applied sciences

MDPI

Article

Accelerated DEVS Simulation Using Collaborative Computation on Multi-Cores and GPUs for Fire-Spreading IoT Sensing Applications

Seongseop Kim, Jeonghun Cho and Daejin Park *

School of Electronics Engineering, Kyungpook National University, Daegu 41566, Korea; kss92318@gmail.com (S.K.); jcho.knu@gmail.com (J.C.)

* Correspondence: boltanut@knu.ac.kr; Tel.: +82-053-950-5548

Received: 23 July 2018; Accepted: 24 August 2018; Published: 26 August 2018

Abstract: Discrete event system specification (DEVS) has been widely used in event-driven simulations for sensor-driven Internet of things (IoT) applications, such as monitoring the spread of fire disaster. Event-driven models for IoT sensor nodes and their communication is described in DEVS and they have to be integrated with continuous models of fire-spreading dynamics so that the hybrid system modeling and simulation approach have to be considered for both continuous behavior of fire-spreading and event-driven communications by large-scale IoT sensor devices. The hybrid-integrated modelling and simulation for fire-spreading in wide area and large-scale IoT devices result in more complex model evaluation, including simulation time synchronization, so that simulation acceleration is important by considering scalability in large-scale IoT-driven applications that sense fire-spreading. In this study, we proposed a scalable simulation acceleration of a DEVS-based hybrid system using heterogeneous architecture based on multi-cores and graphic processing units (GPUs). We evaluated the power consumption comparison of the proposed accelerated-simulation approach in terms of the composition of the event-driven IoT models and continuous fire-spreading models, which are tightly described in differential equations across a large number of cellular models. The demonstrated result shows that the full utilization of CPU-GPU integrated computing resources, on which event-driven models and continuous models are efficiently deployed and optimally distributed, could enable an advantage for high-performance simulation speedup in terms of execution time, although more power consumption is required, but the total energy consumption could be reduced due to fast simulation time.

Keywords: discrete event simulation; hybrid simulation; GPU-based accelerated simulation; simulation kernel distribution on multi-core architecture

1. Introduction

Simulation has been widely used to solve various problems in dynamic systems where the characteristics of a system are not statically described in analytic equations. discrete event system specification (DEVS)-based modeling and simulation (M&S) [1] has been widely adopted as an efficient simulation framework by splitting the model description part and simulation-related tasks. An M&S approach for event models, which are described with the state transitions by discrete events, has advantages when it comes to evaluating simulation models in terms of the bandwidth and speed compared with a simulation based on a discrete time-step. DEVS-based M&S has been applied and studied [2–4] to accelerate the simulation speed with a reasonable fidelity for a hybrid system with complex behaviors, which are simultaneously described in a discrete event model and continuous time model. The hybrid system has continuous and discrete dynamic behavior, and it is characterized by interactions between the continuous system and the discrete event system. For this reason, DEVS-based

hybrid system simulation has been widely used for expressing [3] and simulating complex systems and has been studied in various fields [5,6]. In most cases, however, these hybrid systems are more complex and require more time than discrete event-based simulations because the continuous system has to be scheduled frequently in the manner of a small discrete time step in a simulation kernel.

DEVS-based simulation studies using cellular models have been introduced as an extension of DEVS M&S formalism [7]. The cellular model is defined as a grid of cells, for which the discrete events describe the relation between cells and the internal state transitions are based on continuous time models such as differential equations. DEVS-based cellular models can be applied in hybrid systems with differential-equation-based state transitions and event-driven propagation models such as fire-spreading. Cellular models consist of a large number of cells, which are represented with the detailed model description in continuous time domain, and event-driven model evaluation of cells in hybrid systems have big overhead in communicating with the cross-coupled events. To mitigate the drawback caused by this style of hybrid description of cellular models, aggressive enhancement in a simulator's kernel level has to be considered to accelerate simulation speeds. The parallel simulation studies have been presented [8], parallelizing DEVS-based simulation in multiple cores [9]. In addition, parallelization approaches using graphic processing units (GPUs), which have an advantage in performing large amounts of computation, have been introduced [10]. GPUs are used in many research fields because they make it easy to convert iterative operations for performing tasks in the parallel way. GPU-based simulation enables integrated acceleration for continuous and discrete-event models by executing control flows in sequential algorithms on the small number of CPUs executing and large number of parallelized tasks on the GPU.

In this study, we describe the accelerated process of simulation and DEVS-based hybrid systems, which possess discrete system and continuous system behaviors. We use large-scale cellular model simulations, which can be applied not only to a homogeneous system but also to the modeling of an Internet of Things (IoT) system, such as a smart farm and a smart factory. We propose the distribution of the simulation process to the multi-cores and GPUs of a large-scale cellular model following hybrid systems. In addition, we propose parallelizing a simulation using a GPU. We simulate in the proposed heterogeneous architecture, which involves the coupling of multi-cores and GPU-based systems and analyze the result by changing the simulation parameters. The simulation time is also analyzed by composing several configurations of target devices according to the number of devices. In addition, we analyze each maximum power consumption via a configuration of target devices to find an efficient operation of the proposed simulation.

This paper is organized as follows. Section 2 describes previous works related to this study, and Section 3 summarizes the basic background of DEVS. In Section 4, we present our approach to tightly integrate CPU-GPU architecture with cellular models on a simulator kernel using the multi-cores and GPUs proposed in this study, and Section 5 shows experimental results of the simulation acceleration for the fire-spreading model, which is adopted as a case study. We discuss our proposed approach in Section 6 in terms of the simulation time, the power consumption, and the configuration of CPUs and GPUs. Section 7 provides the conclusions of our study.

2. Related Works

The simulation of complex systems using a hybrid system based on DEVS has been used and studied in various fields [5,6,11]. In addition, many studies have been done on parallelism studied [12–14], also using partitioning models to sub-models [9]. Various approaches in parallel simulation for DEVS-based models have been proposed to resolve the synchronization issues during simultaneous evaluation of multiple models. A parallel simulation method using null-messages and lookahead-based synchronization mechanism has been introduced [15]. A method to reduce the inter-process communication overhead in parallel simulations has been introduced by flattening the hierarchical model structure [16]. In parallelizing the simulation of DEVS-based hybrid system models, the simulation time could be reduced by delaying the synchronization between continuous

models and discrete-event models within allowable time [17]. These previous approaches are based on enhancing the simulator architecture on general CPU architecture. Our approach starts from the detailed understanding of the given hybrid system models. By exploiting the CPU-GPU and dedicated memory, the continuous models and discrete-event models are efficiently partitioned into CPU-GPU architecture and the runtime simulation data is allocated on the GPU memory to reduce the copy overhead from simulation kernel on CPU.

Some previously conducted studies used another architecture as a co-processor to parallelize DEVS. Studies have been carried out to speed up the DEVS-based simulation process using GPUs as co-processors [10,18]. As part of DEVS simulation, the cellular model has been studied based on DEVS with its homogeneous features [19,20], especially spreading phenomena such as fire-spreading [21,22]. In cellular model simulation, generally, the greater the number of cells, the more detailed simulation results can be obtained, but more computation is required. To solve this problem, a large-scale cellular model was studied in terms of high-performance [23].

The discrete-event model simulation acceleration using GPU has been introduced [18]. This study showed a GPU utilization method to mitigate the inefficient simulation evaluation caused by the irregular behavior in the time advance of simulation. For parallel simulation of cellular models, the simulation evaluation mainly executed on CPU and GPU is used as an accelerator [10]. In contrast to the existing GPU-based DEVS parallelization studies, our approach is to accelerate the operation of a DEVS-based hybrid system and to verify the high-performance of the proposed architecture by using large-scale cellular models, which can generate detailed simulation results. In addition, we propose simulation acceleration and enhancement methods by mitigating the bottleneck caused by parallel access to GPU memory, which is driven by multiple CPUs executing the parallel simulations. Our proposed study focuses on exploring efficient utilization of CPU-GPU and collaboration of heterogeneous models in extending hybrid simulation using DEVS-driven simulator. We found that memory access pattern during simulation is caused by the structure of hybrid models, which is one of the major causes of slow simulation, even though powerful GPU has been adopted. Simulation of continuous models on GPU requires additional time cost in the data copy from system memory, and multiple CPUs simultaneously access GPU causing a bottleneck effect in accessing GPU memory. Our approach tries to partition structure of hybrid models into simulation models on multiple CPU and efficiently allocate runtime simulation data on GPU memory and try to reduce the bottleneck of simultaneous accesses to GPU, which is powered by an multi process service (MPS) feature of compute unified device architecture (CUDA).

In this paper, we aim to implement a tightly coupled simulation architecture for DEVS-based event-driven IoT applications. We propose an accelerated CPU-GPU collaborative computation, especially for the simulation of a hybrid system that is a combined discrete event system, which can be represented as an event-driven model, and a continuous time system. We propose a partitioning simulation process for each device. In addition, we analyze the simulation process based on several hardware configurations according to the number of CPUs and GPUs, and we analyze the power consumption to measure efficiency.

3. Background

3.1. Discrete Event System

A discrete event system (DES) is a time event system, in which each discrete state is updated based on asynchronous events that occur over time. In a discrete event system, the state of the system is changed dynamically while the system is being simulated. Various modeling methods exist for a discrete event system. Among them, DEVS is used for modular, layered formalism, including a discrete event system based on state transition, a continuous state system represented by differential equations, and a hybrid system in which both exist. DEVS is also used for object-oriented modeling and analysis. A real system modeled using DEVS can be depicted as a set of atomic and coupled models.

Figure 1 displays a DEVS atomic model of DES, and the following specifications represent its formal representation. An atomic model of DEVS is basically represented by three variables and four functions. The three variables include X, Y, and S, and the four functions represent an external state transition function, an internal state transition function, an output function, and a time advance function. X is the set of input events, and Y is the set of output events. The S is the set of sequential states, which cover all states of the modeling system. σ_{Ext} is the external state transition function with system state S, input X, and e is the elapsed time since the last state transition. σ_{Int} is the internal state transition function. λ is the output function of the system. ta is the time advance function, which can take any real value between 0 and inf. The time advance function is used as an operation trigger for the internal state transition function.

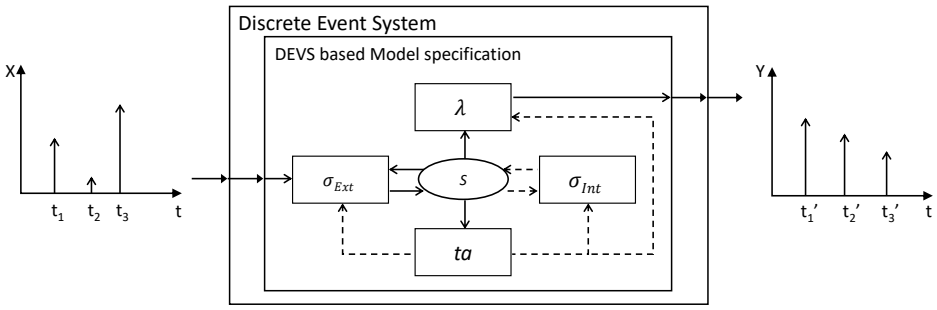


Figure 1. Specification of discrete event system.

The state variables with discrete values in DEVS are updated when the model receives input from outside of the model. In addition, if the input is not received for a certain period of time, the internal state transition function is operated after waiting for a predetermined time, and the state variable is updated. The amount of time necessary to wait for an input is determined by the time advance function. With this operation, unlike other system modeling methods, the DEVS modeling system has a different structure in which the state of the system is changed not only by external input but also by an internal transition.

3.2. Continuous System

We can define a continuous system (CS) as follows. The input affects the state of the system. The state is affected not only by the input at the present time, but also by its past inputs. Thus, differential equations follow. In a simulation of the continuous system, simulation time is divided into many small steps, which can represent a discrete time system. The system model state affected by differential equations is updated over time. With this characteristic of discrete time simulation, the discrete time system model is applicable to all continuous system simulations.

3.3. Hybrid System

A hybrid system is a representation that can be applied to a system combining discrete event and continuous system modeling. The continuous system modeled by a discrete time system is defined by a set of differential equations ϕ and a continuous system with continuous state Q_{CS}, whose continuous behavior can be modeled by differential equation ϕ. However, the system receives input stimulations that are piecewise constant time functions. Thus, the need to convert piecewise value must exist, which can involve changing a represented event to a continuous state. To meet this need, the system is also modeled with a threshold sensor for converting an event to a continuous value. The state value in continuous-time model can be converted with various methods into event states in a discrete-event model. In this paper, we adopted the threshold-based event conversion method, for which the threshold sensor monitors the changes of its state only at particular points f_i, the threshold value, which is defined in the modeling process, is used to determine the event transition compared to the change of state

value in the continuous model within the given maximum duration of current state. The specification of the hybrid system is described in Figure 2.

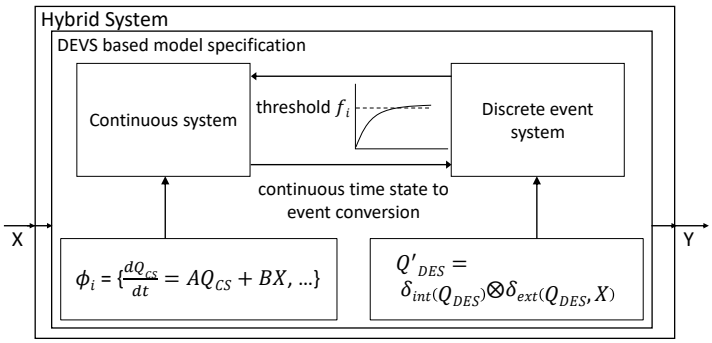


Figure 2. Specification of a hybrid system.

3.4. CUDA Multi-Process Service

CUDA (compute unified device architecture) is a general purpose parallel computing platform and programming model that has advantages in solving many complex computational problems. The computation of GPU typically involves copying data over to previously allocated regions in GPU memory, running a GPU kernel which uses copied data, and copying them back to host CPU. Generally, a process of GPU is assigned and serially scheduled by GPU scheduler. Therefore, GPU operations are inefficient if a process scheduled at a particular time slice is in underutilization of GPU resources. For GPU utilization, CUDA supports Multi-Process Service (MPS). The CUDA MPS is one of the CUDA application programming interface (API), which makes CUDA kernels be processed concurrently on the same GPU. CUDA MPS has benefits at GPU utilization, reducing on-GPU context storage, reducing GPU context switching and identifying candidate applications. Though the total amount of computation work stays the same, the GPU will allow kernel launches from different processes to run concurrently and remove an unnecessary point of serialization from the computation. CUDA MPS is based on client-server architecture. The MPS server manages the GPU hardware resources and these resources including CUDA context belong to MPS clients through the MPS server. This procedure can be client CUDA contexts to bypass the hardware limitations that are associated with GPU time sliced scheduling.

4. Proposed Architecture

4.1. Multi-Core and GPU Tightly Coupled Simulation

To achieve accelerating a large-scale hybrid system simulation in a multi-core and a GPU-coupled architecture, it is essential to divide the simulation process among each device on the characteristics of both systems and devices. GPUs are particularly beneficial when applying the same instruction to multiple data and calculating the amount of floating point data.

The large-scale hybrid system models can be applied to the complex-connected IoT applications, which consist of multiple heterogeneous models such as sensors, processors, networking and actuators. The complex behavior of IoT-driven applications can be generally modeled with the hybrid modeling specification to extend DEVS formalism, so the proposed simulation framework is widely adopted to perform modeling and simulation (M&S) for IoT-driven applications. The continuous time models and discrete-event models of the given IoT applications can be efficiently partitioned and allocated on the CPU and GPU to perform the accelerated simulation by considering the connected behavior of hybrid system models.

The continuous-time models, which are tightly coupled with event-driven fire sensor IoT node models, can be converted into discrete-event simulation models using a small time-advance step.

This conversion approach is enabled by transparent translation without any model modification, but it requires tremendous simulation time synchronization and communication data elaboration. Simulation for continuous-time model updates the continuous state value in high resolution time step, such as evaluation of continuous differential equation and synchronizes the time and data with the correlated models, so the simulation kernel intervention frequently happens after model evaluation finished. Compared to simulator interactions of event-driven models, the simulation time synchronization and communication for sharing the update data result in a lot of computation overhead. We adopted a GPU with a powerful floating-point computing unit and efficiently distributed the converted event-driven models into CPUs and GPUs, so that, by evaluating the continuous-time models in GPU, the fire-spreading behavior could be simulated in a DEVS-based event-driven single-simulation framework by triggering the events on the event-driven sensor IoT nodes in a large-scale parallel manner. The entire simulation process is divided into multi-cores for discrete-event simulation and GPUs for continuous-time simulation, as displayed in Figure 3.

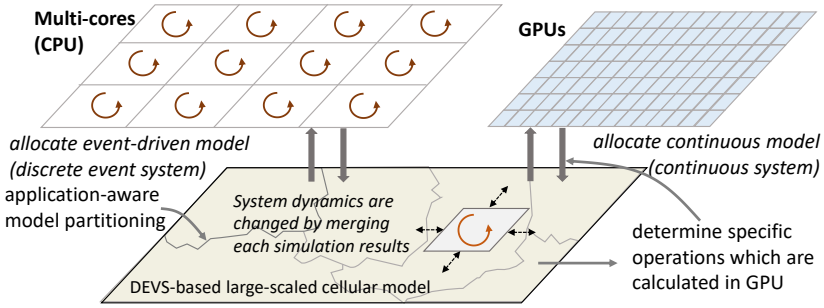


Figure 3. Proposed CPU-GPU tightly coupled computation in DEVS-based simulation.

To provide more detail about the proposed architecture for distributing large-scale hybrid systems to multi-cores and GPUs, we use a cellular model as an example. The cellular model used for the target model in this study consists of multiple cell connections, and each cell has an internal state change that follows differential equations that can be modeled by CS. The large-scale cell model can consist of a large number of cells, such as K cells. In this case, each cell has a CS model based on a discrete time system. Therefore, the process of simulating the CS of each cell that follows the differential equations is repetitive, and if these repeated models are simulated in GPUs, the result will be effective in terms of the simulation speed. Following is an important consideration proposed in this study for implementing the DEVS-based multi-core and GPU framework.

To perform tasks in the GPU, the host CPU is required to copy calculation data into shared memory in the GPU, which is one of the major causes that limits the performance enhancement in CPU-GPU heterogeneous architecture, although the GPU has the advantage of accelerating large amounts of repetitive computation. The process of copying data to the GPU can be ineffective if the copying process happens frequently or with too little data, causing the simulation-specific model evaluation and time synchronization to carefully consider memory management and copy strategies in CPU-GPU tightly-coupled architecture to increase utilization.

Multiple CPUs, including multiple cores, execute the simulation kernel by assigning individual model evaluation in the GPU; this means that multiple access to GPU memory from CPU cores results in irregular bottleneck effects and overall performance degradation due to the CUDA context switching for the iterative evaluation of multiple DEVS-based simulation models. The integrated simulation of DEVS-based converted models of hybrid systems requires a simulator-level manipulation of memory interaction to increase the utilization of large-scale parallel execution in multiple GPU cores.

In this paper, we propose the following two approaches for efficient data copy, distribution, and time synchronization in a large-scale simulation using asymmetric memory-interconnected architecture in heterogeneous CPU-GPU. First, we introduce the fine-grained simulation separation of

event-driven models and continuous-time models to reduce the possibility of the drawback due to the data transfer overhead between CPU-GPU memory. Within maximum simulation time windows in which the two heterogeneous models are independently evaluated, event-driven simulation is performed in advance for all cell-based models in the fire-spreading territory; then, the simulation results are copied as a snapshot to GPU kernel, in which the continuous-time models of each fire-spreading cell are executed at the same time through multiple GPU cores. The experiment demonstrates that the proposed strategy is considered to be one of the best candidates with small data copy overhead between CPU and GPU memory. The simulation time-overhead reduction from this approach is because the small amount of simulation copy data is transferred in advance by interleaving the memory copy and simulation evaluation between two event-driven and continuous-time models.

Second, the separated simulation kernels for a large number of simulation models are distributed into GPU cores, but GPU kernel-call overhead is inevitable, which is also one of the limiting factors of the overall performance of hybrid system simulation in CPU-GPU architecture. We introduce the simulation model description and the GPU utilization method based on CUDA MPS to reduce the time overhead due to the complexity of time synchronization between the large number of simulation models.

To apply the proposed two approaches to extremely increase the utilization of CPU-GPU computing resources, we need to know the characteristics of the target model to accurately determine the maximum time window in which the event-driven simulation is evaluated in advance and in which the continuous-time models are simulated in parallel. Using this time value, we could optimally control the execution flow of the large-scale hybrid system simulation in terms of acceleration, as shown in Figure 4.

The cellular model used as the target model in this study, and similar multiple-state connection models have large numbers of nodes, generally represented by differential equations. Figure 5 illustrates the concept of model specification that we used in our study. In the simulation, we used cellular models, which consist of many hybrid models, represented in Figure 5. The hybrid systems are combinations DES and CS. To accelerate them via partitioning based on their computation characteristics, we proposed a discrete-event simulation in multi-cores (CPUs) and continuous-time simulation in GPUs, as CS follows differential equations that require high-performance computing for accelerated simulation. In addition, we partitioned DES models into several sub-models for simulating in multi-cores. Cellular models can be divided based on their localities, and each simulation of sub cellular models is computed in one CPU core including communication between other sub-cellular models computing other CPU cores.

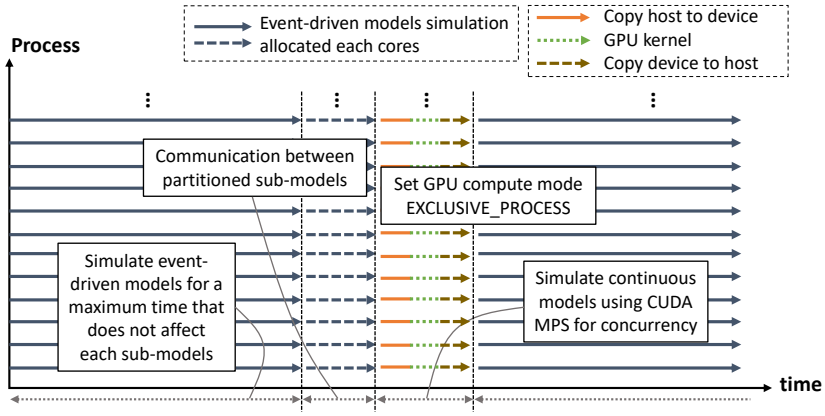


Figure 4. Idea of using GPU in hybrid system simulation.

Figure 6a illustrates DEVS flow of proposed heterogeneous architecture. DES models are allocated in CPU and CS models are allocated in GPU based on their characteristics of computation. In this process, in particular, the memory that is used to simulate CS is copied to GPU memory for computing in GPU. The DEVS simulator, which is adopted for this study, has a clear separation layer between an atomic model and coupled model to connect multiple models, and the corresponding coordinators as

simulation kernels. We could successfully map the simulator evaluating continuous models into the GPU, with minimal modification of simulator framework.

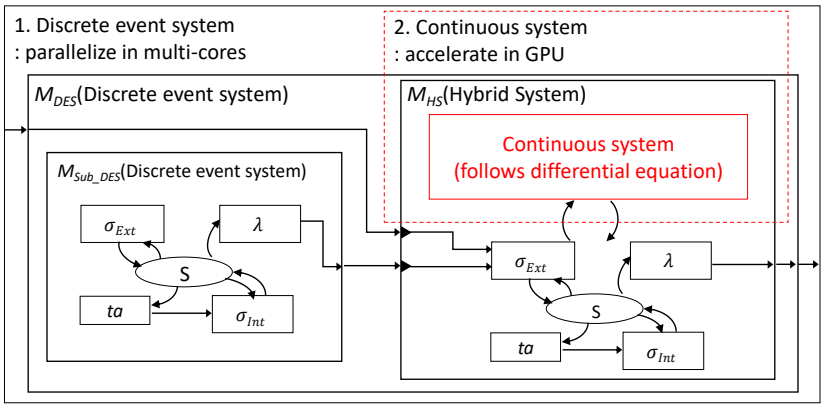
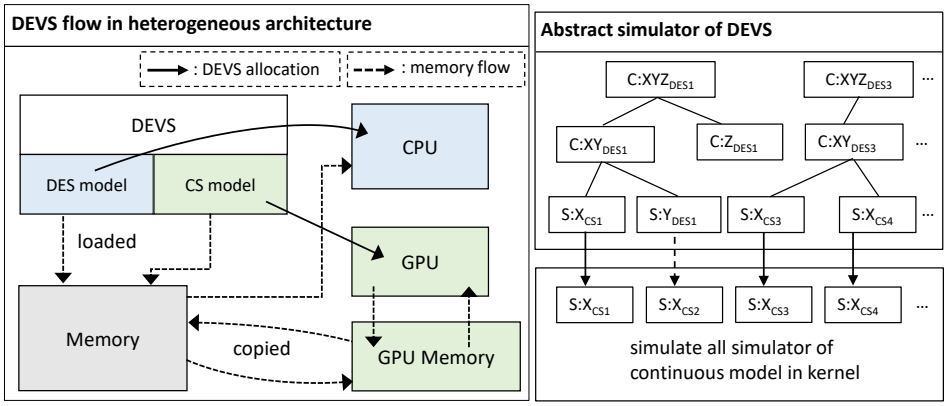


Figure 5. Proposed architecture in terms of model specification.

Figure 6b describes the concepts of abstract DEVS based simulators that proposed in this study. S:X means a simulator for atomic model X, and the atomic model represents the minimum model that cannot be divided into child atomic models. C:XY refers to a coordinator for coupled model XY, and the coupled model integrates multiple atomic models. The simulator updates the simulation clock for the atomic model, and it calls for the four atomic functions for the state transition. The coordinator manages simulation clock and calls four coupled DEVS functions for message routing and conflict resolution. Simulators for the continuous atomic model are simulated in the kernel when, after all, DES are simulated by the time global t_N. A brief DEVS simulation cycle represented by a simulator and a coordinator is displayed in Algorithm 1. The state of the atomic model is changed as global t_N is imminent to the subsequent t_N. Thus, we apply our approach of synchronizing all cellular models by finding out global t_N, which updated the state of the last cellular model. After this global t_N, all atomic models of CS, which, following differential equations, are simulated using multi-threads represented by a thread and a block index in GPUs.

Algorithm 1 Proposed DEVS simulation cycle

for number of K cells
  for discrete-event simulation
    coordinator.send(t_N)
    simulator.reply($next\ t_N$)
    coordinator.compare($next\ t_N$)
    coordinator.send($global\ t_N$)
    simulator.check($global\ t_N$)
    if imminent
      simulator.computes($output_{msg}$)
    end if
    simulator.send($output_{msg}$)
    simulator.computes($state$)
  end for
  for continuous-time simulation by GPU kernel
    for i = 1 to K, i is calculated by thread, block index
      simulator.compute($continuous - time\ model_{(i)}$)
    end for
  end for
end for



(**a**) (**b**)

Figure 6. (**a**) flow of proposed architecture and (**b**) its abstract simulator.

5. Implementation

5.1. Simulation in Multi-Cores and GPUs

The simulation flow of the proposed heterogeneous simulation architecture using multi-cores and GPUs is depicted in Figure 7a. A hybrid model combining DES and CS is simulated using the DEVS engine, and the state-change function of the CS model (i.e., the operation part of the differential equation) is operated on the GPU. The simulation process of the DES model is parallelized by partitioning them into sub-models. The distributed cell-based models are independently evaluated by the simulation kernel across multiple cores of CPU and GPUs within the maximum time steps, which are carefully determined by the simulation model-specific analysis. Then, simulated data, as a result, are shared between the connected models, and state transitions to describe the system dynamics, which are performed in the synchronized simulation time step.

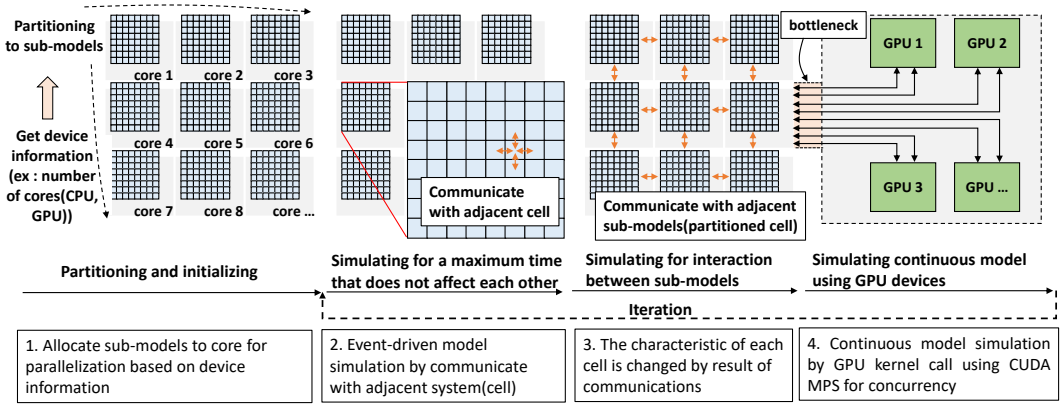
The sliced number of sub-partitioned cell models can be determined and reconfigured by considering the physical number of cores in the target system and the parallelized simulation kernels. The hybrid models, which consist of cell-based territory region models and fire-spreading models, are appropriately separated into the small group of the parallel kernels, which can be simulated, and they are distributed into multiple cores in CPU. The evaluated simulation results are burst copied into continuous-time models of all fire-spreading cell regions, which are allocated in GPU memory.

The continuous-time simulation model of each cell-base territory region is configured with the input initial value from the triggered events, which are results from the event-driven simulation being done in advance. The prepared all-cellular models of the fire-spreading region could be evaluated using the separate simulation kernels in parallel on the GPU by using CUDA MPS for the efficient concurrency of time synchronization to ensure better GPU resource utilization. All the necessary simulation data are then prepared and filled into the simulation models. This style of the interleaved interactions, which carefully manage between DEVS simulation and the simulation-copy operations, resulted in the reduction of the data copy overhead in the GPU kernel memory.

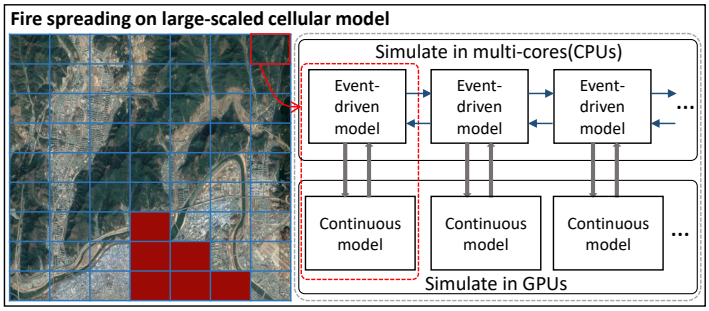
5.2. Fire Spreading Based on a Large-Scale Cellular Model

To analyze the proposed heterogeneous architecture, we simulate a fire propagation phenomenon, as displayed in Figure 7b. We adopted the fire-spreading models as a case study, which is one of the tightly-coupled hybrid model of continuous and discrete-event models, for which inefficient simulation is caused by the single high-speed sequential CPU architecture, but this paper focuses on accelerating the simulation for the cross-coupled complex models of continuous time and discrete event models by utilizing the heterogeneous architecture of CPU-GPU powered hardware platform. The proposed

method can be applied to the simulation for the limited cases of sensor-based IoT applications such as the collaborative fire sensor design in the case of a fire-spreading environment.



(**a**)



(**b**)

Figure 7. (**a**) simulation flow in proposed heterogeneous architecture; (**b**) case study: fire-spreading on large-scale cellular model (base territory image is captured from earth.google.com).

This system is modeled based on the DEVS formalism. Each cell is in an unstable state when the received states of neighbor cells are over a threshold value. This state transition process is modeled in the CS and simulated in the GPU as a proposed method. Each cell is the same as in a DEVS cellular model, and all of the state transition functions in the cells are the same as those in hybrid systems, which have the same differential equations based on the Runge–Kutta method. We can then pick a time to copy the data to the GPU. For a cellular model-based fire-spreading simulation, we can obtain more detailed results by using a large number of cells. The proposed fire-spreading model can be divided into two parts: the overall operation process of each cell, including interaction with neighbor cells and the local state-changing part that is affected by the external and internal states. These two parts are divided into multi-cores and GPUs based on their characteristics that are related to the accelerated simulation.

The structure of the DEVS-based fire-spreading model of the hybrid system is displayed in Figure 8. The M_{Cell} consists of M_{Gen}, which generates states for internal state-change, and M_{Proc}, which changes the neighbor's state value and its own state value. The results of this model evaluation are transferred to the connected state sensing model M_{Sen}, which is used to determine whether the current cell has fire state transition compared to the pre-defined threshold value. Each cell receives temperature data around neighbor cells and determines the next temperature based on its current temperature and heat injections from adjacent cells, for which the propagation model is represented with the differential equation. The temperature variable of this differential equation is compared to the pre-defined threshold value to perform the next transition of stable and unstable state. Therefore, two heterogeneous models of continuous-time model for temperature transition and discrete-event model for fire state transition have a cross-coupled relationship in the simulation process. M_{HS} is

modeled based on the hybrid system's specifications. The specifications of a formal representation of the M_{HS} model is depicted below.

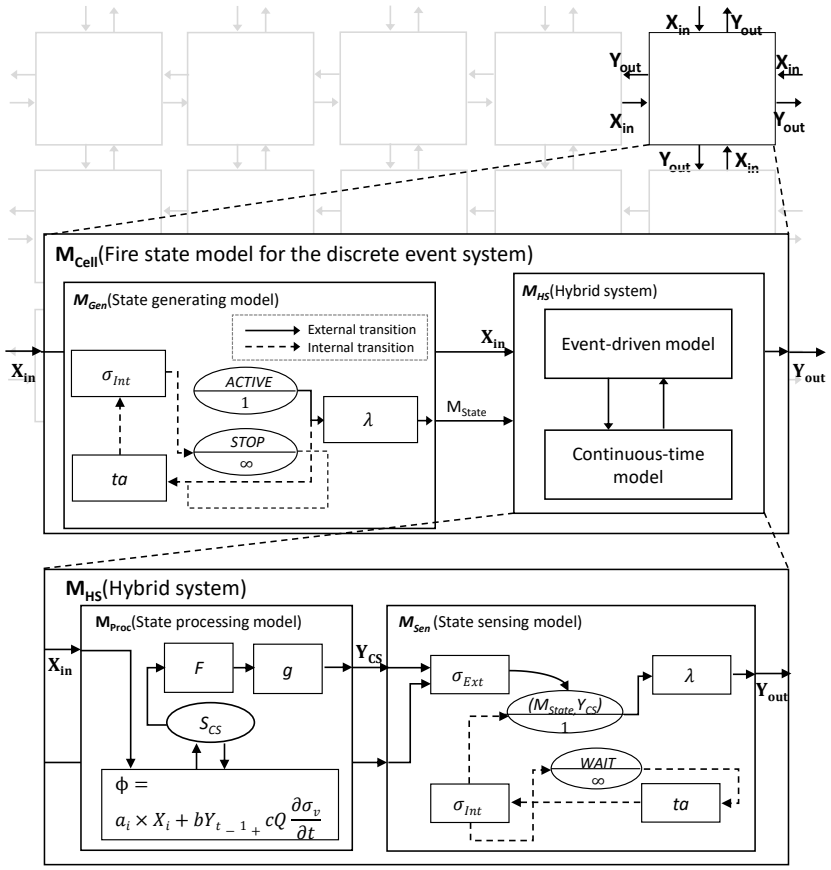


Figure 8. DEVS based fire-spreading model of a large-scale hybrid system.

Definition 1. $M_{HS} =< X, Y, S_{DES}, \sigma_{Ext}, \sigma_{Int}, \lambda, ta, S_{CS}, F, g, \phi >$

- $X = \{X_{in}\} \cup \{M_{state}\}$ is the set of inputs where X_{in} represents vector value of neighbor cells;
- $Y =$ the set of outputs;
- $S_{DES} = (wait,\ done,\ stable,\ unstable)$ is the set of DES model states;
- $\sigma_{Ext} : (Y_{CS}, M_{state}) \rightarrow\ (stable, unstable),\ Y_{CS} \rightarrow\ done$;
- $\sigma_{Int} : Y_{CS} \rightarrow\ (stable, unstable),\ done \rightarrow\ wait$;
- $\lambda : done \rightarrow\ Y_{CS;}$
- $ta : wait \rightarrow\ \infty,\ done \rightarrow\ 1$;
- $S_{CS} =$ set of σ_v which used in internal transition;
- $F = \{f_{i,1}, f_{i,2}, ...f_{i,n}\}$ is the set of threshold sensor;
- $g =$ the set of outputs which represent continuous value;
- $\phi = \{\phi_1, \phi_2, ...\phi_n\}$ is the set of differential equation.

X is the set of inputs, represented by the vector value from neighbor cells, and Y is the set of outputs. In addition, S_{DES} is the set of the discrete event system model states: *wait*, *done*, *stable*, and *unstable*. These states are updated by σ_{Ext} and σ_{Int} to check if the current state is stable or unstable to synchronize their operations. λ is the output function from $S_{DES} \times Y_{CS}$. *ta* is the time advance function in the DES model. For a CS, S_{CS} is the set of states, and F is the set of the threshold sensor for the conversion of the continuous state of the event. Meanwhile, g is the output function, and ϕ represents the differential equations for a CS.

6. Experimental Results

In this paper, we analyzed the proposed architecture in terms of the simulation execution time and the maximum power consumption for each configuration of hardware architecture, as shown in Table 1. Figure 9 displays the hardware used for the proposed heterogeneous architecture analysis. The simulation of a proposed architecture is computed for eight device configurations as displayed in Table 2. The execution time of the simulation is measured by changing the number of cells, the case of hardware configurations, and the time step size to calculate the differential equations using the Runge–Kutta method. Furthermore, we compared the maximum power consumption and execution time of eight cases under the specific simulation conditions in terms of efficiency. All of the measured execution times in the parts of the experiment are maximum simulation time windows in which the two heterogeneous models are independently evaluated.

Table 1. Hardware specifications.

Devices	Model	Cores
CPU	Intel Xeon CPU E5-2620 v4 × 2	16
GPU	NVIDIA GeForce GTX 1080Ti × 4	14,336
Memory	16 GB × 8	-

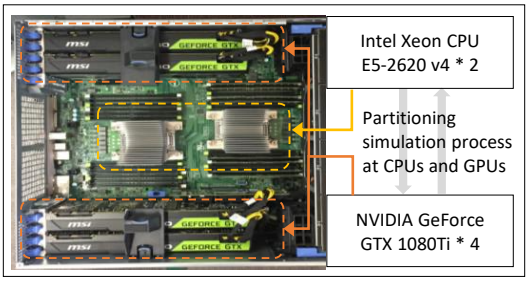


Figure 9. Target hardware for computing a proposed simulation framework.

Table 2. Hardware configurations.

Case	Devices
1	4 CPU cores
2	8 CPU cores
3	12 CPU cores
4	16 CPU cores
5	4 CPU cores, 3584 GPU cores
6	8 CPU cores, 7186 GPU cores
7	12 CPU cores, 10,752 GPU cores
8	16 CPU cores, 14,336 GPU cores

Figure 10 reveals the execution time of the simulation when changing the number of cells and the simulation target device configurations as parameters. We measure the simulation time for about eight cases of hardware configurations, and we measure it by increasing the number of cells from 90,000 to 2,880,000. The step size for computing a CS is 0.0001. The result indicates a higher level of performance in terms of the execution time when a large number of cells are used in the simulation. The results of case 4 and case 8, which use the same number of CPU cores, are as follows. When experimenting using 180,000 cells, case 8 shows that the simulation time is about 10% faster than in case 4. In the experiment using 2,880,000 cells under the same hardware conditions, the simulation time of case 8 is about 80% faster and results represent that the simulation time decreases more as the number of cells increases.

With this result, the proposed CPU-GPU tightly coupled simulation has an advantage when using a large number of cells.

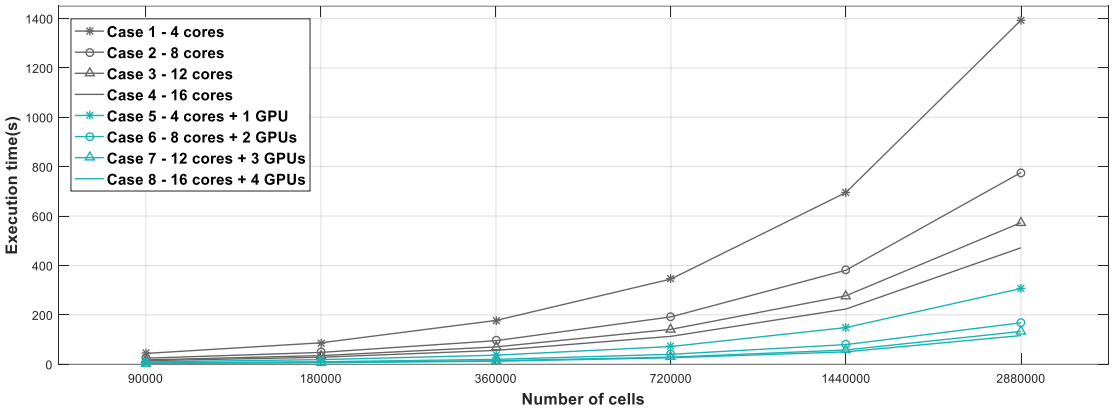


Figure 10. Execution time of simulation according to the number of cells.

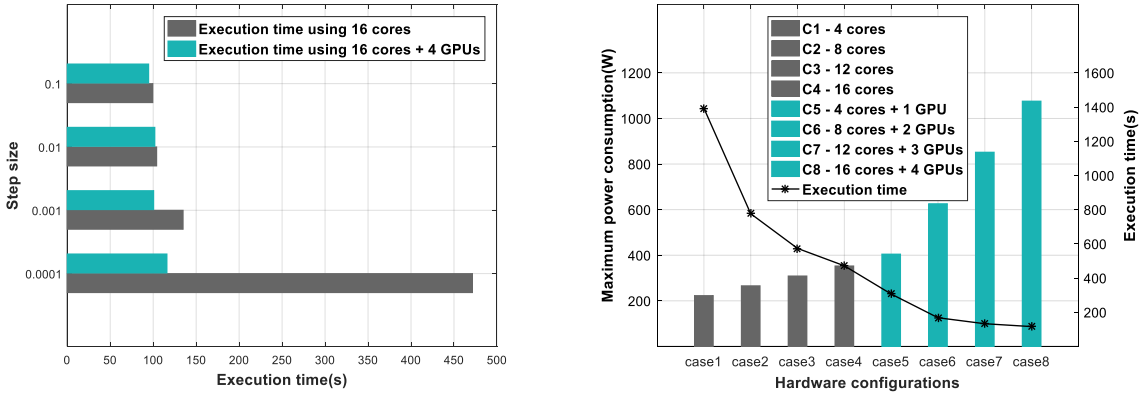
Table 3 represents the simulation time of CUDA MPS-based GPU operations used to reduce overhead due to GPU operation requests on multiple cores. To analyze the performance of MPS for a bottleneck due to GPU access by multiple cores, execution time is analyzed by increasing GPU from 1 to 4 in 16 cores. When 16 tasks are assigned to 16 cores with one GPU device that is requested to operate, a lot of CUDA context switching and underutilization may occur. Therefore, GPU operation takes a lot of time. In this case, the result represents that the GPU operation using CUDA MPS is reduced by 44% or more from the previous 2.206 s to 1.227 s. Likewise, the result reveals that the execution time is reduced to 45.339%, 41.584%, and 12.299% when the CUDA MPS was used for experiment conditions that each have a different number of GPUs. From this result, the use of CUDA MPS in heterogeneous architecture using multi-cores and GPUs improves GPU utilization and reduces the context switching, thereby enabling faster operation by reducing overhead.

Table 3. Reducing execution time of the GPU by using the CUDA MPS.

Hardware Configuration	Without the CUDA MPS	With the CUDA MPS	Reduction Rate
16 cores + 1 GPU	2.206 s	1.227 s	44.369%
16 cores + 2 GPUs	1.289 s	0.704 s	45.339%
16 cores + 3 GPUs	0.951 s	0.565 s	40.571%
16 cores + 4 GPUs	0.666 s	0.537 s	19.354%

The accuracy of a differential equation using the Runge–Kutta method depends on the step size of the time-based calculation step. Generally, a small step size can lead to an accurate result. In this experiment, CS is implemented using first-order differential equations as shown in M_{Proc} model of Figure 8. The simulation execution time is analyzed by changing the time step size. The result is displayed in Figure 11a. The hardware condition of this experiment is 16 CPU cores and four GPUs. We change the step size for computing a CS from 0.1 to 0.0001. As the time step size changes from 0.1 to 0.0001, the difference between the simulation time using only the CPU and the simulation time in the CPU-GPU heterogeneous architecture increases from 0.05 times to 3.1 times. In addition, each execution time of the 16 CPU cores, such as in case 4, has a significant difference. The difference between CPU-only simulation time when using a 0.1 step size and a 0.0001 step size is 3.7 times. However, at the same time step condition, the simulation time difference of the proposed structure, especially in case 8, is only 0.02 times. The results reveal that all of the execution times of the proposed CPU-GPU framework-based simulation are similar and the result shows that the proposed architecture

is very efficient in terms of simulation time when simulating a large-scale system that includes multiple floating-point calculations such as CS.



(**a**) (**b**)

Figure 11. (**a**) execution time of simulation according to the step size of differential equations; (**b**) maximum power consumption according to the hardware configurations.

We also analyze our approach in terms of additional memory usage by the GPU, as shown in Table 4 and detailed the execution time for analyzing the bottleneck when copying data between the host and the device. The table shows that each GPU requires about 680 MB of additional memory; in particular, case 8 requires 2.5 GB of additional memory. Although additional memory is used, we discover that the execution time is dramatically improved by using the proposed heterogeneous simulation architecture.

Table 4. Additional memory usage.

Hardware Configuration	Memory Usage in GPU
4 cores + 1 GPU	686 MB
8 cores + 2 GPUs	1308 MB
12 cores + 3 GPUs	1938 MB
16 cores + 4 GPUs	2552 MB

We compare the maximum power consumption and the simulation execution time for eight hardware cases to find out the efficiency, as depicted in Figure 11b. The power consumption of the CPU is obtained theoretically, and the power consumption of the GPU is obtained using the *nvidia-smi* tool. The experiment condition follows: 2,880,000 cells, 0.0001 step size, and eight hardware configurations. The result shows that the two values of the power consumption and the execution time are inversely related to each other as expected. When high power is consumed in the simulation, high-performance results can be obtained in terms of the execution time. Among the results obtained by the configuration of eight hardware devices, case 4 and case 5, which have similar power consumption, is analyzed as follows. The maximum power consumption of case 5 and case 4 is 405 Watts and 353 Watts, respectively, and the result of case 5 is about 14% higher than that of case 4. However, regarding the execution time of both simulations, case 5 is about 30% faster. That is, the simulation execution time is further reduced compared to the additional power consumption. Likewise, in case 1 and case 8, where the difference between the maximum power consumption and simulation time is the most different, the comparison of the results shows that case 8 consumes 3.8 times more power than case 1, but that the simulation time is reduced by 11 times. By analyzing these results, we conclude that the proposed structure has advantages in reducing the simulation execution time, even with additional power consumption.

7. Conclusions

In this paper, we implemented an accelerated simulation of a DEVS-based hybrid system using a heterogeneous architecture, such as tight coupling of multi-cores and GPUs. Large-scale cellular models are used as the target models that can be represented event-driven IoT applications, especially the fire-spreading model. The partition of the simulation is based on the computation characteristics of the hybrid system, which featured the combination of a discrete event system and continuous time system. The continuous-time simulation that required high-performance computing for differential equations is allocated in the GPUs, and the discrete-event simulation is parallelized in the CPUs. The proposed architecture-based simulation was analyzed at various configurations of target hardware, with the parameters of the computation step size and the number of cells being changed. We analyzed the proposed architecture in terms of power consumption and the execution time. Therefore, by using the proposed architecture, we discovered efficient hardware distribution and configuration through an analysis of the amount of computation power consumption. In addition, in the DEVS-based hybrid system, we analyzed the execution time of the proposed heterogeneous architecture via the calculation step size of the continuous-time simulation. The result shows that the full utilization of CPU-GPU integrated computing resources, on which event-driven models and continuous models are optimally distributed, has an advantage at high-performance simulation in terms of execution time, although more power consumption is required. In future work, this simulation framework will be extended with the interoperation layer to effectively represent the detailed dynamics of real IoT sensors and microcontrollers by directly interacting the simulation models with the hardware IoT sensor systems.

Author Contributions: S.K. designed the entire core architecture and performed the hardware/software implementation and experiments; J.C. performed the validation; D.P. is a principle investigator for this project, who has responsibility as a corresponding author.

Acknowledgments: This study was supported by the BK21 Plus project funded by the Ministry of Education, Korea (21A20131600011). This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2014R1A6A3A04059410) and (2016R1D1A1B03934343) and (NRF-2018R1A6A1A03025109).

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analysis, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

DEVS	Discrete Event System Specification
IoT	Internet of Things
DES	Discrete Event System
CS	Continuous System
CUDA	Compute Unified Device Architecture
CUDA MPS	CUDA Multi Process Service

References

1. Zeigler, B.P.; Kim, T.G.; Praehofer, H. *Theory of Modeling and Simulation*; Academic Press, Inc.: Orlando, FL, USA, 2000.
2. Zeigler, B.P.; Praehofer, H.; Kim, T.G. *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*; Academic Press: Cambridge, MA, USA, 2000.
3. Zeigler, B.P.; Song, H.S.; Kim, T.G.; Praehofer, H. *DEVS Framework for Modelling, Simulation, Analysis, and Design of Hybrid Systems*; Hybrid Systems II; Antsaklis, P., Kohn, W., Nerode, A., Sastry, S., Eds.; Springer: Berlin/Heidelberg, Germany, 1995; pp. 529–551.
4. Zeigler, B.; Moon, Y.; Kim, D.; Ball, G. The DEVS environment for high-performance modeling and simulation. *Comput. Sci. Eng. IEEE* **1997**, *4*, 61–71, doi:10.1109/99.615432.

5. Bergero, F.; Kofman, E. PowerDEVS: A tool for hybrid system modeling and real-time simulation. *Simulation* **2011**, *87*, 113–132, doi:10.1177/0037549710368029.
6. Hong, J.H.; Seo, K.M.; Kim, T.G. Simulation-based optimization for design parameter exploration in hybrid system: A defense system example. *Simulation* **2013**, *89*, 362–380, doi:10.1177/0037549712466707.
7. Wainer, G.A. Modeling and simulation of complex systems with Cell-DEVS. In Proceedings of the 2004 Winter Simulation Conference, Washington, DC, USA, 5–8 December 2004; p. 60, doi:10.1109/WSC.2004.1371301.
8. Liu, Q.; Wainer, G. Parallel Environment for DEVS and Cell-DEVS Models. *Simulation* **2007**, *83*, 449–471.
9. Liu, Q.; Wainer, G. Multicore Acceleration of Discrete Event System Specification Systems. *Simulation* **2012**, *88*, 801–831.
10. Seok, M.G.; Kim, T.G. Parallel Discrete Event Simulation for DEVS Cellular Models Using a GPU. In Proceedings of the 2012 Symposium on High Performance Computing, Orlando, FL, USA, 26–30 March 2012; pp. 11:1–11:7.
11. Saadawi, H.; Wainer, G. On the Verification of Hybrid DEVS Models. In Proceedings of the 2012 Symposium on Theory of Modeling and Simulation—DEVS Integrative M&S Symposium, Orlando, FL, USA, 26–30 March 2012; pp. 26:1–26:8.
12. Chow, A.C.; Zeigler, B.P.; Kim, D.H. Abstract simulator for the parallel DEVS formalism. In Proceedings of the Fifth Annual Conference on AI, and Planning in High Autonomy Systems, Gainesville, FL, USA, 7–9 December 1994; pp. 157–163, doi:10.1109/AIHAS.1994.390488.
13. Troccoli, A.; Wainer, G. Implementing Parallel Cell-DEVS. In Proceedings of the 36th Annual Symposium on Simulation, Washington, DC, USA, 30 March–2 April 2003; p. 273.
14. Chow, A.C.H.; Zeigler, B.P. Parallel DEVS: a parallel, hierarchical, modular modeling formalism. In Proceedings of the 26th Conference on Winter Simulation, Orlando, FL, USA, 11–14 December 1994; pp. 716–722.
15. Jafer, S.; Wainer, G. Flattened Conservative Parallel Simulator for DEVS and CELL-DEVS. In Proceedings of the 2009 International Conference on Computational Science and Engineering, Vancouver, BC, Canada, 29–31 August 2009; pp. 443–448.
16. Himmelspach, J.; Ewald, R.; Leye, S.; Uhrmacher, A.M. Parallel and Distributed Simulation of Parallel DEVS Models. In Proceedings of the 2007 Spring Simulation Multiconference—Volume 2, Norfolk, Virginia, 25–29 March 2007; pp. 249–256.
17. Bergero, F.; Kofman, E.; Cellier, F. A novel parallelization technique for DEVS simulation of continuous and hybrid systems. *Simulation* **2013**, *89*, 663–683.
18. Park, H.; Fishwick, P.A. A GPU-Based Application Framework Supporting Fast Discrete-Event Simulation. *Simulation* **2010**, *86*, 613–628.
19. Wainer, G.A.; Giambiasi, N. Application of the Cell-DEVS Paradigm for Cell Spaces Modelling and Simulation. *Simulation* **2001**, *76*, 22–39.
20. Wainer, G.A.; Giambiasi, N. N-dimensional Cell-DEVS Models. *Discret. Event Dyn. Syst.* **2002**, *12*, 135–157.
21. Ntaimo, L.; Zeigler, B.P.; Vasconcelos, M.J.; Khargharia, B. Forest Fire Spread and Suppression in DEVS. *Simulation* **2004**, *80*, 479–500.
22. Muzy, A.; Innocenti, E.; Aiello, A.; Santucci, J.F.; Wainer, G. Methods for Special Applications: Cell-DEVS Quantization Techniques in a Fire Spreading Application. In Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers. Winter Simulation Conference, San Diego, CA, USA, 8–11 December 2002; pp. 542–549.
23. Hu, X.; Zeigler, B.P. A High Performance Simulation Engine for Large-Scale Cellular DEVS Models. In *High Performance Computing Symposium (HPC'04), Advanced Simulation Technologies Conference*; University of Arizona: Tucson, AZ, USA, 2004; pp. 3–8.