Available online at www.ijpe-online.com

**International Journal of Performability Engineering**

Vol. 13, No. 6, October 2017, pp. 785-795
DOI: 10.23940/ijpe.17.06.p1.785795

# Schedulability Analysis and Symbolic Verification Method for Heterogeneous Multicore Real-Time Systems

Wei Wang*, Zhengyu Liao, Dong Guo, Hui Zhang, Chunqi Tian, Jianing Tong

*Department of computer science and technology, Tongji University, Shanghai 200092, China*

**Abstract**

As heterogeneous multicore real-time systems are increasingly applied in safety critical systems, it is important to ensure the correctness of these systems. One key attribute of real-time systems is the schedulability that guarantees to satisfy the timing requirements. This paper presents a method for modeling and verifying schedulability of heterogeneous multi-core systems and the method we present uses timed-automata (TA) to model tasks and resources of heterogeneous systems considering their special features. Also this method allows us to establish complex dependences between tasks and use different scheduling policies. After that we choose CPU-GPU heterogeneous multi-core systems as an example and we model three TA networks according to three levels of this system, which are real-time tasks, resources and scheduling management modules. Finally, we use UPPAAL to verify if the model we established satisfies habitudes. According to our method, we present a link between model checking methods and schedulability analysis method for heterogeneous multicore real-time systems and we can automatically and accurately verify the schedulability of selected systems.

*Keywords*: schedulability; model checking; heterogeneous multiprocessor; real-time system; timed automata

(Submitted on July 25, 2017; Revised on August 30, 2017; Accepted on September 15, 2017)

(This paper was presented at the Third International Symposium on System and Software Reliability.)



## 1. Introduction

Real-time systems refer to those information-processing systems that make response to the input from outer in specific time. Their correctness relies not only on logical results, but also on the duration of processing [2]. Real-time systems wildly spread across all-sorted embedded systems, including automobiles, high-speed rails, airplanes and other security-critical systems. For instance, in the field of aviation, flight control software's every conduction must be restrained in a fixed time duration to guarantee the security high precisely. With the approach of industry 4.0 era, real-time systems make strong bonds with people's daily life and safe production, which thus makes the method of how to guarantee the security and correctness to be one of vital research directions among security-critical fields.

On the other hand, with the failure of Moore's Law, the hardware platforms of real-time systems have been turning to heterogeneous multicore processing architecture. The advent of heterogeneous multicore real-time dispatching problems significantly broadens the traditional researches of real-time dispatching. Therefore, the exploration of real-time dispatching methods dealing with heterogeneous multicore system contributes to a core problem in this field.

Paper [2] demonstrates the definable and indefinable situations when utilizing task automaton to analyzing schedulability. What's more, paper [10] expands that system schedulability can be defined in non-preemptive scheduler or fix run-time, with the assumption that the task does not migrate between multiple processors queues. More fascinatingly, paper [7] establishes a model, which can verify the schedulability of real-time systems containing a set of tasks and resources. This model supports multi-properties, including time offset and task dependencies, etc. Paper [12] proposes a UPPAAL model of multi-processor real-time system schedulability analysis, adopting two ways to verify the system model

* Corresponding author.
*E-mail address*: wwang@tongji.edu.cn.

to analyze the system schedulability, via symbolic model checking and statistical model checking. Paper [11] introduces formal methods, and it proposes the establishment of automatic machines for the on-chip multi-core system to verify system schedulability. Yet, this model does not distinguish between different internal architecture of the processor, and thus it does not reflect the heterogeneous features [1]. Paper [13] proposes a method for modeling of heterogeneous multicore systems, but the model is too rough to use model checking method to check the features of these systems. Paper [4] comes up with the description of formal methods considering GPU internal structure. However, it does not take into account the GPU-based task scheduling analysis [9].

To sum up, currently, schedulability analysis methods are mostly used to deal with traditional multi-core CPU system's scheduling problems. In other circumstances, researches of scheduling of GPU or DSP based systems are still limited. This paper will take of multi-core real-time systems into consideration, and use model checking methods to check these features.

## 2. Heterogeneous Multi-Core Real-Time System Scheduling Model

### *2.1. Multi-Core Heterogeneous System*

Multi-core processor systems in accordance with different compositions, can be divided into three categories [3]:

- Same kind while multi-core: All the processing cores are the same in these systems; they have the same computing power, and tasks required in the system are completely equally distributed to all processing cores.
- Same class while multi-core: All the processor cores are equipped with equivalent core class in these systems. For instance, processors are CPUs or DSPs completely. Nevertheless, not every individual processor is indistinguishable in the particular class. That is, maybe all the processors are CPUs, but with divergent computing abilities.
- Heterogeneous while multi-core: These systems include more than two kinds of processors with diverse inner structures, leading to sharp disparity in computing ability. Assignments are supposed to combine their own characteristics with task processors' performances.

Main objects of this paper are heterogeneous multi-core real-time systems, which including Same class while multi-core and heterogeneous while multi-core processor structures. The following are descriptions about these two structures: we use $P_t=\{p_{11},p_{12}\ldots p_{tn}\}$ to represent the system processor, one subscript means processor kinds and the other serves as the subdivision of processors. Therefore, $P_{ij}$ refers to the $i$ processor and the $j$ kind. And thus, we can harness $P_s=\{p_{s1},p_{s2}\ldots p_{sn}\}$ to express heterogeneous while multi-core systems.

### *2.2. Schedulability Framework Based On Model Checking*

Symbolic model checking methods can be used for verifying heterogeneous real-time systems' schedulability. Concrete verification steps for heterogeneous real-time systems schedulability are showed in Figure 1.

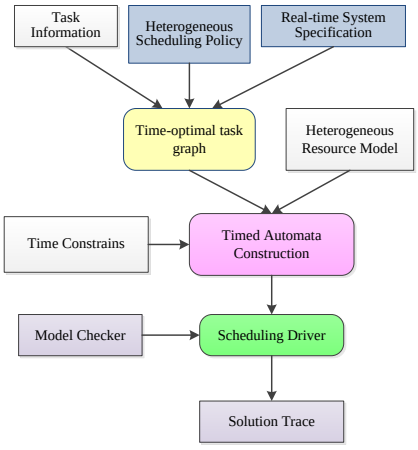


Figure 1. Flow chart of symbolic model checking for heterogeneous multi-core real-time systems

Task information, scheduling policy and systems description describe specific structure of heterogeneous systems via the formal method. Time constraint modules describe the real-time requirements of missions. The above four components cooperatively make up time automata networks, which can make formalized models for heterogeneous multi-core systems in

time and space dimensions. Then, we can select one kind of standard modelling verification tool (UPPAAL) to establish time automata networks of heterogeneous real-time systems to validate the related properties.

### *2.3. Assignment Schedulability Analysis and Modelling*

Traditional computer tasks T can often be divided into a group of multiple sub-tasks such as $\tau_1,\tau_2...\tau_n$, τ represents a single sub-tasks, in the meanwhile, there exists constraints between tasks on time and space by each other. These constraints can be presented by task map $G = ( T, \eta )$ , where $\eta$ is a partial order constraint on a set of tasks, shown in Figure 2.

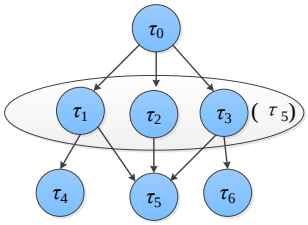


Figure 2.The dependency graph of tasks

*G* is a directed acyclic graph. Nodes in the graph represent the task and edges represent dependencies between tasks. For instance, $\tau_0,\tau_1$ represent $s\tau_1$ can begin to run only after $\tau_0$ comes to an end. We use $\Pi(\tau_5)$ to represent directly dependent tasks set of task $\tau_5$. Only when all the tasks in $\Pi(\tau_5)$ are finished, can $\tau_5$ start to run.

Schedulability problems are composed by multi-task specific dispatching, and thus, we need to define the schedulability for the assignments first of all. Generally, when we say a task is schedulable it means the task's response time meets the constraints of time and also the result. If the system is scheduled, it indicates all tasks are schedulable in the system under the current scheduling policy.

## 3. Framework of Analyzing Schedulability based on Timed Automata

### *3.1. System Model*

Heterogeneous real-time systems' tasks are consisted of task collections, heterogeneous resources and scheduling management model:

- Task collections incorporate a set of limited periodic tasks, and there exists dependencies between tasks
- Heterogeneous resources encompass substantial kinds of processors, a plethora of inter-processor buses
- The scheduling management module is responsible for scheduling tasks to run on a processor

According to above, the module is composed from top to bottom shown as follows:

*System=Alltasks||Allresource||Scheduler;*

*Alltasks=τ ∈TTask (τ )||DependencyManager;*

*Allresource=Processors(different type)||Bus;*

*Scheduler=Scheduler||ρ ∈PPolicy (ρ).*

### *3.2. Task Modeling*

Real-time system tasks are divided into periodic tasks and aperiodic tasks. Periodic tasks have fixed time period T, that is, if the task is created in time t, the corresponsive task will be recreated in time t+T. Aperiodic tasks will be randomly created. In this paper, we mainly talk about periodic tasks. In a periodic task, the main time properties are described below:

- Initial time offset∈ N, time interval between the beginning and the time when the system starts to run the first task of a cycle;
- Best/worst running time period 0< bect ≤ wcet∈ N, every single task running time period at best or worst condition;
- Responsive time respet∈ N, time period from being created to coming to end;
- period∈ N;
- deadline∈ N;
- Executing time exec∈ N and exec∈ [bect, wcet].

We use timed automata to model tasks, and the model is shown in Figure 3.

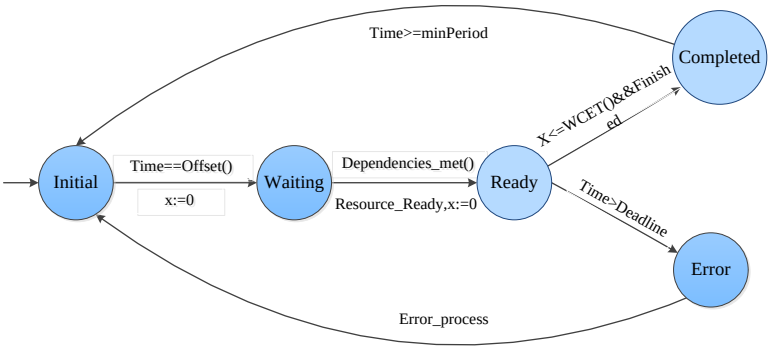


Figure 3. Timed automata model of tasks

Timed Automata model of tasks includes Initial, Waiting, Ready and other two states. When the constraint conditions are met, states will switch automatically. For example, after the initialization of the task is completed, if the time T exceeds the *intialOffset*, the task will switch from *Initial* to *Waiting*; when Dependence_met() is true and its responsive Resource are free, the task state will switch from *Waiting* to *Ready*; and then, if the task running smoothly, which means response time is less than worst time, the task state will switch from *Ready* to *Completed*, else it will turn to *Error*. When one cycle is reached, the task state will switch from *Completed/Error* to *Initial*.

### *3.3. Assignment Dependence Models*

Assignment dependences can be expressed by adjacency matrix. Assuming there are n sub-tasks in the system, they are $\tau_1,\tau_2 \ldots \tau_n$. Then we can use an $n*n$ matrix $J$ to uniquely represent this kind of task dependence. Particularly, if a sub-task $\tau_i$ must execute after task $\tau_j$ is completed, we assume that the value of matrix $J_{ij}$ is 1, else is 0. Thus, only do we need to find out if the all the values of $J_{i*}$ equal to 1 that we will define that task $\tau_j$ meet the executing condition or not. If the tasks where all the values of $J_{i*}$ equal to 1 are completed, the task $i$ meet the executing dependences, else we need to wait for other assignments to be executed.

### *3.4. Resources Models*

Owing to the fact that we have considered the time constraints when we talk about Task Modeling, we need to take the relationship of resource occupancy into consideration when we model the resources. In this fact, resources should have two states: Idle and Inuse.

Other resources (processor) can be divided into two types: preemptive resources and non-preemptive resources. Preemptive resources refer that when the task is running on resource, it can be disrupted, which means that high priority tasks can be preempted to occupy computing resources; non-preemptive resources refers that once the resource is occupied by tasks, the resource can be free only after task is completed or making errors. Tasks won't be interrupted in this situation.

According to the statement above, this paper will establish resource automata model shown as Figure 4. In Figure 4, except the basic statuses such as Idle and Inuse, we add Insert status *S0/S1*. When one single task arrives, it will ask whether the resource is free. If it is free, the resource will switch from Idle to Insert; when the resource is in Inuse, the resource will switch from Inuse to Insert. When the resource is in Insert, it will determine which task would be executed according to the preemption property and priority of tasks.

### *3.5. Scheduling Policy Modeling*

Common scheduling strategies include: First In and First Out (FIFO), Fixed Priority Strategies (FPS) and The Earliest End Time Strategy (EDF). FIFO strategy executes tasks consecutively depending on the beginning time of task execution. Such policy turns assign tasks to different processors to perform in chronological order without disruption or preemption. FPS strategy gave the task a fixed priority when generating the task. When the new task arrives, scheduler decides whether to interrupt the current task according to the task's priority. EDF sorts the different tasks according to the latest end time of the event, and give priorities to tasks so that they can be completed earlier. When the new assignments arrive, task scheduling policies decide whether or not to insert tasks according to this property.

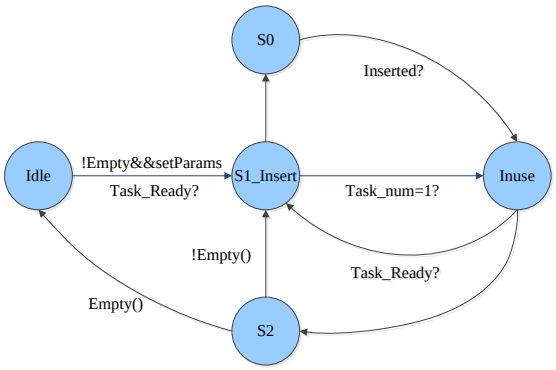


Figure 4. Timed automata model of resources

According to the statement above, for FIFO and FPS, scheduling modules seem relatively easy. Only do we need to find out are the application time and priorities to insert in sequence. Their automata models are shown in Figure 5. For EDF, we need to consider the deadline of tasks and calculate current execution time, leading to relatively complicated models. The specific scheduling policy's model is shown in Figure 6.

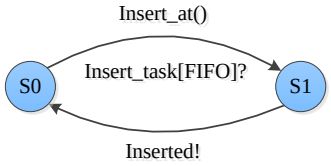


Figure 5. Timed Automata model of FIFO/FPS

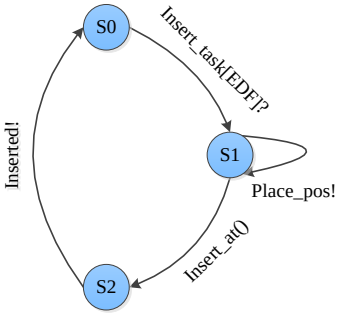


Figure 6. Timed Automata model of EDF

### *3.6. Formal description of Schedulability*

A system's schedulability can be described as the schedulability of all tasks in the system, while a task's schedulability can be described as the fact that the responsive time when meeting the time constraint condition is less than the worst responsive time[3]. Thus, for the system *S*, it includes *n* sub-tasks $\tau_1,\tau_2 \ldots \tau_n$, its schedulability formal description is followed:

$$\forall i \subseteq n, Ext(i) < Bext(i) \,\&\, J_{i*} == 1 \tag{1}$$

In the equation (1), *n* is the number of sub-tasks in the system. *Ext(i)* represents the executing time period of the task *i*. *Bext(i)* represents the worst executing time period of tasks *i*, $J_{i*}$means the constraining relationship with the task *i*.

## 4. CPU-GPU Heterogeneous System Schedulability Verification

### *4.1. CPU-GPU Heterogeneous Systems' Schedulability Modeling*

As the GPU is wildly used, CPU-GPU heterogeneous structures are increasingly applied in real-time system, making the schedulability of computing assignments in GPU and management problems to be a core problem [6,8]. For the purpose of validating the method proposed by this paper, we make models for a specific CPU-GPU heterogeneous system in this

chapter, and verify the schedulability properties of systems. In this paper, we adopt a scheduling system referred in the article [4] as the real example, and its entire schedulability structure is shown in Figure 7. For convenience, we make it more concise, by splitting the whole CPU-GPU heterogeneous system into CPU module, GPU module, the system bus, and shared RAM. Its simplified structure is shown in Figure 8.

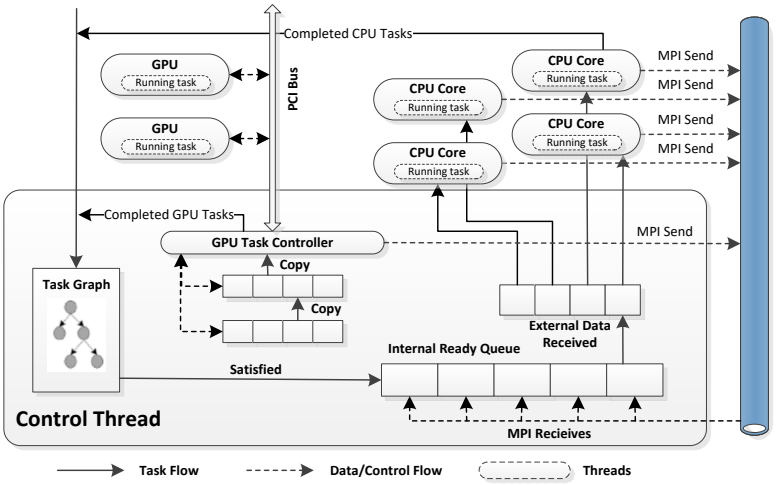


Figure 7. Architecture of CPU-GPU scheduling heterogeneous system

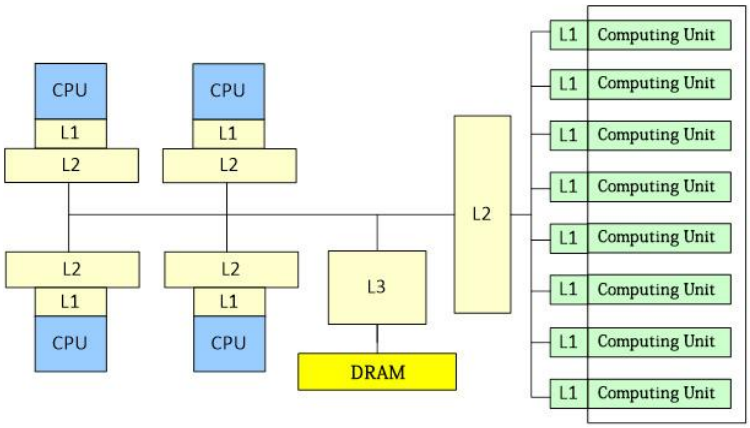


Figure 8. Simplified architecture of CPU-GPU scheduling heterogeneous system

CPU-GPU heterogeneous systems can be separated into two kinds precisely according to scheduling approaches: saster-slave and peer to peer. In master-slave structure, CPU is the main processor while GPU is aided processor, which means, the majority of assignments will be processed by CPU while only a hand of tasks can be available to GPU. However, for peer to peer systems, they value CPU and GPU as computing resources. They have no master-slave fixed relationships. In this article, we will only talk about peer to peer architecture.

In peer to peer heterogeneous structure systems, tasks' scheduling executions are decided by dispatching module. Assignment dispatching module will assign tasks automatically. When a set of tasks arrive, assignments dispatch modules will divide them into different groups for CPUs/GPUs to execute, and CPU and GPU can share real-time data through buses.

### *4.2. CPU-GPU Heterogeneous Systems' Schedulability Models and Analysis*

CPU-GPU heterogeneous systems' schedulability models have been described in chapter one. The following mainly talks about basic GPU computing unit, tasking multi-threaded attribute, more complex tasking relying relationship and RAM occupation and synchronization unit.

#### *4.2.1. GPU Arithmetic Unit Models*

Selecting the most common GPU computing model as an example, CUDA is a processing unit composed of multi-kernels. Every kernel is corresponsive to a grid that is encompassed by several blocks without mutual connection. A block comprises

substantial threads, which share RAM together and can be executed in random order. Through a method called *barrier (_syncthreads())*,they could synchronize. A thread is the smallest unit of execution kernel of its implementation. For this calculation model, threads are lightweight, which means interrupt or switching cost is small.

Therefore, we can assume that taking GPU as the smallest computing unit is an easy structure model reasonably, but the number of it is overwhelming. GPU computing unit timed automata model is shown in Figure 9.

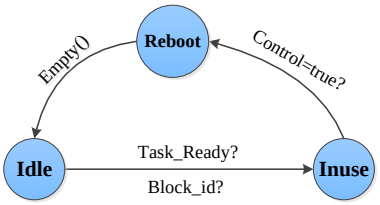


Figure 9. Timed automata of GPU units

This computing model composes of two kinds of status: Inuse and Idle. The same thread block shares the fastest cache when different thread block can't. Thus, the smallest unit in GPU should compose Block_id the property. Finally, based on the property of light thread, we think the task in computing unit is uninterruptible.

If the unit task calculation error occurs or a higher priority task arrives, it will discard the original calculating unit task, turning Reboot to Idle and waiting for the next order. Therefore, computing unit P can be expressed by the following tuples:

$$p = (p\_id, block\_id, policy) \tag{2}$$

*p_id* represents the computing unit *id*; *block_id* represents the *block id*; *policy* represents the computing executing policy.

### *4.2.2. Multithreaded Property Task Models*

Timing property modeling of tasks is similar to the above about GPU, and we don't perform further discussion. On the other hand, due to the SIMD property, we believe that tasks are running on multiple threads nodes, which means that tasks and threads are of one to many relationships. If there happens error in one thread, it will drop the error thread immediately and start another thread. Corresponding to specific tasks, it embodies in one task model corresponding to the plurality of computing units to perform [12].

In such task set T, any task can be expressed as the following tuple:

$$\tau = (\text{id}, \text{n}, \text{p\_id}[], \text{task\_r}(), \text{initoffset}, \text{bect}, \text{wect}, \text{deadline}) \tag{3}$$

In the equation, *id* means task *id*; *n* means the thread number related to the task; *p_id* means the reflection relation between tasks and threads; task_r() is used to express the task resource usage, serving for the thread.

We assume that the entire task is completed. Only when all the tasks in the computing units are turned into error, we think the task is error. And the automata model for tasks in GPU is shown in Figure 10.

### *4.2.3. Task dependencies modeling*

Assignments dispatching in heterogeneous systems are more complex, so the task dependencies cannot simply be represented by task dependency model in the upper section. Considering the fact that different processors are good at different kinds of computing, we allocate the tasks in the most suitable way. For example, CPU specializes in serial logic operation. Therefore, when assigning tasks, we try to assign the tasks that are more logical and task parallelism that are not too much for CPUs to perform; however, the GPU is good at parallel floating-point arithmetic. Thus, we can choose tasks to be concurrently executed with intensive computation assigned to GPUs. So, the model is shown as Figure 11.

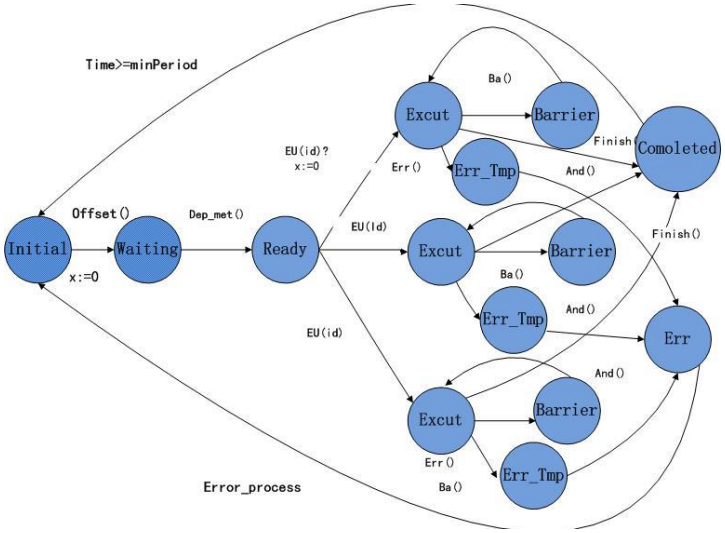


Figure 10. Time automata model of GPU task

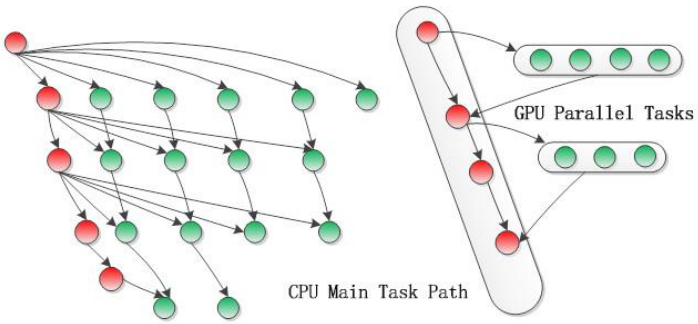


Figure 11. Dependency of complex tasks

This article will divide tasks into the main logic tasks and affiliated computing tasks. The combination of main logic comprises the main path of executing assignments. This part of the main task of the heterogeneous system is executed by CPU module, while the subsidiary of the computing tasks is executed by GPU. For any system S, its tasks are encompassed by $T_C=\{\tau_{C1},\tau_{C2}\ldots\tau_{Cn}\}$ and $T_G=\{\tau_{G1},\tau_{G2}\ldots\tau_{Gm}\}$. Then the dependency system S of each task can be made by three unique adjacency matrix $Matrix_{rr}$ $Matrix_{rg}$ $Matrix_{gg}$ to represent. $Matrix_{rr}$ means the main dependencies between tasks in main paths. $Matrix_{rq}$ means dependence of the main task and the subordinate tasks. $Matrix_{qq}$ shows the dependence between subordinate tasks.

### *4.2.4. Memory usage and synchronization module modeling*

In the process of executing tasks, diverse paths may be occupied. However, different paths may occupy the same group of data or object. Thus, we have to face the preemption of the same memory and synchronization data problem. GPU synchronizes by synchronizing thread *barrier()*. Specific tasks can tell the synchronizing thread whether to synchronize via this property of *Task_r()*. The automata module of *Barrier* thread is shown in Figure 12.

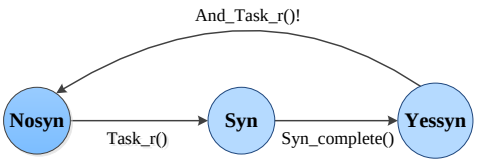


Figure 12. Automata of synchronization in threads

The state machine includes three states: unsynchronized, synchronizing and synchronized. Every single task has the property of *Task_r()*. When the synchronizing thread receives the *Task_r()*, it starts to synchronize the data. By controlling these three states, we can access resources orderly.

### *4.2.5. Scheduling Policy Models*

CPU-GPU heterogeneous system's schedulable policy is quite complex. It not only needs to take the property talked above into consideration, but also allocates different tasks to diverse processor units bases on the property of the task. Here we only selected the end time priority strategies to make modeling scheduling policy for heterogeneous systems. This EDF schedulable model adds a *Choose* status to the models above. The state is responsible for assigning different tasks to different types of processors, its automata is shown in Figure 13.

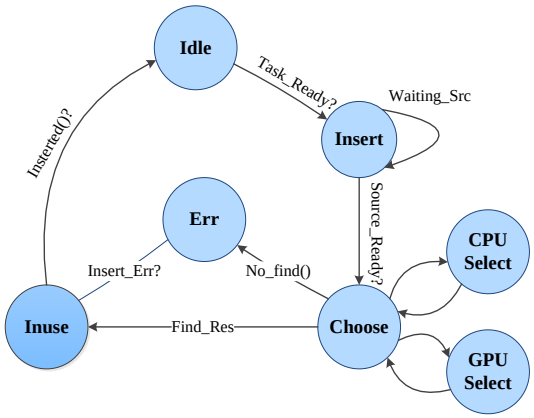


Figure 13. Timed automata model of EDF in heterogeneous systems

In the automata, when the condition of inserting is met, scheduler switches the *Idle* status into *Insert* status. Then the scheduler queries the selecting policy to allocate different tasks to different processors to perform. At the same time the automata switches from *Choose* into *Inuse.* Finally, when the dispatcher receives assignments completed replies, scheduler switches from Inuse status to Idle status.

## 5. UPPAAL-based CPU-GPU Heterogeneous System Schedulability Verification

In this section, we utilized UPPAAL [5]as the verification tool, to verify the schedulability of CPU-GPU heterogeneous systems. The main module part is shown in Figure 14. to Figure 19.

Dependence between tasks and specific properties are given by UPPALL directly.

Aiming to the module above, we select the following properties for the actual verification.

- Property 1: A[] forall (i : tc_id) not TaskCPU(i).Error
- Property 2: A[] forall (i : tg_id) not TaskCPU(i).Error
- Property 3: A[] not deadlock

Property 1 is to verify whether errors take place on computing tasks. Property 2 is to verify whether all of the tasks make error on the GPU modules. Property 3 is to verify whether the system will be in a deadlock. Through the verification of UPPAAL，all of the properties above can't be met. Therefore, GPU heterogeneous systems will be a deadlock in the CPU, GPU parts. The entire system may be in a case of a deadlock. We can solve these problems by designing a better scheduling policy, while scheduling policy design is not the focus of this study. The above experiments show that the proposed method can perform actual verification effectively for the scheduling of heterogeneous systems and it has some practical values.

## 6. Conclusions

This paper presents a method for verifying schedulabilities of multiple heterogeneous systems. Through making the GPU heterogeneous system the actual example research object, we make timed automata for schedulability related parts, including real-time tasks, running platform and schedulable management module, leading to a timed automata network. Then we use symbolic model checking method to model the system's schedulability. Finally, we use UPPAAL to verify whether the schedulable properties are always satisfied.

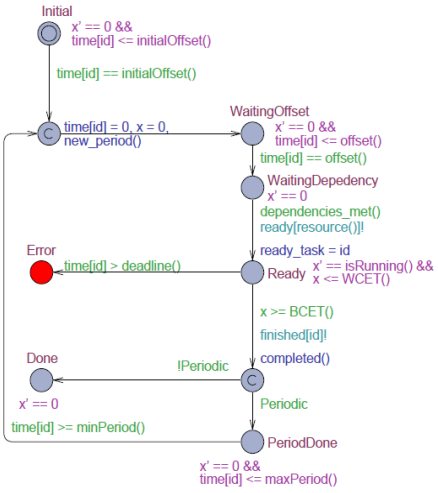


Figure 14. Timed automata of CPU tasks

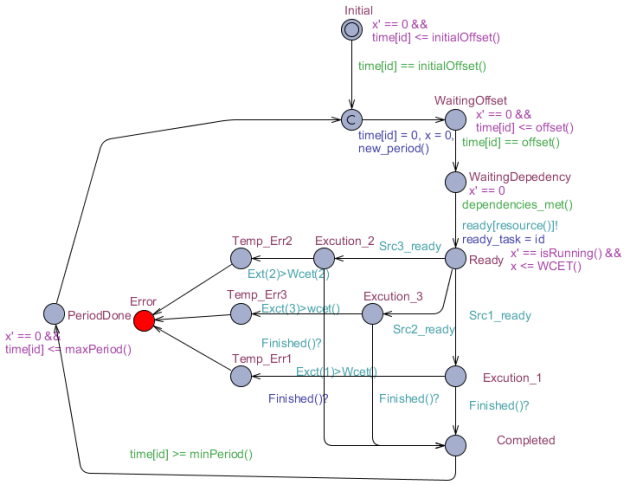


Figure 15. Timed automata of GPU tasks

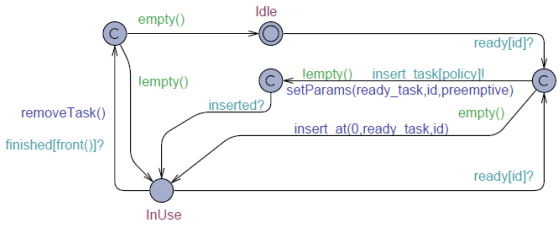


Figure 16. Timed automata of CPU resources

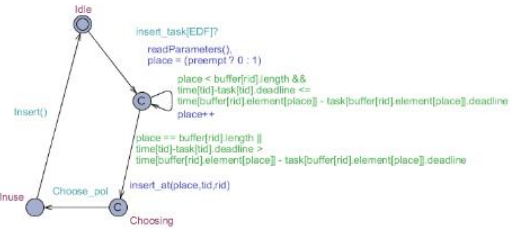


Figure 17. Timed automata of GPU units

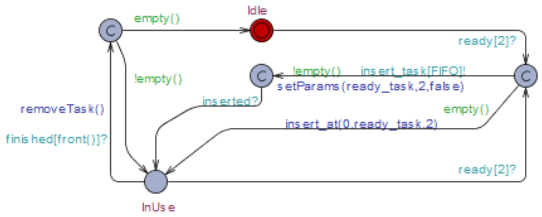


Figure 18. Timed automata of bus

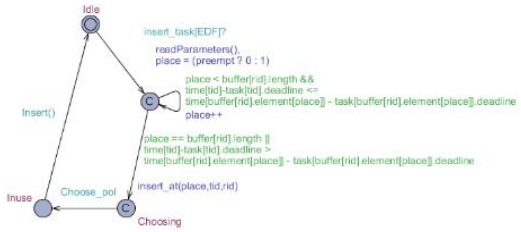


Figure 19. Timed automata of EDF

## References


1. H Alan, QY Meng, B Martin, "Radiation Modeling Using the Uintah Heterogeneous CPU-GPU Runtime System", in *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment* (XSEDE'12), ACM, 2012.
2. G Behrmann, A David, KG Larsen. "A Tutorial on Uppaal[M]//Formal Methods for the Design of Real-Time Systems", *Springer Berlin Heidelberg*, 2004.
3. Sheng-Xin Dai, Mei Hong, Bing Guo, Qiu-Hui Yang, Wei Huang, Bao-Ping Xu, "Schedulability Analysis Model for Multiprocessor Real-Time Systems Using UPPAAL". *Ruan Jian Xue Bao/ Journal of Software*, vol. 26, no. 2, pp. 279-296, 2015.
4. A David, J Illum, KG Larsen, A Skou. "Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1." in *Mosterman PJ, ed. Proc. of the Model-Based Design for Embedded Systems*. Boca Raton: CRC Press, 2010.
5. R. I Davis and A. Burns, "A Survey of Hard Real-Time Scheduling for Multiprocessor Systems", *ACM Computing Surveys*, vol. 43, no. 4, pp. 1-44, 2011.
6. G. Elliott, B Ward, and J Anderson, "GPUSync: A Framework for Real-Time GPU Management", in *Proceedings of the 34th IEEE Real-Time Systems Symposium*, December 2013.
7. E. Fersman, L. Mokrushin, P. Pettersson, Y. Wang. "Schedulability Analysis of Fixed-Priority Systems Using Timed Automata", *Theoretical Computer Science*, vol. 354, no. 2, 2006.
8. R. Gurulingesh, "Real-Time Scheduling on Heterogeneous Multiprocessors", PhD dissertation, 2014.
9. W. Jane, S. Liu, "Real-Time Systems", United States: Pearson Education, 2002.
10. P. Krcal, M. Stigge, W. Yi. "Multi-Processor Schedulability Analysis of Preemptive Real-Time Tasks with Variable Execution Times," in *Proceedings of the Formal Modeling and Analysis of Timed Systems*, Springer-Verlag, 2007.
11. S. Lui, A. Tarek, et al, "Real Time Scheduling Theory: A Historical Perspective," *Journal of Real Time Systems*, vol. 28, no. 2-3, pp. 101-155, 2004.

12. K. Shinpei, L. Karthik, et al, "TimeGraph: GPU Scheduling for Real-Time Multi-Tasking Environments," in *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, 2011.
13. W. Wang, G. Dong, G. S. Zeng, et al, "Reachability Analysis of Cost-Reward Timed Automata for Energy Efficiency Scheduling," in *Proceedings of Programming Models and Applications on Multicores and Manycores* (PMAM'14), ACM, 2014.

**Wei Wang** is an Associate Professor in the Department of Computer Science and Technology at Tongji University, China. He received his Ph.D. in Computer Science and Technology from Tongji University in 2009. He has visited the University of Wisconsin-Madison and University of Florida in 2013 and 2015 separately in USA as a visiting research scholar. His research interests include cloud computing, big data, and formal method. He received IBM Ph.D. Fellowship award in 2007, IBM Teaching Scholarships award in 2011 and Google Teaching Scholarships award in 2012. He is a member of CCF, ACM and IEEE.

**Zhengyu Liao** is a master candidate in the Department of Computer Science and Technology at Tongji University, China. His research interest including formal method and software verification.

**Dong Guo** is a PhD candidate in the Department of Computer Science and Technology at Tongji University, China. His research interest including distributed computing and formal verification.

**Hui Zhang** is a master candidate in the Department of Computer Science and Technology at Tongji University, China. His research interest including formal method and software verification.

**Chunqi Tian** is an Associate Professor in the Department of Computer Science and Technology at Tongji University, China. His research interests include networked system and P2P networks.

**Jianing Tong** is a master candidate in the Department of Computer Science and Technology at Tongji University, China. His research interest including data mining and software verification.