International Journal of Future Generation Communication and Networking
Vol. 9, No. 9 (2016), pp. 181-190
http://dx.doi.org/10.14257/ijfgcn.2016.9.9.16

A New Static Data Flow Clustering Algorithm for Task Scheduling of Irregular Mesh in NoCs Based on Complex Networks

Yue Liu[1], MengMeng Cao[1] and Kong Jie[2]
[1]*Institute of Information Engineering, Kaifeng University, Kaifeng 475004, China*
[2]*Naval Academy of Armament, Beijing 100091, China*
[1]*852278078@foxmail.com,* [2]*13501275251@139.com*

Abstract

The majority of recent embedded systems are based on MPSoCs (Multi-Processors System on Chip) architectures. The topologies and the interconnections inside multi processors almost adopt NoCs (Networks on Chip) whose topology and task scheduling algorithm have a direct impact on its performances. In this paper, by using static data flow, a task scheduling algorithm which would automatically assign the application tasks onto different processors is proposed based on complex network. The goal of our algorithm is to replace the static data flow subnetwork by a single dynamic data flow actor such that the global performance in terms of latency and throughput is optimized. Through complex network, it greatly enhances the power of our algorithm in terms of avoiding deadlock, saving energy and providing for integration with more general models of computation. Experimental results show up to 60% performance improvement for real-world examples.

Keywords*: NoC, MPSoC, static data flow, task scheduling algorithm, complex network*

1. Introduction

Multi-Processor System on Chips (MPSoCs) are becoming more and more important as implementation platform for embedded system. However, the high parallelism of multiple processors makes programming of MPSoCs a challenging task. There are different MPSoCs communication topologies and interconnection strategies inside multi-processors system on chip, namely point to point, buses and NOCs [1-2].

Network on chip (NoC)[3-6] is a communication subsystem on an integrated circuit (commonly called a "chip"), typically between intellectual property (IP) cores in a system on a chip (SoC). NoCs can span synchronous and asynchronous clock domains or use unclocked asynchronous logic. 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 SoCs, and the power efficiency of complex SoCs compared to other designs.

Topology of NoC can be regular like“Spidergon”, “Mesh”, “Torus” and “Tree” or irregular. Topology of NoC has a direct impact on its performances [4]. Many regular topologies more or less inefficient have appeared. Irregular topologies are considered more realistic than regular ones with less constraints on network form [7]. Theses irregular topologies are always obtained by mixing different regular forms with hierarchical, hybrid or asymmetric way [4].

The type of topology (regular or irregular) goes with the scope of the MPSOC and nature of the cores used [8]. The regular topologies are suitable for general purpose architectures with homogeneous cores. Under these conditions (general architecture and homogeneous cores) regular topologies lead to a regular and predictable “layout”s. Instead, irregular topologies are more appropriate for MPSoC’s specific applications with heterogeneous cores and memories and having different sizes. For such systems, the irregular architectures are more efficient than regular in term of energy consumption, area and performance.

Mesh is a simple topology which allows access to all resources. Topology of a n*n 2D Mesh is defined as follows:

(1) R = n*n routers;

(2) Each router (except those on the sides) is connected to 4 neighbors' routers and to a core (a processor or a memory) via its input/output channels;

(3) An input/output channel consists of two point to point unidirectional communications between two routers or between a router and a resource. The number of communication channels of an n*n 2D Mesh is $C = 3n^2 - 2n$.

The topology of 4*4 Mesh is shown as Figure 1.

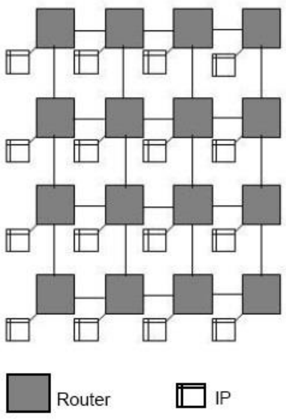

Figure 1. Mesh 4*4 Topology

Some researches [9-11] show that irregular mesh networks are more beneficial to some embedded systems than regular ones. Some typical topologies of irregular mesh are shown as Figure 2.

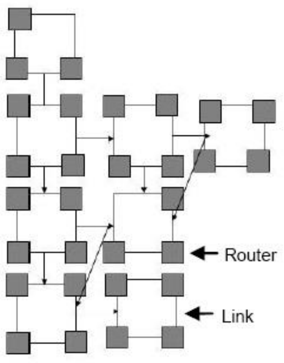

Figure 2. Topology of Irregular Mesh

Mapping applications' tasks onto NoC-based MPSoC platform have become the most important research field of MPSoC. In common approaches, tasks are loaded into the system at run-time. For dynamic mapping techniques, Smit [12] present a run-time task assignment algorithm that maps a task before all other tasks that need the scarce resources for heterogeneous multi-processor architectures. Faruque[13] present a run-time agent based distributed application mapping technique for large NoC-based MPSoC such as 32*32 systems. Nollet [14] present the task migration mechanism which uses task migration points as a point of reference for migrating a task from one processor to another. But there is still only little work on defining the right task scheduling model for MPSoCs, especially for irregular mesh NoC-based MPSoC.

2. Related Work

Dataflow models [15-16] of computation have been extensively studied in task scheduling algorithm. Static data flow (SDF) model is commonly used in task scheduling analyses. Despite extensive work on task scheduling from SDF models, these models can capture the parallelism that is available in a function and thus may be good candidates to represent function behavior. There is little existing work that addresses compositional representations for such models. Geilen [17] proposes abstraction methods that reduce the size of SDF graphs, thus facilitating throughput and latency analysis.

Static data flow model include static dependence information, which is computed using alias analysis algorithms. In the worst case, overly conservative analysis yields completely connected dependence networks. Since there is only one complete graph given a number of vertices, this poor analysis effectively reduces the amount of information about the function. On the other hand, Dynamic data flow networks can be more detailed than Static data flow networks because they represent only observed dependence edges instead of potential dependence edges. Further, other dynamic information can be annotated in these networks.

Dataflow models with deterministic actors, such as Kahn Process Networks [18] and their various subclasses, including SDF, are compositional at the semantic level. Actors can be given semantics as continuous functions on streams, and such functions are closed by composition. However, a dynamic schedule may degrade the performance by introducing scheduling overheads even in the schedules of static data flow networks. To permit the generation of an efficient schedule, a remedy could be the replacement of the static data flow graph by a single actor, for example, clustering all static data flow actors into a new actor. Unfortunately, existing algorithms might result in infeasible schedules or greatly restrict the clustering design space.

In this paper, a new clustering approach for static data flow networks connected to dynamic data flow networks based on complex network would be proposed. In contrast to prior work, a more general actor can be generated which can have dynamic behavior. It allows expressing of a quasi-static schedule for the static data flow graph in order to avoid deadlocks which might occur when restricting to static schedules only. The quasi-static schedule can be automatically derived by our clustering approach and expressed in form of a finite state machine, which can be easily integrated into a dynamic schedule of all remaining actors mapped onto the same processor while reducing the overall scheduling overhead.

3. Methodology and Problem Definition

In this section, the problem that the paper is dedicated to is formally defined and introduces the necessary mathematical notations of methodology.

Complex network is a graph (network) with non-trivial topological features that do not occur in lattices or random networks but often occur in real networks. We can use complex network to describe data flow model.

Definition 1 (Data Flow Network): A data flow network is a directed network $g = (A, C, L)$ containing a set of actors A (vertices) and a set of channels $C \subseteq A \times A$ represented by the edges of the network. Additionally, the data flow network contains an initial fill level function $L : C \to N_0$ (N_0 denotes the set of non-negative integers) which associates with each channel $(a_{src}, a_{dest}) \in C$ its number of initial tokens.

Furthermore, we introduce a notion of so called actor input ports $i \in I$ and so called actor output ports $o \in O$ where $a.I \subseteq I$ and $a.O \subseteq O$ are the set of input and output ports of the actor a, the set of subnetwork input ports $g_r.I$ and subnetwork output ports $g_r.O$ are defined, where $g_r.I$ and $g_r.O$ are actor input and output ports connected to the subnetworks.

By mapping an application modeled by a data flow network $g = (A, C, L)$ onto an MPSoC, subnetworks of the data flow network are bound to the distinguished programmable processors. Communications between the subnetworks are bound to the on chip buses or networks on chip. The proposed clustering approach computes for a static data flow subnetwork g_r induced by the set of static data flow actors $A_s \subseteq g$. A bound onto a single processor of an MPSoC a composite actor a_r which replaces this subnetwork g_r.This composite actor implements a data flow schedule which can vary from static over quasi-static to dynamic.

Definition 2 (Clustering): Given a data flow network g and a static data flow subnetwork g_r induced by all static data flow actors $a \in A_S \subseteq g$ .Clustering replaces g_r by a single actor a_r, called composite actor, implementing a data flow schedule for the actors $a \in A_S$, resulting in a new data flow network g', $g'.A = g.A + \{a_r\} - A_S$, and $g'.C = g.C + C_a - C_S$, where $C_S = \{c = (a_1, a_2) \in g.C \mid a_1 \in A_S \vee a_2 \in A_S\}$ and C_a is the set of edges connecting a_r with the remaining DDF actors $g.A - A_S$, where

$$C_a = \{(a_{src}, a_{\sin k}) \in g'.A \times g'.A \mid (a_{src} = a_r \Rightarrow \exists (a', a_{\sin k}) \in g.C : a' \in A_S) \vee$$
$$(a_{\sin k} = a_r \Rightarrow \exists (a_{src}, a') \in g.C : a' \in A_S)\}$$

Definition 3 (Cluster FSM): The cluster FSM of a composite actor a_r is a tuple $m = (Q, q_0, T, N, R)$ containing a finite set of states Q and an initial state $q_0 \in Q$,a finite set of transitions $(q_{src}, q_{dest}) \in T \subseteq Q \times Q$, a guard function $N : T \to N_0^{|g_r.I|}$ specifying the precondition for the number of tokens required on each channel connected to the subnetwork input ports $g_r.I$ to execute a transition, and finally an action function $R : T \to g_r.A^*$ encoding a static scheduling sequence for the actors $a \in g_r.A$ of the subnetwork.

With above notations, the subnetwork $g_{2,r}$ can be replaced by a composite actor $a_{2,r}$ as depicted in Figure 3.

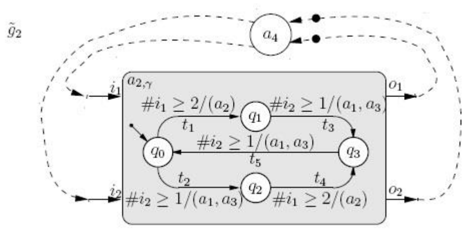

Figure 3. Replacing Subnetwork $g_{2,r}$ by the Composite Actor $a_{2,r}$

We use $\#i$ to denote the number of available tokens on the channel connected to the actor input port i. From Figure 3 we can see that two transitions t_1 and t_2 are leaving the start state q_0. t_1 requires at least two tokens on input port i_1 denoted by the precondition

$\#i_1 \geq 2$ and executes the static schedule (a_2) whereas t_2 requires at least one input token on input port i_2 denoted by $\#i_2 \geq 1$ and executes the static schedule (a_1, a_3).

Using those definitions, we can represent static schedules, quasi-static schedules and dynamic schedules.

4. Static Data Flow Clustering Algorithm Based on Complex Network

In the section, a methodical way to construct the cluster FSM m as defined in Definition 3 that represents a quasi-static schedule for a given static data flow subnetwork g_r is presented. The algorithm is that each output $o \in g_r.O$ of g_r might have a feedback by other data flow actors to each input $i \in g_r.I$. Because any produced token on an output $o \in g_r.O$ might cause through these feedback loops the activation of an actor $a \in g_r.A$ in the same subnetwork. In particular, postponing the production of an output token may result in a deadlock of the entire system. Hence, the quasi-static schedule determined by our clustering algorithm guarantees the production of a maximum number of output tokens from the consumption of a minimal number of input tokens. However, our clustering algorithm requires that tokens produced by an output port depend on all input ports. Otherwise an unbounded accumulation of tokens may happen. Therefore, we need to define the following clustering condition:

Definition 4 (Cluster Condition): A static data flow subnetwork g_r can be clustered by the given algorithm if the subnetwork disregarding its inputs and outputs is deadlock free itself and for each pair of actors (a_{src}, a_{dest}) possessing a subnetwork input and output port there exists a directed path $p \in g_r.C^*$ from actor a_{src} to actor a_{dest}.

Our clustering algorithm can be divided into three steps: Preprocessing, Compute the set of input/output states and Construct the cluster FSM. The step 1 is used for computing some termination criteria for step 2 and step 3. Especially, the number of firings of each actor to bring the cluster back into its initial state as well as the number of consumed and produced tokens by these firings would be computed.

Step 1.1 Compute the repetition vector $r_{\min,g_r}$ for subnetwork g_r, a positive integer $r_{\min,g_r}(a)$ is assigned to each actor $a \in g_r.A$. in the subnetwork denoting the minimal number of firings of a to return g_r back to its initial state.

Step 1.2 Compute the input/output repetition vector $n_{\min,g_r}$ which assigns to each input $i \in g_r.I$ and each output $o \in g_r.O$ the number of consumed tokens $n_{\min,g_r}(i)$ or produced token $n_{\min,g_r}(o)$ by firing each actor $a \in g_r.A$ exactly $r_{\min,g_r}(a)$ times.

In order to avoid actor feedback loop, it is required that the resulting quasi-static schedule always produces a maximal number of output tokens with a minimal number of input tokens. Each end point of such a production is marked by an input/output state of the subnetwork. So the following three steps to determine the input/output states are proposed:

Step 2.1 Compute for each output port o the input/output dependency tuples encoding the minimal numbers of consumed tokens on the input ports to produce n tokens on o . For this propose, an input/output dependency function dep_{g_r} is formally defined.

Definition 5 (Input/Output Dependency Function): For given a subnetwork g_r, the input/output dependency function dep_{g_r} : $g_r.O \times N_0 \to N_0^{|g_r.I|}$ is a function that associates with a cluster g_r, for each subnetwork output port $o \in g_r.O$, and for a requested number of tokens $n \in N_0$ a vector of minimal number of input tokens $(n_{i_1}, n_{i_2}, ...n_{i_{|g_r.I|}})$ consumed on each subnetwork input port $i \in g_r.I$ to produce the requested number n of tokens on the output port o.

Step 2.2 Given dep_{g_r}, the so-called input/output dependency states Q_{io} can be calculated. Each input/output state $(n_{i_1}, n_{i_2}, ...n_{i_{|g_r.I|}}, n_{o_1}, n_{o_2}, ...n_{o_{|g_r.O|}})$ is a possible state of execution of the static actors $g_r.A$ in a self-scheduled execution and is additionally a constraint to provide the maximum possible number of output actor firings for a minimum number of required input actor firings. Additionally, add the null input/output state $(0,0,...0)$ which may be missing if the subnetwork can produce output tokens without consuming any inputs. The input/output state set can be defined as follows:

Definition 6 (Input/Output State Set):

$$Q_{io} = \{(0,0,...0)\} \cup \{(n_{i_1}, n_{i_2}, ...n_{i_{|g_r.I|}}, n_{o_1}, n_{o_2}, ...n_{o_{|g_r.O|}}) \mid$$
$$(n_{i_1}, n_{i_2}, ...n_{i_{|g_r.I|}}) \in dep_{g_r} \wedge \forall o \in g_r.O : n_o = \max($$
$$\{n \in N_0 \mid dep_{g_r}(o,n) \le (n_{i_1}, n_{i_2}, ...n_{i_{|g_r.I|}})\})\}$$

Each input/output dependency state is a point in the n-dimensional Euclidean vector space $N_0^{|A_I \cup A_O|}$ where each dimension represents the number of firings of a subnetwork input or output actor, respectively. The initial state is then trivially the all zero vector representing the fact that in the beginning no actor of the subnetwork has fired.

Definition 7 (Cluster State Space): The state space Q of a cluster is defined as the least fixpoint $Q = lfp(Q' = \{\max(q_1,q_2)\}q_1,q_2 \in Q'\} \cup Q' \cup Q_{io})$ which enlarges Q' starting from Q_{io} by adding the pointwise maximums of all pairs of input/output states from Q' until no more new states are created.

After computing the input/output states, the cluster FSM can be constructed by ordering the input/output states and computing the transitions between input/output states.

Step 3.1 Compute the partial order $n_1 \le n_2$ on $lfp(Q')$ where $n_1 \le n_2$ *iff* $\forall p \in g_r.I \cup g_r.O : n_1(p) \le n_2(p)$.

Step 3.2 Compute the state set Q of the cluster FSM m by generating a state $q \in Q$ for each input/output state n not greater than or equal to the input/output repetition vector.

Step 3.3 Compute the transition set T of the cluster FSM m by generating a transition $t \in T$ for each tightly ordered pair of input/output states.

Step 3.4 Compute the guard function N of the cluster FSM m by generating a guard function value $N(t)$ encoding the minimal number of tokens on each subnetwork input port to enable the transition t from each tightly ordered pair of input/output states.

Step 3.5 Compute the action function $R(t)$ of the cluster FSM m.

Finally, for each transition on the basis of the partial repetition vector, a single processor schedule is computed by a version of the scheduling algorithm to support partial repetition vectors. This schedule is assigned to $R(t)$ and is one of the schedule phases of the resulting composite actor replacing the static data flow subnetwork g_r.

5. Results

In order to illustrate the benefits of our clustering algorithm, we have applied it and synthetic dataflow networks to an mp3 decoder, which works on irregular mesh of NoCs. In order to evaluate the methodology more thoroughly, we also applied it to generate four different SDF networks G2, G3, G5 and G7 with 60 actors each. All four networks have the same properties except the degree of connectivity. The average input/output degree of each actor in G2 is 2, in G3 it is 3, in G5 it is 5, and in G7 it is 7. Based on these four networks, test cases G2-ND, G3-ND, G5-ND and G7-ND have been constructed randomly by marking a variable number ND of actors as dynamic. For each test case, the settings ND = 6, 12, 18, . . ., 54, 60 have been considered, and for each setting, ten instances have been generated. The ten different instances per setting are used to compute average results in the experiments.

The first observation is that the size of the clusters satisfying the cluster condition depends not only on the number of dynamic actors in the network but also on the connectivity degree. From Figure 4. we can see that a lower degree of connectivity leads to a smaller number of static actors that can be clustered.

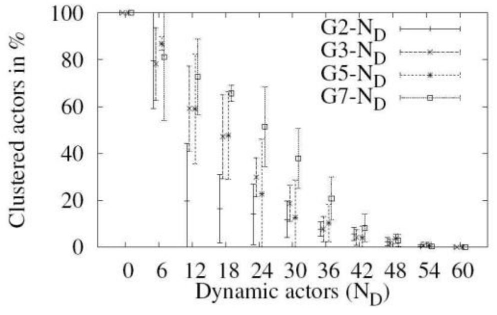

Figure 4. Average Number and Standard Deviation of Static Actors that can be clustered

Then, our proposed clustering approach was applied for computing quasi static schedules and was used during software synthesis. Our proposed clustering approach has another advantage when comparing it with the FSM approach on the basis of the required compile time. In these results, compile time of our clustering approach is always a small fraction of compile time of the FSM approach, which is shown as Figure 5. This significantly extends the area of applicability of clustering during embedded software synthesis.

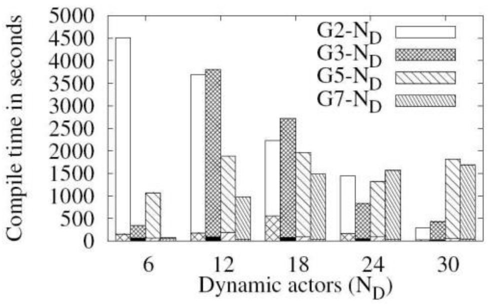

Figure 5. Average Compile Time of the FSM Approach and our Approach

In embedded systems, the available amount of memory is typically constrained. Hence, we tested both clustering approaches with given code size constraints which have been set to 150% of the memory requirements of the previously computed dynamic schedules. The average speedup is shown in Figure 6. It can be seen that in all cases our proposed approach are better than the FSM approach.

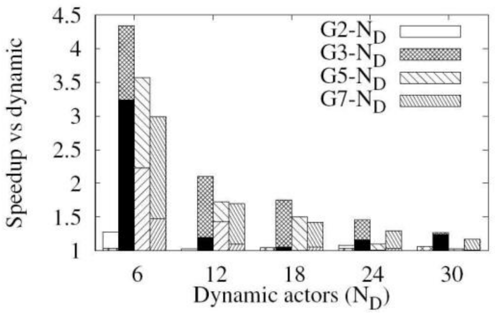

Figure 6. Average Speedup respecting Code Size Constraints

6. Conclusion

In this paper, we presented a clustering approach for static data flow subnetworks based on complex network. The proposed clustering algorithm computes a quasi-static schedule and reduces the scheduling overhead for one processor of MPSoC. Through our clustering approach, the scheduling of these subnetworks can be coordinated with enclosing system representations in a way that systematically exploits the predictability and efficiency of the static data flow model. Future work will focus on optimal clustering techniques, i.e., to generate an efficient representation of the cluster FSM state space for synthesis and to identify which actors should be clustered in order to minimize the scheduling overhead.

References

[1] P. Pande, “Evaluation of MP-SoC Interconnect Architectures: A Case Study”, Proceedings of the 4th IWSOC, **(2004)**.
[2] H. Lee, “On-Chip Communication Architecture Exploration: A Quantitative Evaluation of Point-to-Point, Bus, and Network-on-Chip Approaches”, ACM Transactions on Design Automation of ElectronicSystems, vol. 12, no. 3 **(2007)**.
[3] L. Bennini, “Networks on chips: A new paradigm for component based MPSoC design”, American Scientific Publishers, vol. 11, no. 9, **(2004)**.
[4] T. Bjerregaard, “A Survey of Research and Practices of Networkon-Chip”, ACM Computing Surveys, vol. 7, no. 38, **(2006)**.
[5] F. Karim, “An interconnect architecture for networking systems on chips”, Micro IEEE, vol. 5, no. 22, **(2002)**.
[6] A. Zitouni, “A Generic and Extensible Spidergon NoC”, World academy of science, engineering and technonogy, vol. 7, no. 31, **(2007)**.
[7] L. Bononi, “Simulation and Analysis of Network on Chip Architectures: Ring, Spidergon and 2D Mesh”, Proceedings of the conference on Design, automation and test in Europe: Designers’ forum, **(2006)**.
[8] K. Srinivasan, “An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks”, Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design, **(2005)**.
[9] P. Meloni, “Routing Aware Switch Hardware Customization for Networks on Chips”, Proceedings of Nano-Networks, **(2006)**.
[10] D. Bertozzi, “NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip”, IEEE transactions on parallel and distributed systems, vol. 2, no. 16, **(2005)**.
[11] D. Bertozzi, MinRoot and CMesh, “Interconnection Architectures for Network-on-Chip Systems”, World Academy of Science, Engineering and Technology, vol. 7, no. 54, **(2009)**.
[12] G. J. Smit, “Multi-core architectures and streaming applications”, Proceedings of International Workshop on System Level Interconnect Prediction. **(2008)**.
[13] M. A. Faruque, “Adam: run-time agent-based distributed application mapping for on-chip communication”, Proceedings of the DAC, **(2008).**
[14] V. Nollet, “Centralized run-time resource management in a network-onchip containing reconfigurable hardware tiles”, Proceedings of the DATE, **(2005)**.
[15] A. Edwards and O. Tardieu, “SHIM: A Deterministic Model for Heterogeneous Embedded Systems”, Proceedings of the EMSOFT, **(2005)**.

[16] A. A. Jerraya, A. Bouchhim and F. P´etrot, "Programming Models and HW-SW Interfaces Abstraction for Multi-Processor SoC", Proceedings of the DAC, **(2006)**.
[17] M. C. W. Geilen, "Reduction of Synchronous Dataflow Graphs", Proceedings of the Design Automation Conference, **(2009)**.
[18] G. Kahn, "The semantics of a simple language for parallel programming", Proceedings of the IFIP Congress, **(1974)**.

Authors

Yue Liu, he received his Master degree of Henan University in 2010.He is a lecturer at the Institute of Information Engineering. His main research interests include complex networks, embedded system.

Meng Meng Cao, she received her Master degree in software engineering from Tongji University in 2008. Her main research interests include embedded system, complex networks, artificial intelligence and electrical engineering.

Kong Jie, he was born in 1965. He received the M.E. degree in software engineering from Huazhong University of Science and Technology, Wuhan, China in 2007. He is currently a senior engineer with the Naval Academy of Armament, Beijing, China. His research interests are embedded system, software engineering, database technology and computer network technology.