Efficient Parallel Simulation Method of Variable Structure Systems under Multi-core Environments

Journal:	*Transactions on Modeling and Computer Simulation*
Manuscript ID:	TOMACS-2014-0007
Manuscript Type:	Special Issue on Principles on Advanced Discrete Simulation
Date Submitted by the Author:	13-Jan-2014
Complete List of Authors:	Yang, Chen; Beihang University, School of Automation Science and Electrical Engineering Li, Hu; Beihang University, School of Automation Science and Electrical Engineering Chai, Xudong; Beijing Simulation Center, Chi, Peng; Beijing Simulation Center, Lin, Tingyu; Beihang University, School of Automation Science and Electrical Engineering
Keywords:	Simulation Theory: Systems Theory, Simulation Support Systems, Types of Simulation: Discrete Event, Types of Simulation: Parallel

Efficient Parallel Simulation Method of Variable Structure Systems under Multi-core Environments

CHEN YANG and BO HU LI, Beihang University
XUDONG CHAI and PENG CHI, Beijing Simulation Center
TINGYU LIN, Beihang University

Adaptable structure systems motivate the fast development of variable structure formalisms based on DEVS, but simulating large-scale variable structure models challenges current execution methods both in size and complexity. The emergence of multi-core chips presents an exciting opportunity, so an advanced parallel simulator under multi-core environments, particularly with efficient load balancing strategies, is proposed in order to improve the performance and capacity of large-scale variable structure simulation. The simulator: (1) uses formalized models with alterable ports, unified connection management, object management, thread simulation kernels and reader-writer locks to support dynamic model structure in a distributed or centralized way; (2) based on model connections, can explore the inherent parallelism and dynamic parallelism (brought by structure changes) among models, and employ the multi-thread paradigm and shared-variable communication inside an operating system process to gain good speedup; (3) to address evident load imbalance problems caused by structure changes, adopts an efficient dynamic load balancing method, which can migrate models among cores with very low cost and change cores allocated to the simulation dynamically on demand. The results of an application on simulating one airport control system show that the structure change can be supported while up to 23% performance increase can be gained.

Categories and Subject Descriptors: **I.6.1 [Simulation and Modeling]**: Systems Theory; **I.6.7 [Simulation and Modeling]**: Simulation Support Systems; **I.6.8 [Simulation and Modeling]**: Type of Simulation-discrete event, parallel

General Terms: Design, Algorithms, Experimentation

Additional Key Words and Phrases: Dynamic load balance, discrete event simulation, variable structure system, multi-core, multi-threaded, conservative synchronization, complex system

ACM Reference Format:

Chen Yang, Bo Hu Li, Xudong Chai, Peng Chi and Tingyu Lin, 2010. Efficient Parallel Simulation Method of Variable Structure Systems under Multi-core Environments. ACM Trans. Model. Comput. Simul. 9, 4, Article 39 (March 2010), 6 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Many kinds of real systems, especially flexible, adaptable systems, exhibit changes simultaneously at structural and behavior levels [Zeigler and Praehofer 1990; Uhrmacher 2001] when evolving over time, to name a few samples: vulnerability of computer networks and distributed systems to local failures, automated management of failure responses of complex systems, adaptive systems and self-reconfiguring

Author's addresses: C. Yang, B. H. Li, and T. Lin, School of Automation Science and Electrical Engineering, Beihang University; X. Chai and P. Chi, Beijing Simulation Center, Haidian District, Beijing.

DOI:http://dx.doi.org/10.1145/0000000.0000000

computer/robotic organizations [Zeigler and Praehofer 1990; Zeigler et al. 1991]. The growing of these systems motivates the fast development of variable structure in modeling and simulation (M&S) [Barros 1997; Uhrmacher 2001; Hu et al. 2006]. Variable structure can provide powerful modeling capacity and the flexibility to design and analyze these systems [Hu et al. 2006; Uhrmacher et al. 2006]. Several extended formalisms of DEVS, such as DSDE [Barros 1997], DYNDEVS [Uhrmacher 2001], ρ-DEVS [Uhrmacher et al. 2006] have been proposed to support the modeling of variable structure systems. [Hu et al. 2006] further introduced variable ports and presented the pseudo-code of structure variation in DEVS component-based M&S. However, most of these works focus on the theoretical aspect of variable structure M&S, typically proposing extended DEVS formalisms with consideration of the expressiveness and closure, while the simulation execution is not paid enough attention, especially when the increasing demands of simulation models challenge the capabilities of current simulators [Wang et al. 2013]. Today's most cutting edge simulation applications are trying to employ high performance computing to solve challenging problems, such as aerospace vehicle analysis and design [Biswas et al. 2007] on the Columbia supercomputer in NASA, the Earth Simulator Project [Habata et al. 2003] in Japan, the EU-funded Human Brain Project aiming to simulate the brain on supercomputers. So it is of great importance to improve the capacity of large-scale variable structure simulation. *DEVSJAVA 3.0* [Sun and Hu 2009] and Uhrmacher's work [2006] support to modify the structure of the simulation system at run-time with the features of addition/removal of couplings, ports and components as a whole, but it suffers the inefficiency problem caused by unnecessary simulators and coordinators, and unnecessary system-level messages [Muzy and Nutaro 2005]. *adevs* whose base formalism is DYNDEVS [Uhrmacher 2001] can gain comparatively a good performance, but it does not support variable ports. As said in [Hu et al. 2006], the safe and efficient dynamic change of component-based simulation systems need more effort. The booming chip technology of high performance computing further promotes the demand for efficient methods that can fully exploit multi-core or many-core computers to speedup large-scale simulation of variable structure systems. It is pervasive that a high performance computer equips with tens or hundreds of cores, which means that a computer alone can provide powerful computing power for some large-scale simulation applications. To the best of my knowledge, our work - Ivy [Yang et al. 2013] is the first variable structure, and high performance simulator on multi-core machines.

To substantially improve the capacity of simulating large-scale variable structure systems, four aspects of contributions are made:

—*Support flexible structure changes of the simulation system*. The lock-based concurrent execution method enables safe, flexible and dynamic structure changes of the composition model, with little intervention to the simulation execution.

—*Exploit the natural parallelism present in simulation models*. The algorithms can capture the dynamic parallelism among variable structure models, caused by the interaction structure change, to facilitate the parallel execution.

—*Take full advantage of multi-core or many-core machines*. Multi-thread paradigm is adopted to substantially utilize the low communication latency and tight memory integration among the cores on a multi-core chip.

—*Guarantee the load balance among tens or hundreds of cores*. An efficient dynamic load balancing method, which can migrate models among cores with very low cost and change cores allocated to the simulation dynamically on demand, has been proposed to address the load imbalance problems.

We only discuss conservative simulation in this paper, because it is hard or costly to save the model's whole state for rollbacks that is common in optimistic simulation, e.g. models built by using commercial software, legacy models or other models containing irrevocable operations. The algorithms and methods presented in this paper can also be applied to the normal PDES, which can be deemed as a special case of variable structure simulation.

Related works and existing problems are elaborated in section 2. The parallel simulation platform - Ivy which our method bases on is briefly reviewed in section 3. The detailed method and experiment result are discussed in section 4, 5 and section 6 respectively. Finally the conclusion is given in section 7.

2. RELATED WORK AND EXISTING PROBLEMS

2.1 Variable Structure Simulation

Based on DYNDEVS [Uhrmacher 2001], Himmelspach and Uhrmacher [2004] implement a simulation engine with support for structure changes, i.e. the creation and deletion of components, and the change of interactions. However it could not support variable ports and it suffers the low efficiency problem of hierarchical DEVS simulators [Wainer et al. 2011; Zacharewicz et al. 2010; Muzy and Nutaro 2005; Wainer and Giambiasi 2001]. Hu et al. [2006] further introduces port alteration possibilities and pays special attention to the control of structure and interface changes by introducing operation boundary constraints, however, the operation boundary for hierarchical DEVS models only defines an order of structure operations and leads to overly complex control of simulation system, causing the flexibility and efficiency problems. Moreover the author discusses variable structures more from general principles, not real implementation methods and parallelism is not discussed. Uhrmacher and Himmelspach [2006] extend their DYNDEVS by introducing variable ports and multi-couplings for cell biological modeling in DEVS, but its implementation algorithms also suffer the efficiency problem with a hierarchical structure, more practical validations are needed, and parallelism is largely overlooked. Muzy and Hu [2008] propose a framework special for dynamic structure cellular automata and agent. Posse and Vangheluwe [2007] introduce a process description language - *kiltera* to support explicitly the notion of structural change and describe an application of *kiltera* to the problem of distributing tasks among a group of servers, still not with simulation experiment, much less parallel execution strategies. Wang et al. [2011] has proposed one SOA-based dynamic evolution method for SMP2 [European Space Agency 2005] simulation systems, however it is not really "dynamic" structure, because the whole system has to be paused before structure changes, which can largely degrade the efficiency of the simulation.

Thus, an efficient method, supporting comprehensive structure change, probably in a distributed and concurrent way should be designed.

2.2 Simulator Architecture

ThreadedWarped [Miller 2010] adopts master-slave architecture: a manager thread, a global queue of simulation objects and several worker threads, however a single scheduler can easily lead to poor scalability. Chen et al. [2011] improve this by proposing a global scheduling mechanism using several event lists and several active worker threads, each of which repeatedly selects and processes the smallest unprocessed event from all event lists, but portability and reuse issues arise when some attributes of a group LPs are shared. Vitali et al. [2012] present a load sharing approach by allocating the computing power to multi-threads dynamically, but the

overhead of frequent thread switching can be obvious. In contrast, multiple scheduling centers with balancing strategies can minimize this periodic overhead. Tang et al. [2012] bring forward a hierarchical parallel simulator for multi-core clusters, however the mechanism that LPs on a core share the same input queues, makes load balancing difficult to realize and the simultaneous reporting problem [Fujimoto and Hybinette 1997] is not obviously addressed. Targeting at the IBM Cell processor with a heterogeneous architecture, Liu and Wainer [2012] optimize their Cell-DEVS simulator, but that is not a general-purpose simulator. Wang et al. [2013] also overlook load balancing in the simulator *ROSS-MT*.

Distributed simulations usually run faster when smaller divergences exist between the local simulation times on different hosts [El-Khatib and Tropper 1999; Bergero et al. 2012]. In contrast, load imbalance on multi-cores can lead to a severe degradation of system performance [Peschlow et al. 2007], as the risk of blocking and waiting is increased for conservative methods. So a multi-thread simulator with load balancing strategies is preferred in order to gain good efficiency.

2.3 Load Balancing

Research on load balance of distributed simulation can be broadly divided into two sets: (1) metrics for detecting load imbalances and deciding about LP movements; (2) protocols or mechanisms to support load migration. For the first set, there already exist a great number of algorithms on static or dynamic load balance [Gan et al. 2000]. A metric called the effective utilization is introduced in [Reiher and Jefferson 1990], which denotes the amount of useful work done by a processor. The simulation advance rate is calculated based on information about the CPU allocation and the virtual time advance of processors [Glazer and Tropper 1993]. The authors in [Grande and Boukerche 2011] [Angelo and Bracuto 2009] [Peschlow et al. 2007] present dynamic balancing methods considering both communication and computation load, which achieves significant reduction of the running time. [Peschlow et al. 2007] takes account of the number of events processed and virtual time advanced, however, due to the different computation requirement for processing an event, this metric cannot reflect LP's real computation load exactly.

M.-R. Jiang, etc [1994] has proposed one algorithm based on [Glazer and Tropper 1993]. The algorithm can be applied to heterogeneous (processors) and non-dedicated (varied background load) systems. However, [Glazer and Tropper 1993; Jiang et al. 1994] only take into account of Operation System Process (OSP) level load balancing methods, which are applicable to coarse-grained models. Under multi-core environments, LPs are executed in multiple threads and the metric for load of the fine-grained LPs should be redesigned, while fast migration mechanisms with proper consideration of communication should be adopted.

The case is different in multi-core environments to some extent, when all LPs are scheduled in one OSP. Comparing with load balance among OSPs in different nodes, load balance inside an OSP does not necessarily need copy and transfer model state, and exit and join to the simulation. LPs can be created as shared models inside an OSP, so that the efficiency of load migration can be greatly improved with proper algorithms.

Most related work on multi-core environments is [Peng et al. 2012]. Peng et al. [2012] brings out a preliminary load balance method by transferring entity state among threads in one physical process, however, this can be time-consuming when a

large number of LPs need migration. The dynamic addition (or removal) of cores to (from) the system is not included, which is important for efficient green computing.

To address the above problems, based on our initial work in [Yang et al. 2013], a comprehensive analysis of the parallel simulator - *Ivy* is presented, and then a high efficient load balancing method on multi-core machines is proposed. Adopting our method, Ivy can support dynamic structure in a distributed or centralized way, extraction of the inherent parallelism between models, and deadlock-free fine-grained parallel execution of models. Moreover it can migrate unbalanced load between cores without pausing of the system and copying of LPs' state, and can add or release cores on demand. The comprehensive experiment shows that our simulator can achieve a good performance.

3. INTRODUCTION OF IVY UNDER MULTI-CORE ENVIRONMENTS

3.1 Basic Principles of Ivy

Besides the motivation to naturally and effectively simulating complex systems with evolving structures from sociology, ecology, or technology area, variable structure simulation also have advantages to support loose coupling of models, successive development and modularity debugging of the simulation system, and also some openness for executive users to adjust the scenario during execution. Variable structure includes the changing of ports, connections and composition.

(1) Variable Ports

In order to (a) upgrade simulation capability for variable structured systems, (b) loose coupling between Component Model Instances (CMIs) and improve reusability, (c) enhance flexibility for successive development of Simulation Component Models (SCMs), we abstract different data protocols that a CMI uses to communicate with others as ports, such as *Port A1*, *Port A2* of *CMI A* (Figure 1). The port lists and the corresponding operation methods (*addPorts* and *removePorts* in model's unified management interface *MgrInterface* [Yang et al 2013]) are formally included in the model implementation to support variable ports. Via variable ports, significant changes can be signalized to the external world, particularly in the molecular biological domain [Uhrmacher 2006]. The method can also support models to change their brothers' ports as stated in [Hu et al. 2006]. However, we are not intended to artificially define the operation bound as in [Hu et al. 2006], but leave the controlling rules to domain users.

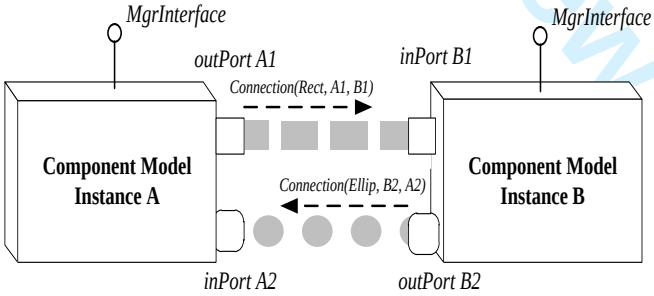

Fig. 1. Ports and connections between CMIs

Example 3.1. In the field control of airplanes, each airplane that plans to take off need firstly send the request to get the permission. Multiple teams of controllers are in charge. Similar to the case given in [Hu et al. 2006], when the frequency of departure airplanes is considerable, new teams of controllers should be added to navigate the airplanes. The transducer should have corresponding ports for each new

team of controllers (Figure 2), to monitor and coordinate the whole operation of airplane taking off. The variation of ports can be easily simulated in our designed supporting system-IVY.

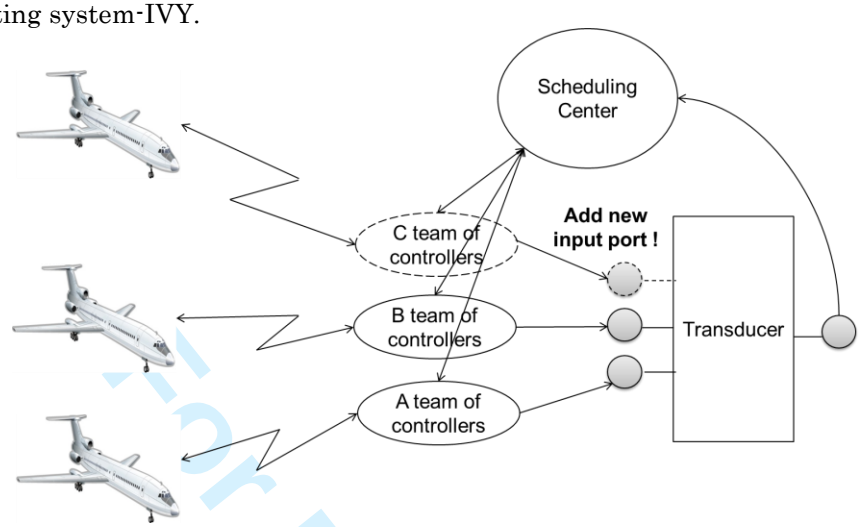

Fig. 2. Add new port for the transducer

(2) Variable Connections

The interaction between CMIs can be described by oriented connections between ports of CMIs, for example *Connection (Rect, A1, B1)* (Figure 1). Directed connections are independent from model implementations, but we should guarantee the type of source ports match that of destination ports. Variation of connections between models can simulate the change of the system network, which can help to study complex systems with variable interaction structure.

Example 3.2. Airplanes will typically fly across different air traffic control areas and need to interact with different control centers to guarantee a safe flight. This can be naturally modeled by connection changes between airplane models and control center models (Figure 3).

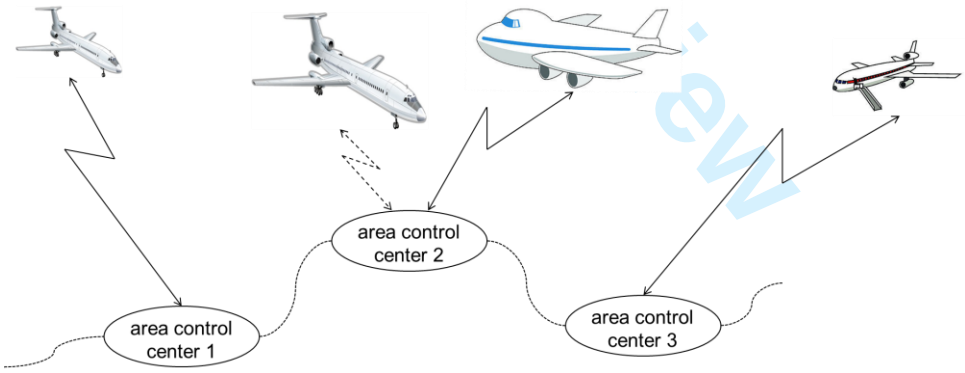

Fig. 3. Variable interaction structure

(3) Variable Composition

When connections related to the model are all removed, then the model can be removed to change composition of simulation systems. Similarly, models can be added to the system by addition of connections and addition of their references to Simulation Engine Instances (SEIs, see section 3.2). However, when necessary, some inactive models can be kept in the heap and initiated as new models to improve the performance of dynamic structure simulation.

Example 3.3. Airplanes will fly over the air traffic control areas when it becomes far enough. If the concentration is paid to the operation of only an air traffic control center, then the airplane models need added to or removed from the simulation system.

(4) Different Meaning of Variable Ports and Connections

We will explain the subtle differences between the removal of input ports, output ports and connections. As shown in Figure 1, we assume that *B*: the model of an electromagnetic broadcaster, *Port B2*: the port to launch waves in a certain frequency, *A*: the model of a receiver, *Port A2*: the port to receive programs in the above frequency, *Connection (Ellip, B2, A2)*: *A* receives waves from *B*. The removal of *Port B2* or *Port A2* models the shutdown or the channel switching of *B* or *A*. There may be several connections from *Port B2*, but the deletion of *Connection (Ellip, B2, A2)* only means the radio transmission to *Port A2* is broken by a jammer and there may exist other uninterrupted connections. The removal of *Port B2* means that all receivers will not receive the wave from *A* anymore, even though the connections exist.

(5) Openness – One Merit of Variable Structure Simulation

Traditional simulation is mostly steered by the scenario, which pre-determines the structure of the simulation system. Generally, the structure cannot be flexibly changed during the runtime, while the parameters are tuned. Some skilled practitioners encoded the conditional structure changes into models, which make for a less elegant and coherent model design, and tend to reduce space efficiency [Uhrmacher 2001]. So structural simulation (as opposed to conventional "trajectory" simulation) is needed to avoid having to force structural changes down to the same level as behavioral ones [Oren 1975]. Moreover, the direct encoding method makes the man-machine interaction at runtime difficult to realize. There are common cases that people will involve and make decisions on the system structure changes according to the simulation situation. However, the stop and restart of the simulation will certainly lead to a waste of the consumed computing power. The supporting of dynamic structure in our work will make this become true and convenient to realize. We can imagine the situation that users interact with the simulation system when necessary, to make the simulation more powerful. Really dynamic structure in our method can help to deal with multi-resolution simulation and on-line simulation, such as symbiotic simulation [Fujimoto et al 2002], dynamic data-driven simulation [National Science Foundation 2005], cyber-physical simulation, to change the structure of simulation models dynamically.

3.2 Execution Architecture and Lifecycle

The execution architecture of IVY can be shown in Figure 4. IVY and models – CMIs compose the simulation system. IVY executes as an OSP, which consists of many thread-level SEIs. SEIs are created to schedule CMIs. To efficiently support simulation of variable structure systems, IVY provides five kinds of core services by making good use of multi-threaded execution and shared variable based communication: object management, connection management, simulation engine instance management, time management and load balance management. The main thread controller existing as the main thread of OSP is designed to respond to user requests, and to configure and control the simulation experiment using the above five kinds of core services of IVY. The life-cycle of IVY, mainly the main thread controller and the SEI, is discussed as follows:

(1) Main Thread Controller

The whole lifecycle of IVY with core services launched by the main thread controller includes:

(a) Object management loads SCMs and instantiates them as CMIs according to application demand. Object management can create or delete CMIs dynamically, when the composition of the simulation system needs changed.

(b) Connection management initiates the network of the simulation system by loading the interaction model and maintains directed connections between ports of CMIs.

(c) Simulation engine instance management creates, initializes, starts, pauses and terminates simulation engine instances. SEIs schedule the execution of CMIs in parallel and are responsible for event passing according to directed connections. Time management is used to synchronize CMIs in the simulation system.

(d) Load balance management migrates CMIs between SEIs, or even employs SEI management to remove or add cores on demand, in order to improve execution efficiency of collaborative tasks on multiple cores.

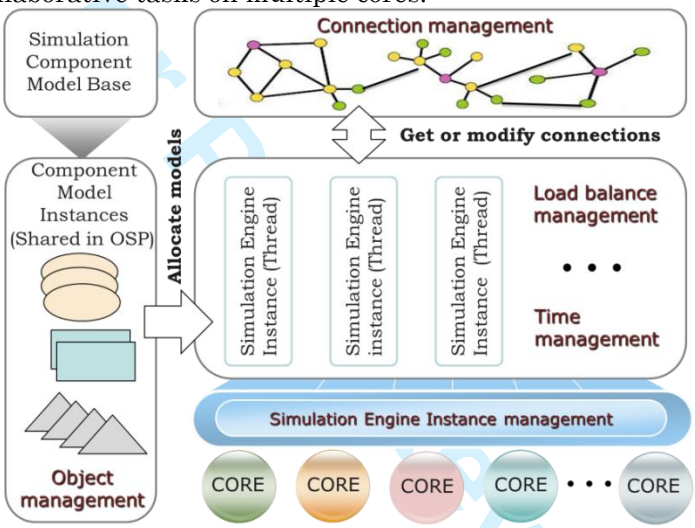

Fig. 4. Execution architecture of Ivy

(2) Simulation Engine Instance

A SEI runs to traverse its CMI reference list repeatedly.

(a) To schedule the next CMI to calculate its LBTS and read safe events from its *inputList* to *waitingList* under the control of LBTS,

(b) To sort events in *waitingList* of the CMI in time order,

(c) To schedule the processing of the earliest event in *waitingList*,

(d) To advance CMI's time to the timestamp of the processed event,

(e) To send generated events to destination CMIs' *inputList*, according to connection management.

(f) Repeat step (c) - (e) until no events exist in *waitingList*, otherwise go to step (a).

4. EFFICIENT STRUCTURE VARIATION AND TIME MANAGEMENT ALGORITHM

4.1 Dynamic and Distributed Structure Variation

(1) Deadlock-free SEI threads

Deadlock is acknowledged as a common problem in multiprocessing systems, parallel computing and distributed systems [Padua and David 2011]. Thus avoiding deadlock plays an important role in the normal execution of our designed system. Ivy uses locks to guarantee the safe access to shared resources among threads [Yang et al.

2013]. Then we made some optimizations by redesigning some data structures and removing some locks. A detailed analysis is shown as following.

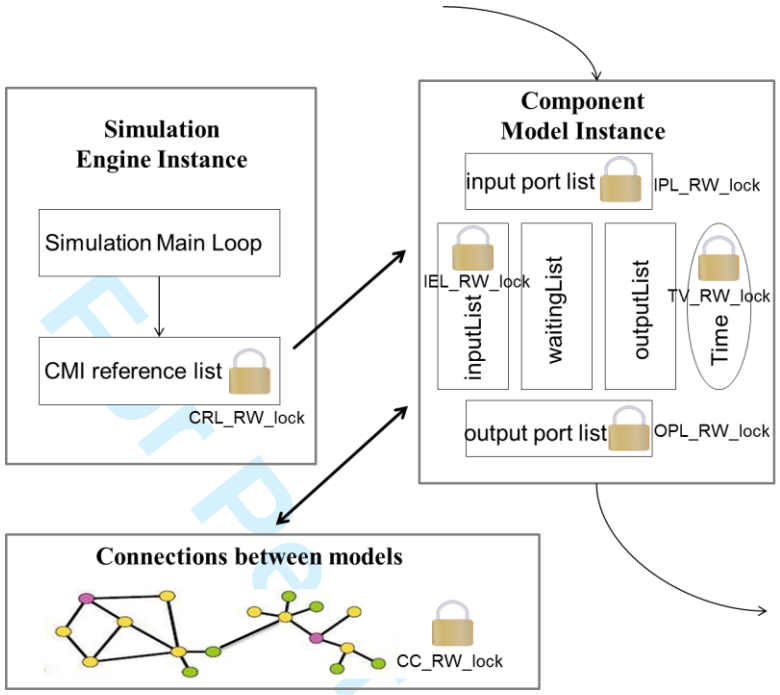

Fig. 5. Locks in IVY

The locks that are used to avoid resource contention include:

Table I. Locks for variable structure in Ivy

Lock Name	Locked Object	Affiliated To	Contention By
CRL_RW_lock	CMI reference list	SEI	MTC[b], SEIs, LB[c]
IPL_RW_lock	input port list	CMI	MTC, SEIs
OPL_RW_lock	output port list	CMI	MTC, SEIs
IEL_RW_lock	input (event) list	CMI	MTC, SEIs
TV_RW_lock	time variable	CMI	MTC, SEIs
CC_RW_lock	CMI connections	CM[a]	MTC, SEIs

[a] CM denotes Connection Management. [b] MTC indicates Main Thread Controller. [c] LB represents Load Balance management.

All these locks are read-write locks. The read-write lock allows multiple reading, single writing and mutually exclusiveness of reading and writing of one shared resource.

Then we will see whether it leads to deadlock, i.e. find out whether there exist cycle dependences. SEI should be analyzed. Figure 6 shows all locks that a SEI thread may try to get in each simulation cycle. Similar to a deadlock of operating system, it is essentially the preemption of lock resources. As shown in Figure 6, ①-⑪ is the complete routine a SEI thread has to experience in its life-cycle. ③-⑨ is repeated by the SEI thread to schedule its CMIs in each simulation loop. The SEI thread is created by SEI management to begin the simulation main loop (step ①). Then it acquires a reader lock of *CRL_RW_lock* to traverse its CMIs (step ②), a reader lock of *CC_RW_lock* to get CMIs that connect to the chosen CMI (step ③),

TV_RW_lock's reader locks of the source CMIs to get their local time to compute LBTS of the chosen CMI (step ④), a reader lock of *IEL_RW_lock* to get the safe events from the input (event) list (step ⑤), a reader lock of *CC_RW_lock* to get CMIs that will receive the new events (step ⑥), a writer lock of *IEL_RW_lock* to write new events to receiver CMIs' input (event) list (step ⑦), the writer lock of *TV_RW_lock* to update the time of the chosen CMI (step ⑧). Step ⑥-⑧ will be repeated after each safe event is processed, until no safe events exist. Then the SEI thread will return and schedule the next CMI in its reference list (repeat Step ③-⑧). *CRL_RW_lock* can guarantee the safe reading and writing of the CMI reference list in a SEI. After a next CMI is chosen, the SEI thread will firstly release *CRL_RW_lock* and then schedule the chosen CMI to process its events. So other SEI threads can access and revise the list after getting the writer lock of *CRL_RW_lock* in the interval.

Any of the above locks will be released after certain task is finished. The entries such as A, B, C, D (Figure 6) may have multi-threads to access, but the threads will release the corresponding lock after certain time during which threads occupying locks are not attempting to get other locks. This contradicts with one necessary condition of a deadlock - the hold and wait condition [Coffman et al. 1971]. So threads in IVY are not deadlocked, which is further verified by our repeated experiments.

(2) Efficiency of Variable Structure

Through our analysis, we can see that the interaction and composition changes of the simulation system requested by CMIs in different SEIs can be achieved in a distributed and unconstrained way. Port changes can be processed by the CMI itself to exhibit the internal state changes by different ports (i.e. reflection [Uhrmacher 2001 and 2006]), or even by other CMIs to enforce external actions. Most importantly, Ivy can achieve really dynamic structure change with little intervention to the normal execution, i.e. any unrelated CMIs can be executed normally, except slight waiting for some CMIs enforced by the spread of LBTS constraints. Comparing with the operation bound method defined in [Hu et al. 2006], Ivy can provide more flexibility of structure changes and leave enough space for users to simulate the complex system using different algorithms. For example, it is not easy to change randomly chosen connections between models by users at runtime using Hu's method. Barros [1997] defined the dynamic structure system network (DSSN) using the DEVS formalism, but the work based on the vision of an executive that resides as a kind of all-mighty atomic model in the coupled model [Uhrmacher et al. 2007], and it might be tedious [Uhrmacher 2001] and show limited capability to deal with complex systems that consist of autonomy entities. Our method based on Ivy can support both the distributed way and the centralized way (autonomy and control [Uhrmacher 1993]) to change model structures, and the way employed to do variable structure simulation is leaved to modelers. As stated in [Sun and Hu 2009], variable structure model can make the simulation more efficient, due to the focus only on "active models" without the burden of all models always active in the system. Still data structures and synchronization method can be designed to gain more efficiency, however this is a starting point, more research effort should be paid.

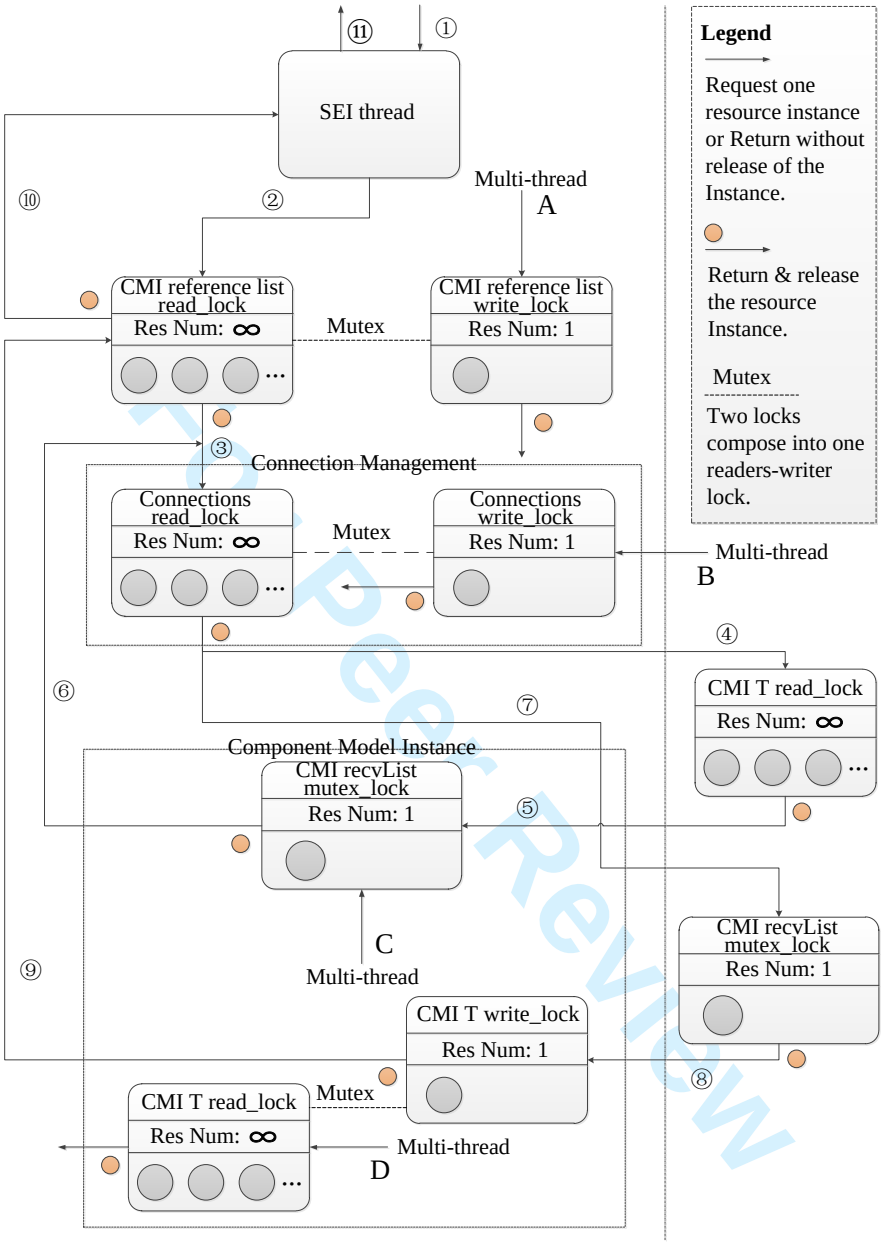

Fig. 6. The executing routine of each SEI thread

4.2 Time Management Algorithm

CMI i has LBTS(i), it can be computed as

$$LBTS(i) = min\{T(j) + LA(j)\} \tag{1}$$

in which CMI i receives messages from CMI j, LA(j) are the lookahead of CMI j, and T(j) subjects to the constraint:

$$T(j) = \begin{cases} T_c(j), \text{if processing safe events} \\ LBTS(j), \text{if no safe events} \end{cases} \tag{2}$$

The transient messages do not exist in our designed supporting system, because the direct insertion of events to destination CMIs with blocking mechanism is employed.

(1) correct synchronization of CMIs

To facilitate the proof, we define the following variables:

T_a : the timestamp of event *a* sent to CMI i

$T_b(i,j)$: the timestamp of event *b* sent from CMI i to CMI j

$T_c(i)$, $T_c(j)$: the current time of CMI i, CMI j

$LA(i)$, $LA(j)$: the lookahead of CMI i, CMI j

Event *b* is scheduled by CMI i, after the process of any event *a*. Assume *b* should be sent to CMI j, in other words CMI j receives messages from CMI i. so

$$T_c(i) = T_a \tag{3}$$

$$T_b(i,j) \geq T_c(i) + LA(i) \tag{4}$$

$$\begin{aligned} T_c(j) &\leq LBTS(j) \\ &= min\{T(m) + LA(m)\} \\ &\leq T_c(i) + LA(i) \end{aligned} \tag{5}$$

CMI i advance its current time to $T_c(i)$ after event *a* is processed, i.e. (3). CMI i can only schedule events with time stamp not less than $T_c(i)+LA(i)$, i.e. (4).

Because there are no transient messages that arrives with the past time stamp, and the scheduling of events is constrained by equation (1) and (2), so (5) is correct. So $T_b(i,j) \geq T_c(j)$, i.e. CMI j will not receive straggler messages. Then by processing received events in time order, CMI j conforms to the local causality constraint [Fujimoto 1990]. Event *a* is scheduled to CMI i, so it can be proofed that CMI i conforms to the local causality constraint similarly. Thus all CMIs are synchronized.

(2) Deadlock avoidance of the time management algorithm

Conservative time management algorithms may lead to a deadlock, manifesting that some CMIs cannot advance their local time anymore.

THEOREM 3.1. If the system encounters a deadlock, there must exist cycles.

PROOF. We assume that there is no cycle in the deadlocked simulation system. Then the system can be abstracted as a Directed Acyclic Graph (DAG), in which nodes represent CMIs and directed connections indicate interaction relationship. The DAG of the simulation system can be topologically sorted as a linear array, so that if the array is aligned in a row, all connections (edges) are directed from the left nodes (vertices) to the right ones. The nodes in the *n-th* position are denoted as CMI n. Intuitively if the upstream nodes, actually CMIs are not pending, the downstream nodes will not encounter a deadlock in a DAG using our proposed algorithm, because the upstream senders that eventually advance their local time make the LBTS of the downstream nodes become larger and larger, so that the downstream nodes can also advance to anywhere in future in a finite amount of time. We will proof it strictly as the following.

LEMMA 3.2. Any CMI n can advance to anywhere in future in a finite amount of time, if the simulation is not over.

The lemma can be translated into mathematic description as:

LEMMA 3.2'. Given any $M > 0$, a $T_M > 0$ can be found, so that after the wall clock time T_M, the virtual time of CMI n, $T(n)>M$.

PROOF. Strong induction is used to proof this lemma.

Basis: $n = 1$, CMI 1 has no senders, so $LBTS(1) = +\infty$. If there are unprocessed events, it is safe to process them without a deadlock and eventually advance its local time to anywhere in future if the simulation is not over. If no events exist, $T(1)=LBTS(1) = +\infty$. Thus Lemma 3.2' holds for $n = 1$.

Induction step: assume Lemma 3.2' holds for $n \le m$.

When $n = m+1$, given any $M(m+1)>0$,

$$LBTS(m+1) = min\{T(h) + LA(h)\}$$

for any upstream CMI h that connects to CMI m+1.

According to the induction step, for any $\delta>0$, $M(m+1)+\delta>0$, a $T_M(k)>0$ for each CMI k ($0<k \le m$) can be found, so that after $T_M(k)$, $T(k)>M(m+1)+\delta$.

We set $T'_M=Max\{T_M(k) \mid 0<k \le m\}$, so after T'_M, $T(l)>M(m+1)+\delta$, for any CMI l that $0<l \le m$. And we assume that CMI p has the smallest $T(p)+LA(p)$ among CMI l that $0<l \le m$.

Then

$$\begin{aligned} LBTS(m+1) &= \min\{T(l) + LA(l)\} \\ &= T(p) + LA(p) \\ &> M(m+1) + \delta \end{aligned} \tag{6}$$

so CMI m+1 can process any events with timestamp less than $M(m + 1) + \delta$, when no events exist, according to (2) and (6),

$$\begin{aligned} T(m+1) &= LBTS(m+1) \\ &> M(m+1) + \delta \\ &> M(m+1) \end{aligned}$$

Thus for $n = m+1$, Lemma 3.2' holds. □

Thus using strong induction, we can infer that the lemma holds. So for any CMI, if any event exists, it will be processed after a finite amount of time. This contradicts to the deadlock assumption. So alternatively there must be cycles in the deadlocked system. □

LEMMA 3.3. Upstream nodes not in the cycles can advance to anywhere in future in a finite amount of time.

PROOF. These nodes and their upstream nodes consist of a DAG, so the lemma can be deduced from lemma 3.2. □

THEOREM 3.4. With our time algorithm there is no deadlock in cycles

PROOF. Assuming that there are unprocessed events, and one of the earliest events is contained in CMI i. Due to the constraint of LBTS, no event can be safely processed. So $T(k) = LBTS(k)=\min\{T(l)+LA(l)\}$, for any CMI k on the cycles.

When deadlocking,

$$\begin{aligned} LBTS(i) &= min\{T(j) + LA(j)\}_{j \to i} \\ &= T(k) + LA(k) \\ &= min\{T(l) + LA(l)\}_{l \to k} + LA(k) \\ &= \cdots \\ &= LBTS(i)_{latest} + LA(m) + \cdots + LA(k) \end{aligned} \tag{7}$$

Because all the upstream nodes can have enough big LBTS after some time, so when deadlocking, all CMI j in (7) belongs to the cycles. Due to finite nodes on the cycles, the constraint node can be ultimately traced to CMI i itself. If any CMI on the cycle has $LA(j) > 0$, LBTS(i) can increase gradually and its unprocessed events will

certainly be processed after some time. Thus the earliest time of events in cycles increases. This contradicts with deadlock of the system. So no deadlock exists on the cycles. However, this algorithm with more efficiency has the same restriction as the null message algorithm: there should be any zero lookahead cycles [Fujimoto 2000]. □

THEOREM 3.5. Downstream nodes will not encounter a deadlock.

PROOF. Due to the incremental time of nodes on upstream and cycles, it can be easily inferred that with enough time advance of nodes on upstream and cycles, the downstream nodes can process any future event. So theorem 3 holds.

If all the nodes do not have events to be processed, then the simulation ends. Otherwise all the nodes can process any future event after a finite amount of time, i.e. the simulation system adopting our time management algorithm can avoid deadlock. □

3) Efficiency Analysis – Dynamic Parallelism

Due to the reasons stated at the beginning, we only consider conservative algorithms in the following analysis. A more related research is [Peng et al. 2012], who adopts a similar multi-thread execution architecture in each federate. However, LBTS of each model in the federate is computed as $\min\{T(i)+LA(i)\}$ for all CMIs of the federate. [Tang et al. 2012] proposed a similar optimized algorithm, by which the approximate LBTS can be computed asynchronously by any thread when needed, and is shared in the federate. These two methods will constrain the extraction of the inherent parallelism between models, especially when the lookahead of models differ greatly.

Our algorithm only takes into account of related models to compute LBTS of the model. When there are changes of the interaction structure, new parallelism (we call it dynamic parallelism) may emerge. Traditional time management algorithms cannot capture this kind of parallelism. For example, when an airplane flies over an air control area, the constrained parallelism between the airplane model and the control center model will not exist anymore, namely the two models can be scheduled totally in parallel. Our algorithm can naturally get this kind of change through connection change to compute LBTS adaptively. Moreover, our algorithm is triggered asynchronously only when no safe events exist in the CMI. As said in the software engineering, there is no silver bullet. Our algorithm which can be more efficient for sparsely and dynamically connected composition models is not so suitable for tightly coupled models. This leaves as future works to evaluate this.

5. LOAD BALANCING OF VARIABLE STRUCTURE SIMULATION SYSTEMS

5.1 High Efficient Migration of CIMs among Cores

Our basic idea is to separate execution related elements of the model from the scheduling threads-SEIs, so that load migration can be easily realized when the model as a whole is migrated. It is appreciated if the executing elements are shared or encapsulated inside the model. When a model is scheduled to process events, four kinds of elements are tightly related, (1) input event queue, (2) current time of the model, (3) model LBTS, and (4) output destinations of events. Next we will elaborate on how the condition is satisfied.

IVY and CMIs compose the simulation system. IVY executes as an OSP, among which CMIs are created and shared. A typical simulation model-a CMI is indicated in Figure 7. According to the formalism defined in [Yang et al 2013] and the implementation of IVY, we can see that the input event queue “inputList”, current time and LBTS of the model are maintained inside the model, so almost three kinds

of related elements are encapsulated inside the model. The computation of LBTS for a model is decided by its message senders, i.e. CMIs that connect to this model. Related connections can be acquired with the help of connection management, which maintains shared connections in the OSP. Similarly, the events are output with the help of connection management by directly inserting events into the input event queue of the destination CMIs. Thus we can see that four kinds of elements related to the model execution are either shared or encapsulated inside the model.

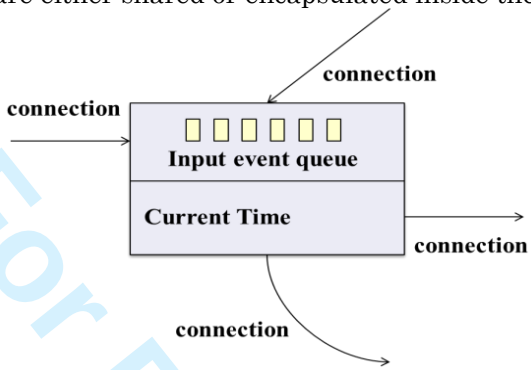

Fig. 7. A model in the simulation system

The behavior of migrated models and affected models should be further checked in order to guarantee correctness of the model execution. There exists one list of CMI references in each SEI (Figure 5, Figure 8). CMI references are used to access model data. Referenced CMIs can be traversed by SEI to schedule LBTS computing, processing of safe events and event output in each simulation cycle. The event processing routine of a CMI is that (1) events are inserted into the input queue of the CMI, (2) model LBTS is computed using the local time and lookahead of models that connect to the CMI, (3) the earliest safe event is processed, (4) the CMI advances to the timestamp of the processed event, (5) newly scheduled events are outputted to destinations by querying connections kept in connection management. Step (3) ~ (5) are repeated until there is no safe event. When there is no safe event, a new simulation cycle is launched from step (2). We can see that dynamic behavior of migrated models is not affected, except for short pause of their execution. So the migration of load - actually models is simplified as the removal of CMI references from source SEIs and the addition of CMI references to destination SEIs.

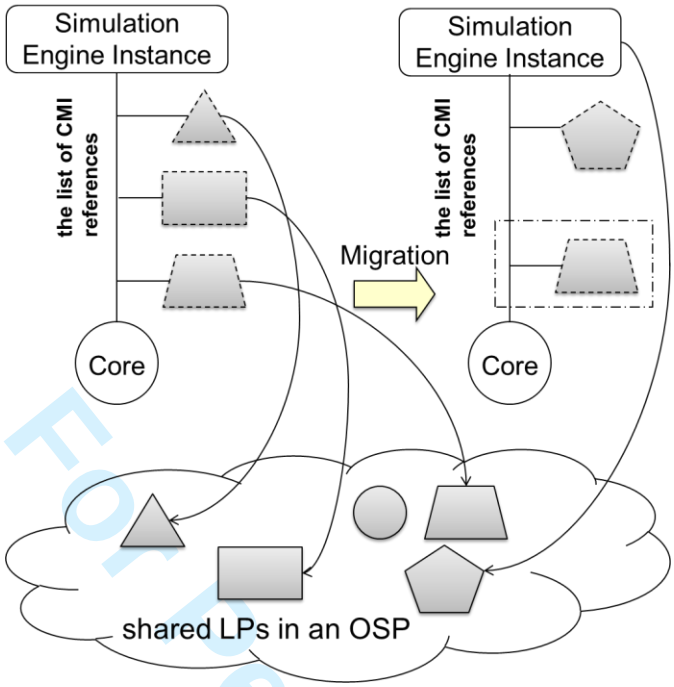

Fig. 8. High efficient migration of load among cores

Operations on the list of model references are controlled by the readers-writer lock *CRL_RW_lock*. The addition and removal function of CMI references are provided by the SEI itself. So while models are being migrated, load balance management will seek to get the writer lock and modify the list in source SEI, and then do so in destination SEI. The execution of unrelated models is almost unaffected, because unrelated models may advance faster and they may be constraint by transiently paused sender CMIs that are being migrated (due to the local causality constraint). During the migrating process, there is a short interval when models are removed from source SEI and not added to destination SEI. However, due to the unchanged connections among models, messages are normally received without the loss of message and the computation of LBTS for receivers is not affected.
So this can be a high efficient way for migration of models without pausing of the system, and without copying and transferring of the model state.

5.2 Dynamic Allocation of Core Resource

Dynamic structure (including the change of composition, interaction structure and model port) can cause inefficiency in a large chance, if the computing resource initially allocated is not adjusted. Reallocation of the occupied computing resource to models can be attained using the above load balancing method (see section 5.1). However this is not enough to achieve high utility of the computing resource, because it is highly possible that less cores decrease synchronization cost for fewer models or more cores increase the running speed by providing more computation power. So the dynamic allocation of core resource is necessary for the simulation systems with large structure change.

Our method to allocate core resource dynamically is to create or delete SEI threads on cores. For the decrease process of occupied cores, the first step is migration of models from SEIs to be deleted, with the aim to decrease communication

cost and balance computation load. And then the redundant SEIs can be safely removed. For the process of adding new cores, SEI management would create SEI threads first and then load balance management would migrate models to the new SEI threads. The whole process can be executed dynamically in the simulation runtime, which can raise the running efficiency with little intervention of the simulation execution. Vitali et al. [2012] firstly proposed a parallel simulation method that can change the number of worker threads within a process in response to the workload variation. Our method can also change the number of allocated cores dynamically on demand to optimize the performance; moreover, the computationally intensive effort of thread context switching is saved. This enables good scalability and can cope effectively with the changing requirement for computing resource.

5.3 Load Balancing Algorithm

Due to extraordinary fast inter-thread communication (delay in nanosecond range), the computation speed of CMIs becomes a primary affecting factor and the communication requirement is taken into consideration as a secondary optional factor.

Inspired by works in [Peschlow et al. 2007; Jiang et al. 1994], a metric for fine-grained component models is proposed by considering event processing time of them. During a monitoring interval, T_{Mtr}, the event set $evtSet(m)$ follows:

$$evtSet(m) = \{evt_i \mid timestamp(evt_i) \in T_{Mtr}, evt_i \in safeEventList(m)\}$$

Events in $evtSet(m)$ are processed and $Advance_m$ is the simulation advance of CMI m. The load of CMI m is a measure of the amount of CPU time it needs to advance its local simulation clock one unit. We consider only homogeneous cores in multi-core machines, because the metric for heterogeneous cores can be easily deduced like [Jiang et al. 1994].

$$Load_m = \frac{CPU_m}{Advance_m} = \frac{\sum_{evt_i \in evtSet(m)} CPU_m(evt_i)}{Advance_m} \tag{8}$$

The LP load metric (actually the CMI load metric) in [Peschlow et al. 2007] can be acquired by setting $CPU_m(evt_i)$ as one unified value. However, the corresponding metric in [Peschlow et al. 2007] cannot reflect the LP load if events need varied CPU time to be processed. Assumed that there exist LPs whose total number is M in the monitoring phase, and M' in the next phase, the mean load of the system and future total load in the next phase are defined as:

$$MEAN = \frac{\sum_{m=1}^{M} Load_m}{M} \tag{9}$$

$$LPsLoad = \sum_{m=1}^{M'} Load_m \tag{10}$$

The loads of new CMIs are evaluated as the same as CMIs that have the same template SCM. If there are no existing CMIs instantiated with the same SCM, the load of each new CMI are evaluated as $MEAN$. So the future load is the sum of all CMIs in the system as equation (10). In order to reduce the cost by dynamic instantiation of the SCM, certain number of CMIs can be kept and reused after re-initialized. This leaves as the future work.

The algorithm can be configured, according to the following two cases.

1) Dynamic balancing with limited available cores

When the computational infrastructure is not dedicated, it is highly possible that available cores are limited, due to background load of other applications. As the fluctuation of the background load, the available cores change over time, indicated in equation (11). Moreover, the required computation resource changes because of the variation of the system composition, reflected in equation (10). So the load balance service should further proactively eliminate possible imbalance problems caused by the change of the system composition.

Ratios of the CPU allocation by processor n to the whole CPU allocation can be set to

$$FRAC_n = \frac{EffCPU_n}{EffCPU} = \frac{EffCPU_n}{\sum_{j=1}^{N} EffCPU_j} \tag{11}$$

So the load undertaken by processor n is defined as:

$$CoreAlloc_n = FRAC_n \times LPsLoad \tag{12}$$

This is a classical "bin-packing" problem that any matured bin packing algorithm can be employed, such as [Jiang et al. 1994; Coffman et al. 1997; Lewis 2009]. We have implemented one bin-packing algorithm described in [Jiang et al. 1994].

2) Dynamic balancing with unlimited available cores

When there exist enough available cores in multi-core machines, cores can be allocated dedicatedly to the SEI threads. However, too many cores may lead to costly communication (synchronization) between LPs on different cores. Thus in this case communication should not be neglected any more.

A SEI will schedule event passing to destination CMIs with the help of connection management. During this process, the number of events communicated between LPs can be naturally acquired to compute equation (15) and (16). These data are employed to reduce communication cost on the basis of computation load balance. The size of events is generally overlooked, because the communication delays are more or less the same using shared variable communication.

$$RatioSyncCost = \frac{CommCPU}{EffCPU} = \frac{EffCPU - \sum_{m=1}^{M} CPU_m}{EffCPU} \tag{13}$$

Where RatioSyncCost indicates ratio of the communication cost to the whole cost of CPU.

$$minCost = w_{comm} \times \frac{Comm'}{Comm} \times CommCPU + w_{comp} \times \frac{Load'}{Load} \times \frac{CPU}{CPU'} \tag{14}$$

$$Comm_{m \to n} = \frac{LpEvents_{m \to n}}{Advance_m} \tag{15}$$

$$Comm_{m,n} = Comm_{m \to n} + Comm_{n \to m} \tag{16}$$

$$Comm = \sum_{CMI_m \in SEI_i, CMI_n \in SEI_j} Comm_{m,n} \tag{17}$$

{Comm, Load, CPU} and {Comm', Load', CPU'} are states of the simulation system before and after dynamic balancing. Comm and Comm' denote the total inter-thread communication by considering interactions between LPs in different SEI threads. Because of only one single thread for each SEI, little communication cost between LPs in one SEI can be achieved.

RatioSyncCost can be set before the simulation to indicate the acceptable threshold of communication cost. When the threshold is exceeded, the load balancing service considering both computation and communication is started up. A large

number of modern computational intelligence methods can be employed to achieve minimal cost in equation (14). The values of weights w_{comm} and w_{comp} depend on application type (computation intensive or communication intensive) of simulation and can be assigned using experiment methods or machine learning methods.

6. EXPERIMENT AND ANALYSIS

According to the standard interface defined in [Yang et al. 2013], the port alteration method is encapsulated inside the model as the unique entry to modify ports, so naturally variable ports can be supported. Our initial application [Yang et al. 2013], in which a tanker aircraft and several normal planes fly in formation to one remote place has validated the removal and addition of connection and system composition. The situation is that: when a normal plane does not receive oil refueling in time, it would land emergently on a nearby airport (exit from the formation fly) and should be removed from the simulation system; before it lands, it will request for reinforce – a new normal plane will join. So we will not explain the detailed support of structure changes in this section.

This section demonstrates the simulation models and experiment results, based on Ivy. Ivy including the load balancing strategies has been tested to acquire its performance on a common simulation system and a variable structure simulation system using a typical scenario of application.

6.1 Simulation Models

The scenario is that multiple teams of controllers in a major airport direct hundreds of airplanes to land or take off. Each team is in charge of airplanes in certain sector airspace that can be scanned with radars. The airplanes enter or depart from the controlled airspace when their distances with the control tower are less or more than 80km. The simulation for this scenario includes the change of airplane models in the simulation system. The simulation system consists of one scheduling center, several teams of controllers and hundreds of airplanes. The landing or taking off plan of airplanes are established finally by the scheduling center. Multiple teams of controllers are responsible for coordination between airplanes and the scheduling center. The airplanes are randomly generated and they join the simulation to simulate the entrance of airplanes into controlled airspace. After landing, the airplane will stay in the airport for different time interval, take off and ultimately depart from the controlled airspace, which is simulated by the removal of airplane models. The time interval is drawn from the same normal distribution with a standard deviation of 10%. The composition and interaction of the whole simulation system can be abstracted as Figure 9.

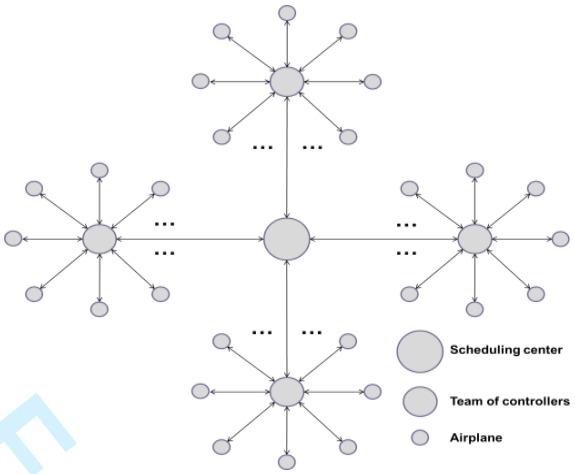

Fig. 9. Model of air traffic control in an airport

6.2 Testing Environment and Results

Because our work focus on the balancing among cores inside one multi-core machine, all tests would run on a high performance multi-core machine with four way 3.07GHz Intel Xeon CPU X5657, 24G RAM. Each CPU contains 6 cores, so 24 independent threads at most can be created to execute exclusively on cores. We have created 500 entities including 1 scheduling center model, 4 team models and 495 airplanes. The event processing time for models was randomly generated among [1ms, 100ms] and saved to simulate different computation requirement of event processing. Ivy and 500 models executed as an OSP host to 4, 8, 12, 16 and 20 threads respectively on the dedicated multi-core machine. The system ran for wall-clock time 10 hours each. Two kinds of conditions are set, 1) the composition of the system is not changed during the system execution; 2) the composition of the system is changed dynamically. Because there are enough available cores, the second method is adopted and the values of weights w_{comm} and w_{comp} is firstly set as 1 and 10 respectively according to our empirical analysis. The cores allocated to the simulation are not changed in this test. The comprehensive experiment covering more applications with dynamic allocation function of cores is the next focus of our work.

1) load balancing of the normal simulation system

Initially, models are scattered evenly among threads on cores. We set the radius of the controlled airspace as an extraordinarily big value. The airplanes are generated at the beginning and randomly distributed in the controlled airspace, so that during the experiment, the composition and interaction structure of the simulation system was not changed. The results of experiments with/without load balancing are shown in Figure 10.

The average performance has been improved by 16.05% (Figure 10), by adopting our load balancing method. With the increase of thread number, our algorithm can achieve better performance, because load balance is increasingly important when there exist many collaborative working threads and system efficiency can be extremely low when load discrepancy is large. Thus we conclude that adopting our balancing method can upgrade performance of the common simulation system.

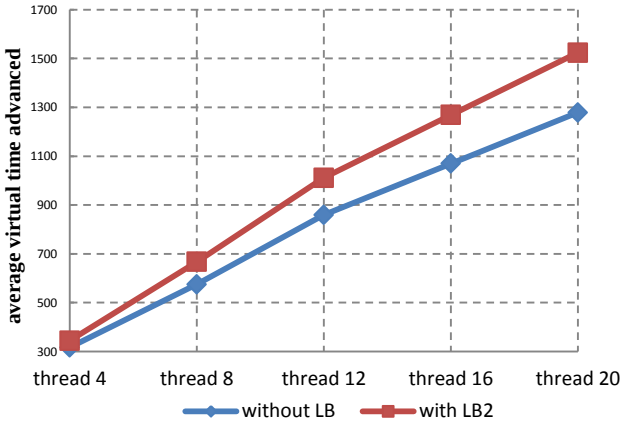

Fig. 10. Experiment results of common simulation

Table I. The Performance Improvement

System Configuration	Thd 4	Thd 8	Thd 12	Thd 16	Thd 20	Aver
Performance improvement	0.0868	0.1610	0.1769	0.1861	0.1918	0.1605

2) load balancing of the simulation system with dynamic structure

In this experiment, the radius of the controlled airspace is set as a normal value - 80km. The airplanes were created to join the simulation and deleted from the simulation dynamically. The airplane would fly in the controlled space and stay in the airport after landing. The comparing experiment results are shown in Figure 11.

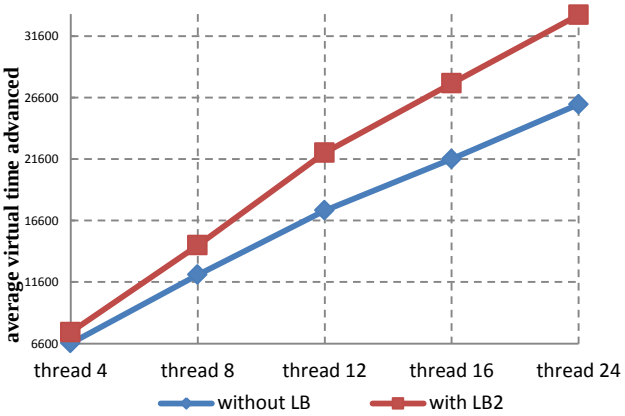

Fig. 11. Experiment results of the variable structure simulation system

Table II. The Performance Improvement

System Configuration	Thd 4	Thd 8	Thd 12	Thd 16	Thd 20	Aver
Performance improvement	0.1352	0.1974	0.2695	0.2848	0.2795	0.2333

From the table of performance improvement, we can infer that our method works well as the number of cores increases. The average performance improvement can be 23.33%, which is better than the common experiment results. With the increase of thread number, our algorithm can achieve better performance. Thus we conclude that adopting our balancing method can upgrade the performance of the variable structure simulation system.

7. CONCLUSIONS AND REMARKS

In this paper, an advanced parallel simulator with load balancing strategies is proposed to support large-scale variable structure simulation. Conventional simulation can also be supported as a special case, when no structure changes are needed. Our initial work [Yang et al. 2013] has presented the normalized interface of a model to provide necessary functions to cooperate with Ivy. The input event list is included inside the model, so that the migration or removal of models can be easily realized without tight couplings with specific SEIs. Connections are shared among threads in the OSP and events are unified passed by connection management, so that even when being migrated (only model references are removed from or added to SEIs), models can still receive messages. The basic principles to support changes of model ports, connections and composition are then introduced. The simulator, which provides five kinds of key services, adopts a peer multi-thread architecture to facilitate fine-grained parallelization. The main thread acting as an executive, is responsible for configure and control the simulation, meanwhile it can also accept request from users to perform structure changes. In other words, some openness of the simulation system is brought. Then the whole lifecycle of Ivy is briefly presented.

Based on the architecture of Ivy, a flexible structure change method is introduced. Reader-writer locks are employed to guarantee the safe concurrency among all operations, namely distributed structure changes and "trajectory" simulations. Deadlock is a common problem for multi-thread applications. However, it can be proved that our method will not encounter any deadlock, using the deadlocking metrics proposed by Coffman [1971]. An improved conservative time management algorithm based on model connections is proposed. The algorithm which can naturally capture the dynamic parallelism between models also enables deadlock-free scheduling of models. We proofed that strictly using the strong induction.

Then an efficient load balancing method, which can seamlessly integrate with other services in Ivy, was proposed. This method minimizes the virtual time discrepancy between LPs among cores at lower cost by using shared models and connections, while keeping each thread always actively schedule models to avoid too much waiting. Metrics in this method further consider the CPU time consumed on event processing and the number of passed events to reflect computation and communication load of simulation for variable structure systems exactly.

Finally, based on our developed simulator Ivy, an application example is given. The simulation results show that our methods (flexible structure change mechanism, dynamic parallelism extraction, fine-grained parallelization on multi-cores and efficient load balancing strategies) can greatly improve system performance. More application examples need implemented to help eliminate the bottlenecks, in order to gain more performance increase.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

The research is supported by the NSFC (National Science Foundation of China) Projects (No. 61103096) in China, the National High-Tech Research and Development Plan of China under Grant No. 2011AA040501, and the Fundamental Research Funds for the Central Universities in China.

REFERENCES

Bernard P. Zeigler, Herbert Praehofer. Systems Theory Challenges in the Simulation of Variable Structure and Intelligent Systems. Computer Aided Systems Theory—EUROCAST '89 Lecture Notes in Computer Science Volume 410, 1990, pp 41-51.

Adelinde Uhrmacher. Dynamic structures in modeling and simulation: a reflective approach. *ACM Trans. Model. Comput. Simul.*, 11(2):206–232, 2001.

Zeigler, B. P., Kim, T. G., Lee, C. 1991. Variable structure modelling methodology: an adaptive computer architecture example. *Transactions of The Society for Computer Simulation*. 7(4):291-319.

A. M. Uhrmacher, J. Himmelspach, etc. INTRODUCING VARIABLE PORTS AND MULTI-COUPLINGS FOR CELL BIOLOGICAL MODELING IN DEVS. In *Proceedings of the 2006 Winter Simulation Conference*.

Hu, X. L., Zeigler, B. P., Mittal, S., Variable Structure in DEVS Component-Based Modeling and Simulation, *simulation*, 2005, DOI: 10.1177/0037549705052227

F. J. Barros. Modeling Formalisms for Dynamic Structure Systems. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 7:501 – 515, 1997.

Wang, J. J., Jagtap, D., Abu-Ghazaleh, N., Ponomarev, D. 2013. Parallel discrete event simulation for multi-core systems: analysis and optimization. *IEEE Transaction on Parallel and Distributed Systems*.

Biswas, Rupak, et al. "Petascale computing: Impact on future NASA missions." Petascale Computing: Architectures and Algorithms (2007): 29-46.

Habata, Shinichi, Mitsuo Yokokawa, and Shigemune Kitawaki. "The earth simulator system." NEC Research and Development 44.1 (2003): 21-26.

Himmelspach, J., Uhrmacher, A. M. 2004. In *Proceedings of the IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. 329-336.

Wainer G, Glinsky E, Gutierrez-Alcaraz M. Studying performance of DEVS modeling and simulation environments using the DEVStone benchmark. *Simulation*, 2011, 87(7): 555-580.

Zacharewicz G, Hamri M E A, Frydman C, et al. A generalized discrete event system (G-DEVS) flattened simulation structure: application to high-level architecture (HLA) compliant simulation of workflow. *Simulation*, 2010, 86(3): 181-197.

Muzy A, Nutaro J J. Algorithms for efficient implementations of the DEVS & DSDEVS abstract simulators[C]// *1st Open International Conference on Modeling & Simulation (OICMS)*. 2005: 273-279.

Wainer, G., and N. Giambiasi, Application of the Cell-DEVS paradigm for cell spaces modeling and simulation. *Simulation*, v. 76, 2001, p. 22-39.

Muzy, A., Hu, X. L., 2008. Specification of Dynamic Structure Cellular Automata & Agents. In *Proceedings of The 14th IEEE Mediterranean Electrotechnical Conference*. 240-246.

Posse, E., Vangheluwe, H. 2007. Kiltera: a simulation language for timed, dynamic structure systems. In *Proceedings of 40th Annual Simulation Symposium*. 293-300.

WANG Chao, YANG Feng, LI Qun, et al. Dynamic evolution method for SMP2 simulation system under service oriented architecture. *Computer Engineering and Applications*, 2011, 47(28):28-32.

European Space Agency. SMP 2.0 Handbook Issue 1 Revision 2[Z]. EGOS-SIM-GEN-TN-0099, 2005.10.

Ryan James Miller, "Optimistic Parallel Discrete Event Simulation on a Beowulf Cluster of Multi-core Machines," Master Dissertation, Cicinati University, July, 2010.

L. Chen, Y. Lu, Y. Yao, S. Peng, and L. Wu, "A well-balanced Time Warp system on multi-core environments," In *Proceedings of the 25th Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 2011, pp. 1–9.

Vitali, R., Pellegrini, A., Quaglia, F. 2012. Load sharing for optimistic parallel simulations on multi-core machines. *ACM SIGMETRICS Performance Evaluation Review*. 40(3): 2-11.

Wenjie Tang, Yiping Yao, Feng Zhu. A hierarchical parallel discrete event simulation kernel for multicore platform, *Cluster Computing*, 2012.

Liu, Q., Wainer, G. Multicore acceleration of Discrete Event System Specification systems. *Simulation*. 88(7): 801-831.

K. El-Khatib and C. Tropper. On metrics for the dynamic load balancing of optimistic simulations. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*. IEEE Computer Society, 1999.

Bergero F, Kofman E, Cellier F. A novel parallelization technique for DEVS simulation of continuous and hybrid systems. *Simulation*, 2012.

Peschlow, P., Honecker, T. and Martini, P. (2007) 'A flexible dynamic partitioning algorithm for optimistic distributed simulation', In *Proceedings of 21th ACM/IEEE/SCS International Workshop on Principles*

of Advanced and Distributed Simulation, IEEE Press, San Diego, California, pp.219–228.
Gan, B.P., Low, Y.H., Jain, S., Turner, S.J., Cai, W., Hsu, W.J. and Huang, S.Y. (2000) , Load balancing for conservative simulation on shared memory multiprocessor systems, In *Proceedings of Fourteenth Workshop on Parallel and Distributed Simulation*, Bologna, Italy, pp.139–146.
P. L. Reiher and D. Jefferson. Virtual time based dynamic load management in the time warp operating system. In *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, pages 103–111. SCS, 1990.
D.W. Glazer and C. Tropper. On process migration and load balancing in time warp. *IEEE Transactions on Parallel and Distributed Systems*, 4(3):318–327, 1993.
Robson E. De Grande, Azzedine Boukerche, Dynamic balancing of communication and computation load for HLA-based simulations on large-scale distributed systems, *J. Parallel Distrib. Comput.* 71 (2011) 40–52.
G. D'Angelo, M. Bracuto, Distributed simulation of large-scale and detailed models, *Int. J. of Simulation and Process Modelling*, 2009 Vol.5, No.2, pp.120 - 131.
M.-R. Jiang, S.-P. Shieh, and C.-L. Liu. Dynamic load balancing in parallel simulation using time warp mechanism. In *Proceedings of the 1994 International Conference on Parallel and Distributed Systems*, pages 222–229. IEEE Computer Society, 1994.
Peng Yong, Cai Ying, Zhong Rong-hua, etc. Parallel Framework for HLA Federate Oriented to Simulation Component on Multicore. *Journal of Software*, 2012,23(8):2188−2206.
Chen Yang, Bo Hu Li, Peng Chi, Ivy: A parallel simulation engine for variable structure systems under multi-core environments. *Int. J. Service and Computing Oriented Manufacturing*, 2013.
Chen Yang, Bo Hu Li, Xudong Chai, Peng Chi, An Efficient Dynamic Load Balancing Method for Simulation of Variable Structure Systems. In *Proceedings of 8th EUROSIM Congress on Modelling and Simulation 2013*, Cardiff, Wales, United Kingdom.
Oren, T., Simulation of time-varying systems, In J. Rose, editor, Advances in cybernetics and systems, Gordon and breach science publishers Ltd., England, 1975.
Padua, David (2011). Encyclopedia of Parallel Computing. Springer. p. 524. Retrieved 28 January 2012
E.G. Coffman, M.J. Elphick, A. Shoshani. System deadlocks. *Computing surveys*. Vol.3, No.2, June 1971.
Uhrmacher A M, Ewald R, John M, et al. Combining micro and macro-modeling in DEVS for computational biology. In *Proceedings of the 39th conference on Winter simulation: 40 years! The best is yet to come*. IEEE Press, 2007: 871-880.
Uhrmacher, Adelinde M. "Variable structure models: autonomy and control answers from two different modeling approaches." *AI, Simulation, and Planning in High Autonomy Systems, 1993. Integrating Virtual Reality and Model-Based Environments. Proceedings. Fourth Annual Conference*. IEEE, 1993.
Yi Sun, Xiaolin Hu, Performance measurement of dynamic structure DEVS for large-scale cellular space models, Simulation, 2009, 85(5): 335-351.
R.M. Fujimoto, Parallel discrete event simulation, *Communications of the ACM* 33 (10) (1990) 30–53.
Christofides, Nicos (1975), Graph theory: an algorithmic approach, Academic Press, pp. 170–174.
Richard M. Fujimoto, Parallel and distributed simulation systems. 2000. JOHN WILEY & SONS, INC. PP. 39-40.
E. G. Coffman, M. R. Garey, D. S. Johnson, Approximation algorithms for bin packing: a survey, Approximation algorithms for NP-hard problems, Pages 46 – 93, PWS Publishing Co. Boston, MA, USA, 1997.
Lewis, R. (2009), "A General-Purpose Hill-Climbing Method for Order Independent Minimum Grouping Problems: A Case Study in Graph Colouring and Bin Packing", *Computers and Operations Research* 36 (7): 2295–2310.

Received January 2014; revised March 2014; accepted June 2014