro, Rodrigo Rodrigues, and Barbara Liskov. 2003. BASE: Using abstraction to improve fault tolerance. *ACM transactions on computer systems* 21, 3 (Aug. 2003), 236–269. https://doi.org/10.1145/859716.859718

[5] John F Croix and Sunil P Khatri. 2009. Introduction to GPU programming for EDA. In *Proceedings of the 2009 International Conference on Computer-Aided Design (ICCAD '09)*. ACM, New York, NY, USA, 276–280.

[6] Roland Ewald, Jan Himmelspach, and Adelinde M Uhrmacher. 2008. An algorithm selection approach for simulation systems. In *2008 22nd Workshop on Principles of Advanced and Distributed Simulation*. IEEE, Piscataway, NJ, USA, 91–98.

[7] Andreas K Fidjeland, Etienne B Roesch, Murray P Shanahan, and Wayne Luk. 2009. NeMo: a platform for neural modelling of spiking neurons using GPUs. In *2009 20th IEEE international conference on application-specific systems, architectures and processors*. IEEE, Piscataway, NJ, USA, 137–144.

[8] Richard M Fujimoto. 1990. Performance of Time Warp Under Synthetic Workloads. In *Distributed Simulation (PADS '90)*, David Nicol (Ed.). Society for Computer Simulation International, San Diego, CA, USA, 23–28.

[9] Everette S Gardner Jr and ED McKenzie. 1985. Forecasting trends in time series. *Management science* 31, 10 (1985), 1237–1246.

[10] Matthew Garrett. 2007. Powering Down: Smart power management is all about doing more with the resources we have. *Queue* 5, 7 (2007), 16–21.

[11] Charles C. Holt. 2004. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting* 20, 1 (2004), 5–10. https://doi.org/10.1016/j.ijforecast.2003.09.015

[12] Maria Hybinette and Richard M Fujimoto. 2001. Cloning parallel simulations. *ACM transactions on modeling and computer simulation: a publication of the Association for Computing Machinery* 11, 4 (Oct. 2001), 378–407. https://doi.org/10.1145/508366.508370

[13] David Jefferson, Brian Beckman, Frederick Wieland, Leo Blume, and Mike DiLoreto. 1987. Time warp operating system. In *Proceedings of the eleventh ACM Symposium on Operating systems principles*. ACM, New York, NY, USA, 77–93.

[14] Till Köster, Nicola M Drüeke, and Adelinde M Uhrmacher. 2018. Latency optimized execution of sequential simulators by parallel parameter optimization. In *Proceedings of the 2018 Winter Simulation Conference*. IEEE, Piscataway, NJ, USA, 4230–4231.

[15] Xinhu Liu and Philipp Andelfinger. 2017. Time Warp on the GPU: Design and assessment. In *Proceedings of the 2017 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM PADS '17)*. ACM, New York, NY, USA, 109–120. https://doi.org/10.1145/3064911.3064912

[16] Unai Lopez-Novoa, Alexander Mendiburu, and Jose Miguel-Alonso. 2014. A survey of performance modeling and simulation techniques for accelerator-based computing. *IEEE Transactions on Parallel and Distributed Systems* 26, 1 (2014), 272–281.

[17] Jayram Moorkanikara Nageswaran, Nikil Dutt, Jeffrey L Krichmar, Alex Nicolau, and Alex Veidenbaum. 2009. Efficient simulation of large-scale spiking neural networks using CUDA graphics processors. In *2009 International Joint Conference on Neural Networks*. IEEE, Piscataway, NJ, USA, 2145–2152.

[18] Cristobal A Navarro, Nancy Hitschfeld-Kahler, and Luis Mateu. 2014. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. *Communications in Computational Physics* 15, 2 (2014), 285–329.

[19] John A Nelder and Roger Mead. 1965. A simplex method for function minimization. *The computer journal* 7, 4 (1965), 308–313.

[20] Quang Anh Pham Nguyen, Philipp Andelfinger, Wen Jun Tan, Wentong Cai, and Alois Knoll. 2021. Transitioning spiking neural network simulators to heterogeneous hardware. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 31, 2 (2021), 1–26.

[21] David M Nicol. 1993. The cost of conservative synchronization in parallel discrete event simulations. *Journal of the ACM (JACM)* 40, 2 (1993), 304–333.

[22] NVIDIA. 2024. CUDA, release: 10.3. https://developer.nvidia.com/cuda-toolkit

[23] Hyungwook Park and Paul A Fishwick. 2010. A GPU-based application framework supporting fast discrete-event simulation. *Simulation* 86, 10 (2010), 613–628.

[24] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia. 2012. The ROme OpTimistic Simulator: Core Internals and Programming Model. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTOOLS)*. ICST, Brussels, Belgium, 96–98. https://doi.org/10.4108/icst.simutools.2011.245551

[25] Alessandro Pellegrini, Roberto Vitali, and Francesco Quaglia. 2015. Autonomic State Management for Optimistic Simulation Platforms. *IEEE Transactions on Parallel and Distributed Systems* 26 (2015), 1560–1569. https://doi.org/10.1109/TPDS.2014.2323967

[26] Kalyan S Perumalla. 2006. Discrete-event execution alternatives on general purpose graphical processing units (GPGPUs). In *20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06)*. IEEE, Piscataway, NJ, USA, 74–81.

[27] Francesco Quaglia. 2007. Software Diversity-Based Active Replication as an Approach for Enhancing the Performance of Advanced Simulation Systems. *Int. J. Found. Comput. Sci.* 18, 3 (2007), 495–515. https://doi.org/10.1142/S0129054107004802

[28] Roberto Vitali, Alessandro Pellegrini, and Francesco Quaglia. 2012. A load-sharing architecture for high performance optimistic simulations on multi-core machines. In *Proceedings of the 19th International Conference on High Performance Computing (HiPC '12)*. IEEE, Piscataway, NJ, USA, 1–10. https://doi.org/10.1109/hipc.2012.6507510

[29] Jiajian Xiao, Philipp Andelfinger, Wentong Cai, Paul Richmond, Alois Knoll, and David Eckhoff. 2020. OpenABLext: An automatic code generation framework for agent-based simulations on CPU-GPU-FPGA heterogeneous platforms. *Concurrency and Computation: Practice and Experience* 32, 21 (2020), e5807.

[30] Jiajian Xiao, Philipp Andelfinger, David Eckhoff, Wentong Cai, and Alois Knoll. 2019. A survey on agent-based simulation using hardware accelerators. *ACM Computing Surveys (CSUR)* 51, 6 (2019), 1–35.

[31] Aiqi Zhu, Qi Chang, Ji Xu, and Wei Ge. 2023. A dynamic load balancing algorithm for CFD–DEM simulation with CPU–GPU heterogeneous computing. *Powder Technology* 428 (2023), 118782.