International Journal of Smart Home
Vol. 9, No. 10, (2015), pp. 179-188
http://dx.doi.org/10.14257/ijsh.2015.9.10.20

A Novel Task Communication and Scheduling Algorithm for NoC-based MPSoC

[1]Weihua Zhang, [1]Gengxin Sun and [2]Sheng Bin

[1] International College of Qingdao University, Qingdao, China
[2]Software Technical College of Qingdao University, Qingdao, China
zhangweihua@qdu.edu.cn

Abstract

With the high performance demand, recent embedded systems are mostly based on NoC (Network-on-Chip) architectures, which would bring complex on-chip communication and scheduling problems. In this paper, a novel task scheduling algorithm which statically schedules both communication transactions and computation tasks onto heterogeneous NoC architectures under real-time constraints is presented. Our algorithm is capable of assigning tasks onto different processing elements (PE) automatically and scheduling their execution. We map tasks onto an 8× 8 NoC-based MPSoC to show that our scheduling algorithm leads to reduction in the total execution time, energy consumption. Experimental results show that for a multimedia application, more than 40% energy savings have been observed compared to the schedules generated by a standard scheduler.

Keywords: *NoC, MPSoC, heterogeneous architectures, task scheduling algorithm, energy consumption*

1. Introduction

The multiprocessor System-on-Chip (MPSoC) is a system-on-a-chip which uses multiple processors (multi-core), usually targeted for embedded applications. It is used by platforms that contain multiple, usually heterogeneous processing elements (PEs). The first appearance of MPSoC has been since the early 1990s, with the development of technics, the number of PEs inside a chip is increasing day after day, which led to the complexity of interconnection topologies and communication infrastructure between PEs. The communication infrastructure can be bus-based, point-to-point or Network-on-Chip (NoC)-based. Thereinto, the NoC is the most advantageous communication infrastructure as it has several advantages over others.

NoC is a communication subsystem on a chip, NoC technology applies networking theory and methods to on-chip communication and brings notable improvements over conventional bus and crossbar interconnections. NoC improves the scalability of MPSoC, and the power efficiency of complex MPSoC compared to other designs.

Although NoC can borrow concepts and techniques from the well-established domain of computer networking, it is impractical to blindly reuse features of computer networks and symmetric multiprocessors. In particular, NoC switches should be small, energy-efficient, and fast. The routing algorithms should be implemented by simple logic, and the number of data buffers should be minimal. Network topology and properties may be application-specific.

NoC is physically constituted of network adapters, routing nodes, and links allowing to connect the routing nodes [1-2]. The fundamentals NoC components are shown as Fig. 1.

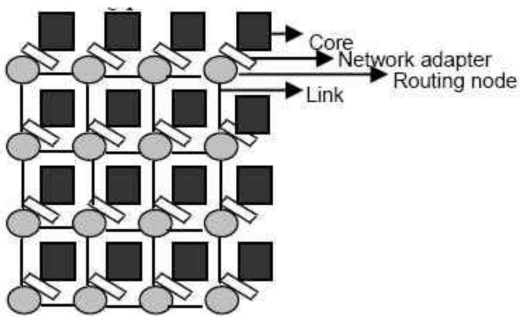

Figure 1. The Fundamentals NoC Components

Network adapter is the implementation of the interface connecting routing node to network, whose role is interfacing cores and network in a standardized way. A network adapter contains two sub blocs: core interface and network interface. Routers constitute the most important element in the NoC architecture. They allow the implementation of the routing strategy and the flow control algorithm.

Topology of NoC means interconnections between the different nodes, which can be unidirectional or bidirectional. It can be regular like “Spidergon”, “Mesh”, “Torus” and “Tree” or irregular. Topology of Noc has a direct impact on its performances [3]. The “Spidergon” topology is based on an even number of nodes, where each node is connected, rather than to its two neighbors by unidirectional links [4]. “Mesh” is a simple topology which allows access to all resources. It is characterized by its scalability [5]. “Torus” is proposed to reduce latencies of “Mesh” topology while retaining its simplicity [6]. The only difference between the two topologies “Mesh” and “Torus” is that for “Torus”, edge routers of one side are connected to edge routers of the opposite side. Each router in this topology is connected to 4 neighbor’s routers and to a core through input/output channels. In the “Tree” topology, the routers are placed on the nodes and the terminals on the leaves [7]. Each node is defined by its two coordinates (n, p): n to indicate the node level, p to indicate its position.

2. Related Work

How to mapping applications’ tasks onto MPSoC platform is the most important research field of MPSoC. It can be accomplished at either static (design-time) or dynamic (run-time) mapping techniques. In dynamic approaches, tasks are loaded into the system at run-time. In heterogeneous MPSoC, task migration is used to improve the performance by relocating a task from one PE to another PE. Task migration can also be used to insert a new task into the system at run-time [8].

In view of some optimization criteria, such as reducing energy consumption and reducing total execution time, tasks are mapped onto the MPSoC platform. Static mapping techniques [9-11] for NoC-based MPSoC are presented for a long time, but static mapping techniques are not suitable for configuration change over time and requires run-time mapping of applications. Run-time mapping techniques are required for adaptive systems, such as networking and multimedia applications, where the workloads are dynamic. For dynamic mapping techniques, Chou [12] propose a run-time mapping strategy that allows homogeneous MPSoC system to better respond to real-time changes. Mehran [13] present a Dynamic Spiral Mapping heuristic algorithm for 2D mesh topologies, which try to place the communicating tasks close to each other. Others run-time mapping techniques for mapping tasks onto heterogeneous MPSoC also have been developed. Smit [14] present a run-time task assignment algorithm that maps a task before all other tasks that need the scarce resources for heterogeneous multi-core architectures. Faruque [15] present a run-time agent based distributed application mapping technique for large MPSoC such as 32 × 32 systems. Nollet [16] present the task migration mechanism which uses task migration points as a point of reference for migrating a task from one PE to another. For the target MPSoC architecture which contains software and hardware PEs, Carvalho [17] present heuristic algorithm for run-time mapping of tasks in NoC-based heterogeneous MPSoC. In this architecture, each PE can support only one task, and tasks are mapped on the fly, according to the communication requests in the NoC links.

3. NoC-Based Target MPSoC Architecture

Our MPSoC is composed of 16 processing nodes connected by a 2-D Mesh Network on Chip as shown in Fig. 2.

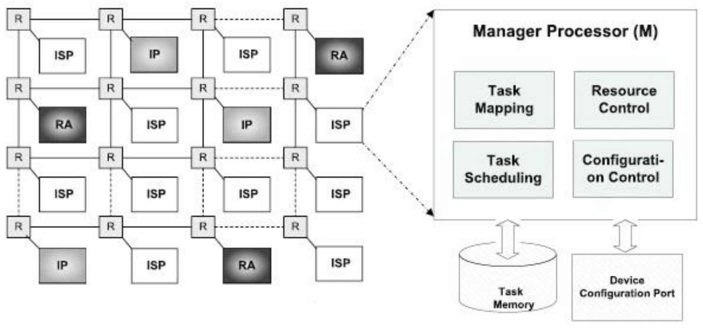

Figure 2. Our MPSoC Architecture

Each processing node was capable of supporting more than one either software tasks or hardware tasks. Software tasks execute in instruction set processors (ISP) and hardware tasks execute in reconfigurable areas (RA) or in dedicated IP-cores (IP). Among the available processing nodes, one of them is used as the Master Processor that is responsible for task mapping and scheduling. For propose of describe better our tasks scheduling algorithm, the abstract view of the NoC architecture can be shown in Figure 3.

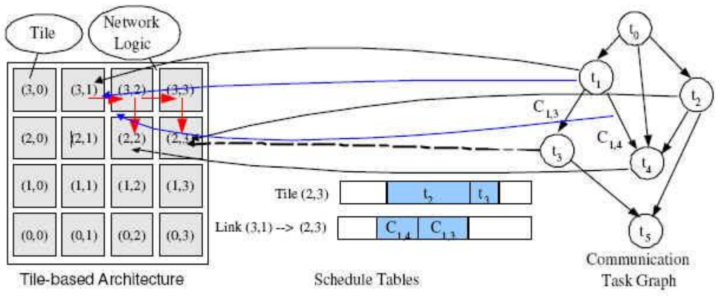

Figure 3. Abstract view of the NoC architecture

The PEs in the tiles of the NoC are heterogeneous. Because of the different performance, it would take different computation time and energy to execute the same task on different tiles.

For performance evaluation purposes, any task running on an embedded system can be described as a Communication Task Graph (CTG). The example can be seen right part of Figure 3. Given a CTG and a target MPSoC architecture, one important problem is how to schedule the tasks onto the target architecture. It is always called as the "scheduling problem" for NoC architectures.

4. Task Communication and Scheduling Algorithm for NoC-based MPSoC Architecture

The task scheduling problem is a traditional topic, but almost all previous work focuses on maximizing the performance. The algorithms developed this way are not suitable for embedded applications, because in which a common objective is to minimize the energy consumption of systems. Moreover, most previous work neglects the inter-processors communication during the scheduling process, in fact considering communication effects is critical for NoC architectures. In this paper, a novel algorithm, whose primary objective of scheduling is minimizing the energy consumption for NoC architecture, is proposed.

4.1. Energy Modeling for NoC Architecture

In energy model for NoC, the bit energy (E_{bit}) metric is defined as the energy consumed when one bit of data is transferred through the router. For NoC with buffers implemented by registers E_{bit} should be calculated as follows:

$$E_{bit} = E_{S_{bit}} + E_{L_{bit}} \tag{1}$$

Where $E_{S_{bit}}$ and $E_{L_{bit}}$ represent respectively the energy consumed on the switch and on the link between tiles.

By using Eq. (1), the average energy consumption for sending one bit of data from tile t_i to tile t_j can be analytically calculated as follows:

$$E_{bit}^{t_i,t_j} = n_{hops} \times E_{S_{bit}} + (n_{hops} - 1) \times E_{L_{bit}} \tag{2}$$

Where n_{hops} is the number of routers the bit passes on its way from t_i to tile t_j .

For 2D mesh networks with minimal routing, Eq. (2) is determined by the Manhattan distance between t_i and t_j .In this paper, the energy model described in Eq. (1) is chosen as it provides an efficient approximation for the NoC architectures under consideration with reasonable accuracy.

4.2. Formulation Definition of Scheduling Algorithm

Given the CTG of an application and a target NoC architecture, a feasible schedule such that the application energy consumption is minimized under performance constraints need to be proposed.

For scheduling algorithm, each task t_i should firstly determine that which PE in the NoC should it be scheduled to, and the time slot when the task will be executed on the PE. Then, the time slot for all the communication transactions in the application should be determined. For example in Figure 3, if t_2 and t_3 are both assigned to the same PE location (2,3), our algorithm should determine which one should be executed first, if tasks t_1, t_3 and t_4 are assigned to PEs respectively at locations (3,1), (2,2) and (2,3), then our algorithm need to determine the execution order of communication transaction $c_{1,3}$ and $c_{1,4}$ on the link from PE (3,1) to PE (3,2).

To formulate better our algorithm, following terms are defined:

Definition 1: A Communication Task Graph (CTG) $G(T,C)$ is a directed acyclic graph, where each node represents a computational module of the application referred to as a task $t_i \in T$.

Each directed arc $c_{i,j} \in C$ represents the communication dependency between t_i and t_j.The direction from t_i to t_j indicates that the task t_j cannot start before t_i is finished. Each $c_{i,j}$ associated with a nonzero value $v(c_{i,j})$, which stands for the communication bits from c_i to c_j.It denotes that t_j can start only after t_i has finished and transferred $v(c_{i,j})$ bits of date to task t_j.

Definition 2: An Architecture Characterization Graph (ACG) $G(P,R)$ is a directed graph, where each node $p_i \in P$ represents one PE in the architecture, and each directed arc $r_{i,j} \in R$ represents the route from p_i to p_j. Each $r_{i,j}$ associated with it metric $e(r_{i,j})$ stands for the average energy consumption of sending one bit of data from p_i to p_j.

Definition 3: Communication transaction $c_{i1,j1}$ is said to be compatible with another communication transaction $c_{i2,j2}$ if and only if their execution times do not overlap or their routing paths.

Definition 4: Task t_i is said to be compatible with task t_j if and only if their execution times do not overlap or they are assigned onto different PEs.

Through above definitions, our scheduling algorithm for heterogeneous NoC architectures under some constraints can be formulated as follows:

$$\min\{Energy = \sum_{\forall t_i} e^{i}_{\mathrm{M}(t_i)} + \sum_{\forall c_{i,j}} v(c_{i,j}) \times e(r_{\mathrm{M}(t_i),\mathrm{M}(t_j)})\} \tag{3}$$

Eq. (2) represents that given a CTG and an ACG, a mapping function $\mathrm{M}(\ )$ from tasks (T) to PEs (P) with a start time for each task and communication transactions.

Since finding optimal schedule for a multi-processor system that consumes the minimum energy is a NP-hard problem, a heuristic algorithm which is capable of finding satisfactory solutions with reasonable short computation time would be proposed in the following.

4.3. Energy-Aware Task Scheduling Algorithm

Our energy-aware scheduling algorithm is based on slack-budgeting which allocates more slack to those tasks whose mapping onto PEs has a larger impact on energy consumption; our proposed algorithm can be divided into the following three steps during scheduling:

Step 1. Budget slack allocation for each task

In this step, firstly for each task ti, there are three metrics need to be calculated: VAR_{e_i} and VAR_{r_i} are individually the variance of the energy consumption and the execution time of t_i on different PEs. M_{t_i} is the mean execution time of task t_i on different PEs.

Task t_i is assigned a weight $W_{t_i} = VAR_{e_i} \times VAR_{r_i}$ by the calculated VAR_{e_i} and VAR_{r_i}, the higher this weight, the higher the priority the task should have in selecting the PE. We can calculate the slack for different paths, and then allocate the slack to different tasks based on their respective weights by M_{t_i} of different tasks. With these weights, the budgeted deadline (BD) of each task is calculated.

Step 2. Task scheduling

Firstly, generating the Ready Tasks List (RTL), that is, the tasks whose precedent tasks have already been scheduled. Let $F(i,k)$ be the earliest finish time of task t_i if it is assigned to PE p_k .For each combination of task $t_i \in RTL$ and PE $p_k \in P$, calculate its $F(i,k)$ as:

$$F(i,k) = DRT(i,k) + r_k^i \tag{4}$$

where r_k^i is the execution time of task t_i on PE p_k. DRT represents the data ready time, which is defined as the latest arrival time of all the receiving communication transactions of the corresponding task.

For each task t_i in the RTL, calculate its metric $\min_{F(i)}$:

$$\min_{F(i)} = \min\{F(i,k)\} \quad \forall k \tag{5}$$

If there are tasks which satisfy $\min_{F(i)} \geq BD_i$, then the task which has the largest value of $\min_{F(i)} - BD_i$ is selected and assign it to the PE that corresponds to $\min_{F(i)}$ .On the other hand, if all the tasks in the RTL satisfies $\min_{F(i)} < BD_i$, then the following way to select the next task to be scheduled would be used.

First, for each t_i in the RTL, a list L_i is generated which contains a list of PEs. Each PE p_k in that list must satisfy the condition $\min_{F(i)} \leq BD_i$. Thus, this list gives all the PEs that if task t_i is scheduled onto any of the tiles, its deadline could be satisfied. Next, for each task t_i, let E_1^i to be the minimum energy consumption if it is scheduled onto a tile in the list L_i. And E_2^i is the second minimum energy consumption, a metric $\delta_E^i = E_2^i - E_1^i$ is calculated for each task. Finally, the task in the RTL which has the largest δ_E and the task is assigned to the PE which leads to E_1^i energy consumption is selected.

Step 3. Revise procedure

Since the optimization objective for our scheduling algorithm is the energy consumption minimization, there may be occasional deadline misses if just relying on aforementioned two steps. So a revise procedure which can be used to fix the missed deadlines is presented. The revise procedure has two main components: local task swapping (LTS) and global task migration (GTM). The relationship between LTS and GTM is shown in Figure 4.

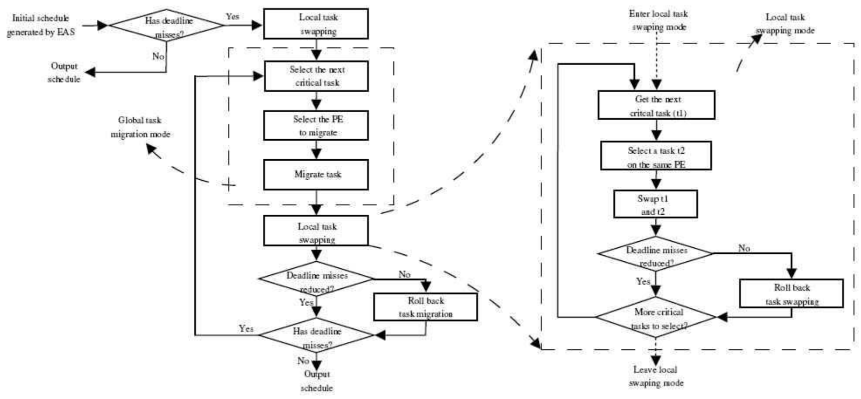

Figure 4. Flow of revise procedure

In the LTS mode, the procedure would pick up each critical task and swap its order of execution with other non-critical tasks assigned to the same PE. It will let the critical tasks execute earlier than non-critical tasks so that the deadline misses can be reduced. If one PE is so heavily loaded that no matter how one change the execution order of the tasks on that PE, the deadline misses can not be fixed by LTS. In this case, GTM is used to identify a critical task and migrate it to other PEs. To reduce its impact on increase of energy, the destinations PEs are tried in the increasing order of the execution and communication energy. If the migration of the task reduces deadline misses, it would be accepted. Otherwise, the next task would be selected.

5. Experiment and Analysis

For evaluating effectiveness of our algorithm, several experiments on random task sets and a set of generic multimedia systems are implemented.

5.1. Random Benchmarks Experiment Results

Two categories of random benchmarks were generated using TGFF [18]. Each category contains 10 randomly generated benchmarks and there are around 500 tasks with about 1000 communication transactions in each benchmark.

Both of the two categories of benchmarks are to be scheduled onto a 8×8 heterogeneous NoC. For evaluating robustness of our algorithm, various parameters are used in TGFF to generate benchmarks with different topologies and task/communication distributions. To evaluate our algorithm, we implemented a standard Earliest Deadline First (EDF) scheduler and compared with two versions of our algorithm EAS and EAS-base (with or without revise procedure) in terms of energy figures and deadline misses.

The comparison of energy consumption of the scheduling algorithms generated for two categories benchmarks is shown respectively in Figure 5 and Figure 6.

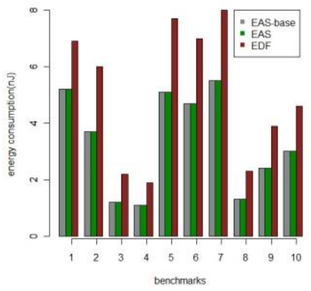

Figure 1. Comparison by Using Category I Benchmarks

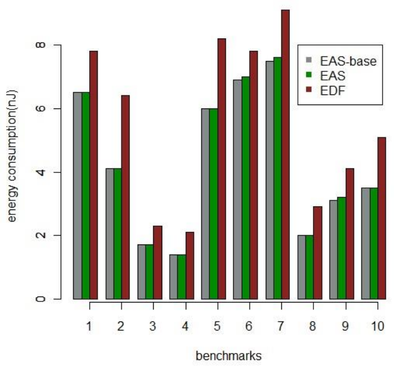

Figure 2. Comparison Using Category II Benchmarks

As shown in Figure 5 and Figure 6, the energy generated by EDF consume, on average, 55% and 39% more energy compared to those generated by our algorithms, for category I and category II of benchmarks respectively.

5.2. Multimedia System Experiment Results

For evaluating our algorithm for real-world applications, a set of generic Multimedia System Benchmarks (MSB) is applied. The experimental system which we consider consists of an MP3/H263 audio/video (A/V) encoder pair. For seeing the trade-off between the energy savings and the performance constraints, the following experiment on the integrated MSB application is performed. Starting with a given encoding rate and decoding rate, we slowly increase encoding rate and decoding rate and then observe its impact on the energy savings, which are shown in Figure 7.

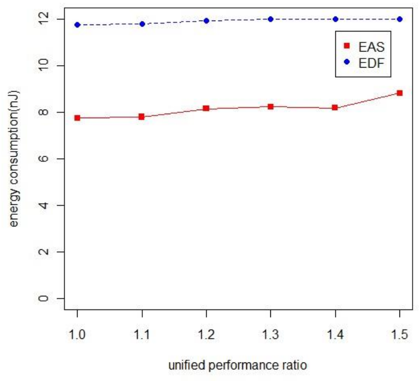

Figure 3. Performance and Energy Tradeoff

From Figure 7, we can note that as the performance requirements become more stringent, the schedule generated by EAS consumes more energy consumption as the scheduler has less flexibility in assigning and ordering the execution of tasks/communication.

6. Conclusion

In this paper, an efficient energy-aware scheduling algorithm which statically schedules tasks onto heterogeneous NoC architectures is proposed. Although our experiments have been evaluated on the architectures interconnected by 2D mesh networks, our algorithm can be adapted to other regular architectures with different network topologies. We have investigated different scenarios depending on performance metrics and they show that improvement in the energy savings compared to the traditional scheduler.

References

[1] T. Bjerregaard, "A Survey of Research and Practices of Network on-Chip", ACM Computing Surveys, vol. 3, no. 38, **(2006)**.
[2] N. Concer, "Equalized, a Novel Routing Algorithm For The Spidergon Network On Chip. Proceedings of Design", Automation and Test in Europe Conference and Exhibition, ACM, **(2009)**.
[3] F. Karim, "An interconnect architecture for networking systems on chips", Micro IEEE, vol. 5, no. 22, **(2002)**.
[4] M. Moadeli, "An Analytical Performance Model for the Spidergon NoC", Proceedings of 21st International Conference on Advanced Networking and Applications, **(2007)**.
[5] F. Karim, M. Root and C. Mesh, "Interconnection Architectures for Network-on-Chip Systems", World Academy of Science, Engineering and Technology, vol. 54, **(2009)**.
[6] M. Aghatabar, "An Empirical Investigation of Mesh and Torus NoC Topologies under Different Routing Algorithms and Traffic Models", Proceedings of the 10th Digital System Design Architectures, Methods and Tools Conference, **(2007)**.
[7] P. Pande, "Design of a switch for network on chip applications", Proceedings of the 2003 IEEE International Symposium on Circuits and Systems, **(2003)**.
[8] H. Kalte and M. Porrmann, "Context saving and restoring for multitasking in reconfigurable systems", Proceedings of FPL, **(2005)**.
[9] V. Nollet, "Run-time management of a mpsoc containing fpga fabric tiles", IEEE Trans. VLSI Syst. vol. 1, no. 16, **(2008)**.
[10] C. Marcon, "Time and energy efficient mapping of embedded applications onto nocs", Proceedings of ASP-DAC, **(2005)**.
[11] M. Ruggiero, "A fast and accurate technique for mapping parallel applications on stream-oriented mpsoc platforms with communication awareness", Int. J. Parallel Progr. vol. 1, no. 36, **(2008)**.
[12] C. L. Chou and R. Marculescu, "User-aware dynamic task allocation in networks-onchip", Proceedings of DATE, **(2008)**.
[13] A. Mehran, "A heuristic dynamic spiral mapping algorithm for network on chip", IEICE Electron. Exp. vol. 5, no. 13, **(2008)**.
[14] G. J. Smit, "Multi-core architectures and streaming applications", Proceedings of International Workshop on System Level Interconnect Prediction. **(2008)**.
[15] M. A. Faruque, "Adam, run-time agent-based distributed application mapping for on-chip communication", Proceedings of the DAC, **(2008)**.
[16] V. Nollet, "Centralized run-time resource management in a network-onchip containing reconfigurable hardware tiles", Proceedings of the DATE. **(2005)**.
[17] E. Carvalho and F. Moraes, "Congestion-aware task mapping in heterogeneous mpsocs", Proceedings of International Symposium on SoC. **(2008)**.
[18] R. P. Dick, D. L. Rhodes and W. Wolf, "TGFF, task graphs for free", Proceedings of International Workshop on Hardware/Software Codesign, **(1998)**.

Authors

Weihua Zhang, is currently an Associate Professor in the School of Computer Science and Engineering at Qingdao University. His main research interests include complex networks, web information retrieval and data mining.

Gengxin Sun, received his Ph.D. degree in Computer Science from Qingdao University, China in 2013. He is currently an Associate Professor in the School of Computer Science and Engineering at Qingdao University. His main research interests include embedded system, operating system, complex networks, web information retrieval and data mining.

Sheng Bin, received her Ph.D. degree in Computer Science from Shandong University of Science and Technology, China in 2009. She is currently a lecturer in the School of Software Technology at Qingdao University, China. Her main research interests include embedded system, operating system, complex networks, cloud computing and data mining.