Journal of Systems Architecture 57 (2011) 840–849

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Resource allocation robustness in multi-core embedded systems with inaccurate information

Jiayin Li [a], Zhong Ming [b,*], Meikang Qiu [a], Gang Quan [c], Xiao Qin [d], Tianzhou Chen [e]

[a] Dept. of Elec. and Comp. Engr., University of Kentucky, Lexington, KY 40506, USA
[b] College of Comp. Sci. and Software Engr., Shenzhen University, Shenzhen, GD 518060, China
[c] Dept. of Elec. and Comp. Engr., Florida International University, Miami, FL 33174, USA
[d] Dept. of Comp. Sci. and Software Engr., Auburn University, Auburn, AL 36849, USA
[e] College of Comp. Sci., Zhejiang University, Hangzhou, ZJ 310027, China

ARTICLE INFO

Article history:
Received 19 October 2010
Received in revised form 22 January 2011
Accepted 28 March 2011
Available online 5 April 2011

Keywords:
Probability
Robustness
Errors
PMF
Multi-core
Embedded system

ABSTRACT

Multi-core technologies are widely used in embedded systems and the resource allocation is vita to guarantee Quality of Service (QoS) requirements for applications on multi-core platforms. For heterogeneous multi-core systems, the statistical characteristics of execution times on different cores play a critical role in the resource allocation, and the differences between the actual execution time and the estimated execution time may significantly affect the performance of resource allocation and cause system to be less robust. In this paper, we present an evaluation method to study the impacts of inaccurate execution time information to the performance of resource allocation. We propose a systematic way to measure the robustness degradation of the system and evaluate how inaccurate probability parameters may affect the performance of resource allocations. Furthermore, we compare the performance of three widely used greedy heuristics when using the inaccurate information with simulations.

1. Introduction

The embedded multi-core technologies are represented mainly by two categories of multi-core processors [1]: (1) processors with dual, quad, and eight cores based on symmetric multiprocessing and (2) processors with the combination of heterogeneous cores. An example of the later kind of multi-core is the typical *system on chip* (SoC), which has almost unlimited combination of heterogeneous processors on the chip. As the number and the heterogeneity of cores increase, resource allocation management in the embedded multi-core system can efficiently improve the QoS.

Embedded systems usually operate in environments replete with uncertainties [2]. Meanwhile, these systems are expected to provide a given level of QoS. resource allocation can deal with the environment uncertainties and satisfy the QoS demand. In resource allocation, the uncertainties in system parameters and their impacts on system performance can be modeled stochastically. This stochastic model is then used to derive a quantitative evaluation of the robustness of a given resource allocation. This quantitative evaluation results in a probability that the allocation will satisfy the given constraints. A proper approach of stochastic model is using the *probability mass function* (PMF) to describe the probability distributions of execution times of tasks running on cores.

According to [3], any claim of robustness for a given system must answer three questions: (a) what behavior of the system makes it robust? (b) What uncertainties is the system robust against? (c) Quantitatively, how robust is the system? For example, some systems are robust if they are capable of finishing all the tasks within a given deadline. A resource allocation deployed in these systems must be robust against uncertainty of the task execution time. The robustness of a system can also be the makespan (total execution time) or the time slackness.

The problem of resource allocation in the field of heterogeneous multi-core systems is NP-complete (e.g., [4]). Heuristics are used to find near optimal solutions (e.g., [5–11]). In static resource allocations, decisions are made based on estimated PMFs of execution time of tasks running on different cores. However, when estimated PMFs of task execution times are based on inaccurate information, estimated PMFs may be different from actual PMFs. Therefore, decisions generated by estimated PMFs may be less robust and the resource allocation is not able to guarantee the given level of QoS.

For example, in a surveillance sensor network, such as the Omnitrack [12], a large number of cameras are installed across

* Corresponding author.
E-mail addresses: jli6@engr.uky.edu (J. Li), mingz@szu.edu.cn (Z. Ming), mqiu@engr.uky.edu (M. Qiu), Gang.Quan@fiu.edu (G. Quan), xqin@auburn.edu (X. Qin), tzchen@zju.edu.cn (T. Chen).

the target field, and connected to sinks. The tasks of sinks include collecting data from the cameras, recognizing the moving objects in the images, compressing the images, and sending the results to the background server for further processing. After the surveillance sensor network is switched on, the tasks come periodically. To better manage resources of a sink, the operating system in each sink schedules a stochastic static resource allocation before the sensors start working. The estimated PMFs can be obtained by observing previous executions of the tasks or analyzing the codes of the tasks. Using the static stochastic resource allocation, certain level of uncertainties can be tolerated, and the sensor network can maintain a given level of QoS.

However, statistical characteristics of a task may be significantly various from period to period, due to impacts from the environment, the system, and tasks themselves. For instance, when the temperature of the target field increases, processors in the sink of a sensor network may be unstable, leading to longer execution time in average. In this case, the mean of the actual PMF may increase. In another case, the frame size of the image may be reduced by the administrator in a surveillance system, which means the data size decreases and the average execution time of this task is shorter. Besides, tasks may arrive at sinks in a short period of time due to the synchronization among cameras. In this case, a lot of tasks need to wait for execution, queuing in the task buffer of the sinks. Since the order of the queue is random, the execution time of a given task may be random. The deviations of actual PMFs increase.

Some questions arise when estimated PMFs are different from actual PMFs: (1) How does the original static schedule work? Does it still maintain the required level of QoS? (2) If the performance of the original schedule degrades, how much is the degradation? (3) How much improvement can re-scheduling provide? Is re-scheduling a practical solution? The stochastic resource allocation includes a lot of convolutions, which are time consuming. Furthermore, the number of convolutions is proportional to the number of processing units, i.e., cores. The recent many-core technologies provide hundreds of cores in one processor. The re-scheduling may become a significant overhead. Our experiment shows that the Min–min algorithm takes more than an hour to schedule 1024 tasks in an eight-core system. Only when the overhead of re-scheduling is smaller than the degradation of the original schedule, the re-scheduling can be considered as a practical solution.

The major objective of this paper is to answer above questions. In the first part of this work, a stochastic model for resource allocation is presented. The estimated task execution time information is known as a PMF. For a given task schedule, the makespan PMF of a core is generated by convoluting PMFs of all the tasks on its task list. A probability that the whole system can complete all tasks in a certain time is computed by convoluting makespan PMFs of cores. So for a given resource allocation, we find the robustness, e.g., makespan, that system can provide with a given probability. We also propose a measurement metric for the impacts of differences between estimated PMFs and actual PMFs. In the second part of this work, we simulate the environment with inaccurate information and compare three greedy heuristics when using the inaccurate information.

In summary, two major contributions of this work include: (1) The development of a metric for measuring the impact of the inaccurate information on stochastic resource allocation; (2) The performance comparison of three greedy heuristics when using incorrect information.

In Section 2, we discuss related works. In Section 3, models for stochastic task scheduling in multi-core embedded systems are presented. We also provide the model for information inaccuracies in this section. A motivational example is provided in Section 4. We discuss three algorithms for stochastic task scheduling in Section 5, followed by experimental results in Section 6. Finally, we give the conclusion in Section 7.

2. Related works

A framework for robust resource allocation is provided in [3]. Authors in [3] give a robustness definition. Also, a four-step procedure is established for deriving a robustness metric. In step one, the robustness of system is described in a quantitative way, and the range of performance parameter $(\beta_{min}, \beta_{max})$ is given. In step two, all the system and environmental parameters that may impact the robustness of the system are modeled. In step three, the relationship between these perturbation parameters and the performance parameters is defined. Finally, the robust range of perturbation parameter is determined by substituting the perturbation parameters in the range of performance parameter $(\beta_{min}, \beta_{max})$.

Previous works have been reported on determining the stochastic behavior of application execution times [13–19]. A new approach for predicting task execution times is proposed in [20]. In [6], the authors present a derivation of the makespan problem that relies on a stochastic representation of task execution times. In [21], the problem of robust static resource allocation for distributed computing systems under imposed QoS constraints is investigated. A stochastic robustness metric is proposed based on a stochastic model describing the uncertainty in system and its impact on system performance. Although the stochastic representation of task execution times can describe the system uncertainty, problems arise when modeling the stochastic representation. There are two conventional ways to model the stochastic representation that is usually PMFs: (1) using the statistic information from previous runs of the same task to generate the PMF directly; (2) assuming PMFs of task execution times are Gaussian distributions, and using the statistic information from previous runs to determine the expectation and the variance [21]. However when the environment is changed, these stochastic representations may not be accurate. For example, a set of PMFs are generated based on some previous runs that occur in a light-weight contention scenario. When they are applied in other heavy contention scenarios, these PMFs are not accurate in the sense that actual ones may have larger variance due to the heavy contention. So resource allocation with these inaccurate PMFs may lead to the violation of QoS requirements. The related works above does not evaluate what the relationship is between the degree of inaccurate in stochastic representation and the degradation of robustness in the system.

3. Model and definition

3.1. Stochastic model

In a normal heterogeneous multi-core embedded system, usually there is a set of tasks to be executed. Also, there are a number of cores with various computation power and characteristics in the system. An estimated probabilistic *estimated time to compute* (ETC) matrix P is known before scheduling. For the convenience of readers, we list acronyms used in the rest of this paper in Table 1. We assume that the estimated probabilistic ETC matrix is generated using the second approach as discussed in Section 2. The entry $P_{i,j}$ of P represents the PMF of execution time of task i on core j. When making mapping decisions, we use the information to generate probability distributions of task completion times on different cores.

For a given set of tasks and a given schedule, the *estimated makespan* distribution is the probability distribution of total execution time of the whole set of tasks based on the ETC matrix. We can

Table 1
Acronyms used in the paper.

Name	Description
QoS	Quality of the service
PMF	Probability mass function
ETC	Estimated time to compute
CAT	Core available time
MCT	Minimum completion time alogrithm
M_o	Original makespan
M_n	New makespan
M_c	Correct makespan
MN_o	Normalized original makespan
MN_n	Normalized new makespan
MN_c	Normailzed correct makespan
R_n	New_ratio
R_c	Correct_ratio
R_i	Improve_ratio

calculate this probability distribution by convoluting probability distributions of task execution times. The robustness in this paper is the minimum makespan (Λ) while maintaining a pre-determined probability θ that all cores will complete their tasks list within Λ.

As estimated PMFs of task execution times are generated with statistic information of previous runs of tasks, any environment or system changes may lead to inaccuracy. Assuming that we can get the updated information about those distribution by some methods, we are able to obtain a resource allocation that meets the QoS requirement with more confidence. We call these distributions (PMFs) updated PMFs. There are methods to obtain updated PMFs, for example, on-line profiling [22,23]. The development of these methods is out of the scope of this paper.

In the case that we can get updated PMFs of task execution times, whether a new resource allocation is necessary becomes another problem. Using a new resource allocation not only requires time to re-run the scheduling algorithm, but also brings the overhead of re-arranging resources in the system. However, if we can predict the degradation of robustness based on the difference between updated PMFs and estimated PMFs, i.e., the degree of inaccurate information, we can decide whether a new resource allocation is necessary. Furthermore, with knowledge of which scheduling algorithm performs the best when using inaccurate information, we can reduce the probability that a new resource allocation is necessary by using the best scheduling algorithm. We will provide some insights on these two questions in our evaluation part in the paper.

3.2. Measurement parameters

Since differences between estimated PMFs and updated PMFs may cause the robustness degradation, several measurement parameters are introduced to measure the robustness degradation.

- *Original Schedule*: Task Schedule generated by using estimated PMFs
- *Remapped Schedule*: Task Schedule generated by using updated PMFs
- *Makespan*: The total time taken for a system to finish all tasks with a given task schedule
- *Original Makespan* (M_o): The makespan using estimated PMFs and the original Schedule
- *New Makespan* (M_n): The makespan using updated PMFs and the original Schedule
- *Correct Makespan* (M_c): The makespan using updated PMFs and the remapped Schedule
- *New_ratio* (R_n):

$$R_n = \frac{M_n - M_o}{M_o} \tag{1}$$

- *Corretc_ratio* (R_c):

$$R_c = \frac{M_c - M_o}{M_o} \tag{2}$$

- *Improve_ratio* (R_i):

$$R_i = \frac{M_n - M_c}{M_c} \tag{3}$$

As discussed in the previous section, the robustness metric in this paper is the minimum makespan (Λ) while maintaining a pre-determined probability θ that all cores will complete their tasks list within Λ. The smaller the makespan (Λ) is, the more robust the system is. Original makespan gives the robustness of the system assuming accurate information is used in the schedule. When inaccurate information is used in the original schedule, new makespan results in the actual robustness of the system without re-running the scheduling algorithm. Correct makespan indicates the new robustness when a new schedule is generated with updated accurate information. New_ratio shows the degradation of the robustness when using the inaccurate information. Improve_ratio reveals the improvement caused by re-running the scheduling algorithm. Correct_ratio indicates impacts of changes of environment on the system's robustness.

4. Motivational example

In this section, we will demonstrate how the inaccurate information impacts the robustness of a schedule. Consider a case with five independent tasks that need to be scheduled in a two cores embedded system. The estimated execution time distributions of different tasks running in these two-core are shown in Fig. 1(a). We assume all these distributions are normal distributions as shown in Fig. 1(b).

In this example, we use the Min-min heuristic, which will be introduced in the next section, to schedule these independent tasks. Task A is scheduled first in core $P1$, followed by task C in core $P0$. Then we schedule task D in core $P1$ right after task A, and task B in core $P0$, as shown in Fig. 2. After we schedule these four tasks in the system, we can compute the probability distributions of makespans in these two cores by convoluting task execution time distributions. Makespan distributions are shown in Fig. 3. For each of these two cores, we can calculate the convolution of the makespan distribution of the core and the execution time distribution of E running in the core, which is shown in Fig. 4. By comparing results of these convolutions, we can make a greedy decision of which core task E is scheduled to. If task E is scheduled in $P0$, all five tasks can be finished by time 34, with the probability of 90%. Otherwise, If task E is scheduled in $P1$, all tasks can be finished by time 27 with the probability of 90%. We schedule task E in $P1$.

In some cases, current statistical characteristics of the task execution time may be different from previous estimated ones. The estimated PMF cannot represent the actual distribution of the task execution time accurately. Assuming that the actual distribution of task E is different from the estimated one, the distribution of E in core $P0$ is a normal distribution with the mean of 9, and the standard deviation of 1, while the distribution in core $P1$ is another normal distribution with the mean of 14 and the standard deviation of 6. In this case, if E is scheduled in $P1$,

(a)

Task	P0		P1	
	Mean (μ)	Standard Deviation(σ)	Mean (μ)	Standard Deviation(σ)
A	6	2	4	1
B	10	4	12	7
C	5	3	8	6
D	7	2	7	1
E	10	1	12	2

(b)

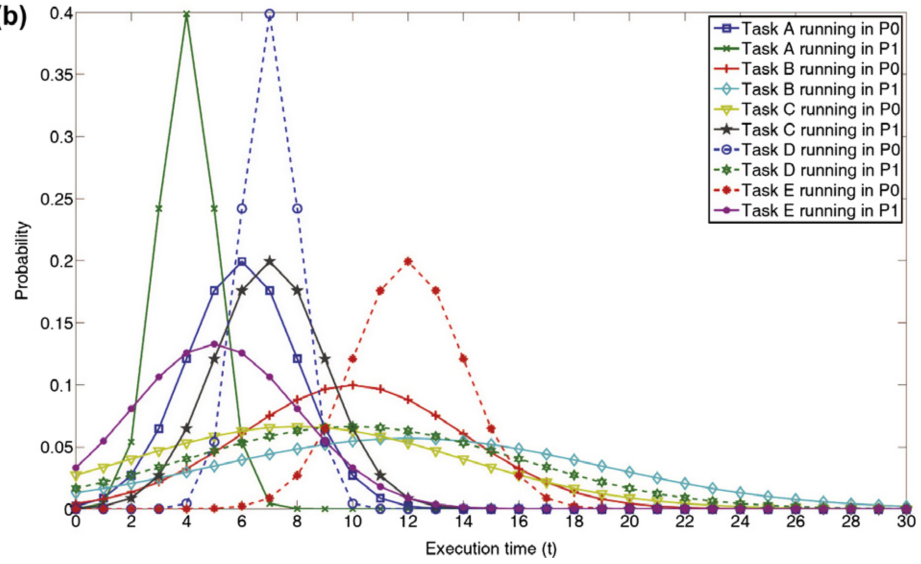


Fig. 1. (a) Means and standard deviations of the task execution time distributions; (b) Normal distributions of task execution time.

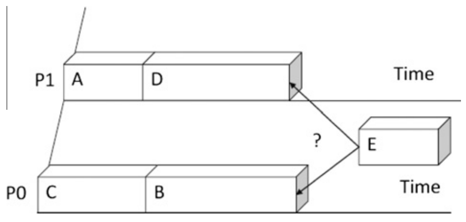


Fig. 2. The schedule without task *E*.

the system will finish tasks by time 34 with 90% guarantee, about 26% robustness degradation. If *E* is scheduled in *P*0, all tasks will be done by time 33 with 90% guarantee, which results in a different greedy decision from the one based on estimated information as shown in Fig. 5.

In this example, the inaccurate information can degrade the robustness, i.e., makespan in this example. Therefore, we will investigate how different degrees of inaccurate impact the robustness and how different scheduling heuristics perform under an inaccurate information environment in following sections.

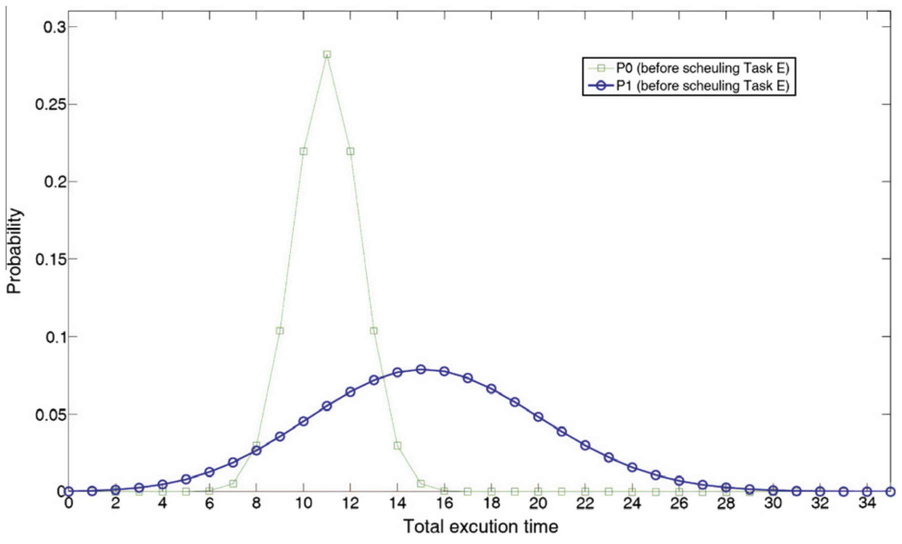


Fig. 3. Makespan probability distributions of cores before task *E* is scheduled.

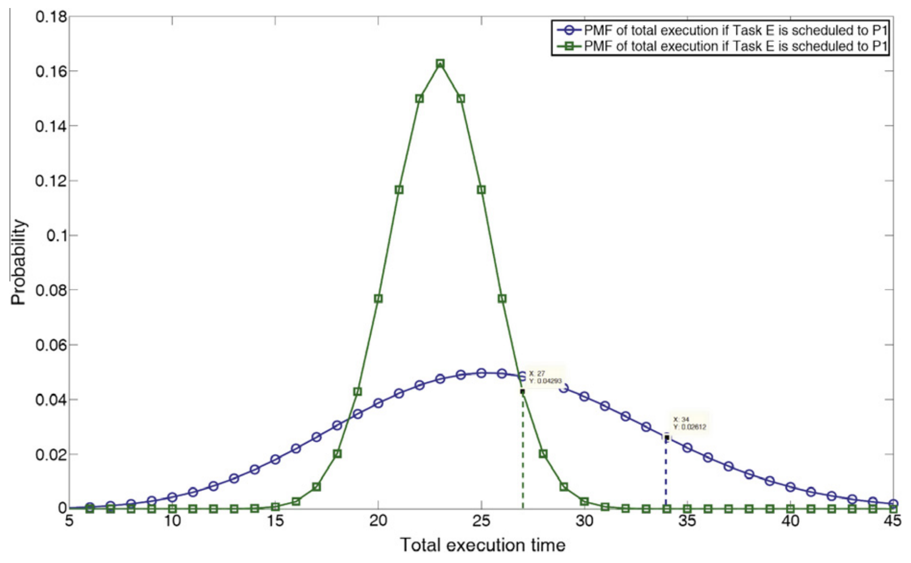


Fig. 4. Estimated makespan probability distributions of cores after task *E* is scheduled.

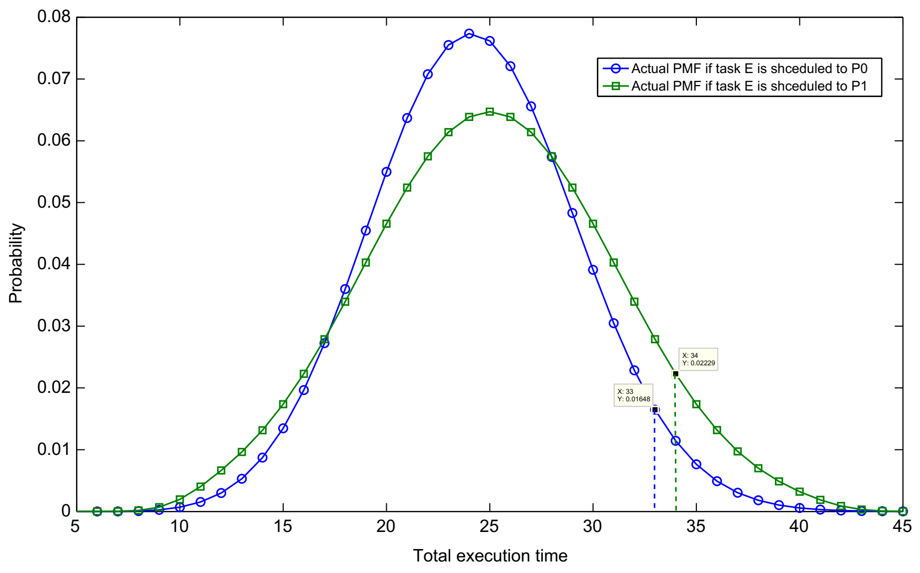


Fig. 5. Actual makespan probability distributions of cores after task *E* is scheduled.

5. Algorithms

5.1. Overview

Three static greedy heuristics are used. *Minimum completion time* (MCT) [24] is an one-phase heuristic. The output of this heuristic depends on the order in which tasks are mapped to cores. *Min–min* [24,25] and *Max–min* [24,25] are two-phase heuristics. These two heuristics are independent from tasks assigning order in the sense that for a given set of tasks and a system with a certain set of cores, outputs are identical no matter how many times it runs.

Greedy heuristics are widely used in heterogeneous system resource allocation. Compared to global heuristics such as *genetic algorithm* and *simulated annealing*, greedy heuristics can get a schedule much quicker than global heuristics. Previous works show that Min–min heuristics can get a schedule as optimal as the one generated by a genetic algorithm.

Definitions of these three heuristics are provided below. *Core available time* (CAT) is the probability distribution of time when the core will finish all tasks that are assigned to this core previously. The PMF of the completion time for a new task t_i on core $c_j, ct_{i,j}$, can be calculated by convoluting the CAT of core c_j and the execution time distribution of task t_i on core c_j.

5.2. MCT

Minimum Completion Time (MCT) [24] assigns tasks in an arbitrary order to cores. For an unmapped task, MCT maps it on the core that can complete this task in the earliest time while maintaining a certain probability. The idea behind MCT is that it considers both the execution time of the task on the core as well as the load balance. Since MCT assigns tasks in an arbitrary order, the scheduling results are non-determinstic. The MCT algorithm is shown in Fig. 6.

5.3. Min–min

Min–min [24,25] selects the task-core pair in two phases. In phase 1, for each unmapped task, the core that can complete it in the earliest time while maintaining a certain probability is selected to form a pair. In phase 2, among all pairs, the pair that has the minimum *ct* is selected, and the task in the pair is mapped to the corresponding core. The idea behind Min-min is that it does its best to keep the current load balance with the least change on it. The Min0-min is provided in Fig. 7.

5.4. Max–min

Max–min [24,25] is similar to Min–min. In phase 1, Max–min does exactly the same as that of Min–min. Then in phase 2, Max–min finds the task-core pairs with the maximum *ct*, which is different from Min–min. The idea behind is that tasks with larger execution time will likely increase the penalty if these tasks are not assigned to their best cores. Fig. 8 shows the Max–min algorithm.

6. Simulation

6.1. Simulation setup

To evaluate the robustness degradation caused by the inaccurate information, the following approach was used to simulate the stochastic resource allocation in a heterogeneous multi-core embedded system. A set of 1024 independent tasks was formed randomly. They consist of 28 task classes, where tasks in the same class are identical. There are 8 heterogeneous cores in a system. Each of these cores has its own computation power and characteristic. So the estimated probabilistic ETC matrix P has the size of 28×8. PMF $P_{i,j}$ is based on Gamma distribution with a mean of $m_{i,j}$ and a standard deviation of $sd_{i,j}$. In our simulation, we generate PMFs by sampling the *probability density functions* (PDF) of Gamma distributions with a start point, an end point and a fixed step. Each of the 40 simulation trials has different estimated probabilistic ETC matrix P.

```
Require: a set of tasks, $m$ different cores, ETC PMF matrix
Ensure: A MCT resource allocation schedule
1: A list of unmapped tasks $U$ is generated.
2: Reorder the list in an arbitrary order.
3: while the list $U$ is not empty do
4:     The first task $i$ in the list $U$ is selected; then among $m$ cores, the core $j$
       which has the minimum $ct_{i,j}$ is also selected.
5:     Assign the task to the core.
6:     Remove the task from the list $U$.
7:     Update the CAT of the selected core.
8: end while
```

Fig. 6. MCT algorithm.

```
Require: a set of tasks, $m$ different cores, ETC PMF matrix
Ensure: A Min-min resource allocation schedule
1: A list of unmapped tasks $U$ is generated.
2: while the list $U$ is not empty do
3:     For each task in the list $U$, find the core that gives the minimum $ct$.
4:     Among task-core pairs formed in step 3, find the pair with the minimum
       $ct$.
5:     Assign the task in the selected pair to the according core.
6:     Remove the task from the list $U$.
7:     Update the CAT of the selected core.
8: end while
```

Fig. 7. Min-min algorithm.

```
Require: a set of tasks, $m$ different cores, ETC PMF matrix
Ensure: A Max-min resource allocation schedule.
1: A list of unmapped tasks $U$ is generated.
2: while the list $U$ is not empty do
3:     For each task in the list $U$, find the core that gives the minimum $ct$.
4:     Among task-core pairs formed in step 3, find the pair with the maxi-
       mum $ct$.
5:     Assign the task in the selected pair to the according core.
6:     Remove the task from the list $U$.
7:     Update the CAT of the selected core.
8: end while
```

Fig. 8. Max–min algorithm.

```
Require: $t$ different tasks, $m$ different cores, coefficient of variation of task
    and core $V_{task}$, $V_{core}$, mean of tasks' ETC $\mu_{task}$
Ensure: A random ETC matrix based on Gamma distribution
1: Compute the shape parameter and the scale parameter of task as well as
   the shape parameter of core
   $\alpha_{task} = 1/{V_{task}}^2 \alpha_{core} = 1/{V_{core}}^2$
   $\beta_{task} = \mu_{task}/\alpha_{task}$
2: for $i$ from 0 to $(t-1)$ do
3:     $q[i] = G(\alpha_{task}, \beta_{task})$
       /*$q[i]$ will be used as mean of $i$-th row in the ETC matrix*/
4:     $\beta_{core}[i] = q[i]/\alpha_{core}$
       /*scale parameter for $i$-th row*/
5:     for $j$ from 0 to $(m-1)$ do
6:         $e[i,j] = G(\alpha_{core}, \beta_{core}[i])$
7:     end for
8: end for
```

Fig. 9. COV based method for generate Gamma random matrix.

Before generating PMFs of Gamma distributions, values of means and standard deviations need to be determined. We randomly generate a 28×8 mean matrix based on Gamma distribution as well as the standard deviation matrix. Here, we use the COV based method [26] with the mean of task execution time from 40 to 80, and both coefficients of variation of tasks and cores uniformly from 0.35 to 1, as shown in Fig. 9. When forming the PMF $P_{i,j}$, we can sample the PDF of Gamma distribution with a mean of $m_{i,j}$ and a standard deviation of $sd_{i,j}$. The objective of this method to generate PMFs for simulation. And this method can be implemented easily by a statistical computing tool *R* [27]. In literature, there are several low-overhead methods [28–30] to generate stochastic profiles with sufficient coverage of variances in practical applications.

To simulate the case in which updated PMFs are different from estimated PMFs, parameters (mean or standard deviation) of updated PMFs are generated by multiplying parameters of estimated PMFs with a scalar matrix S.

For example, if mean values are modified,

$$updated\_mean(i,j) = mean(i,j) \times S_{i,j} \qquad (4)$$

The entry of scalar matrix S is based on a uniform distribution with a range of $[S_{min}, S_{max}]$.

6.2. Simulation results

6.2.1. Compare impacts on robustness when modifying different parameters

In this part, we compare impacts on robustness when using different scalar matrixes as well as modifying different parameters.

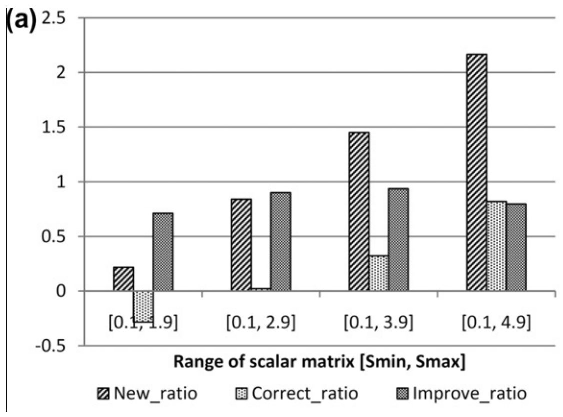
We simulate two different scenarios in which two different kinds of inaccurate information occur:

1. Keep standard deviations unchanged, and multiply means with a scalar matrix.
2. Keep means unchanged, and multiply standard deviations with a scalar.

The first scenario usually happens when the embedded system is employed in a physically inconstant environment. For example, in an environment where temperature changes rapidly, cores will likely run faster in low temperature than that in high temperature. As the temperature increases, means of the probability distribution of execution times may increase. In this case, the statistic information collected previously in low temperature may not be accurate. The second scenario happens when resource contention among tasks changes. When the resource contention is light, a core likely finishes same tasks in a narrow distribution, especially around the mean of the distribution. When the contention is heavy, the distribution of a task class in a core may be wide, i.e., with larger standard deviations. In our simulation, the scalar matrixes are within the range of [0.1, 1.9], [0.1, 2.9], [0.1, 3.9], [0.1, 4.9].

MCT heuristic is used in all these four parameter modifications. The result of each trial is the average value of MCT with 25 different task mapping order.

In Fig. 10(a), the increase of new_ratio is proportional to the increase of the scalar matrix range with 20% to 70% penalty. Obviously, the increase of mean values of the execution time distribution leads to a longer makspan. This 20% to 70% penalty is caused by the inaccurate information used in the original schedule. We find that the improve_ratio, which indicates the improvement of re-scheduling, does not change as much as the increase of the scalar matrix range. Note that when we calculate the improve_ratio, we compare the difference between the new_makespan and the correct_makespan. In the convolution of these two distributions, we use the updated PMFs. The "improve_ratio" columns show that the level of improvement brought from the re-scheduling does not mainly depend on the inaccurate degree of the information, but depends on what the task set consists of. The correct_ratio is also proportional to the increase of the scalar matrix range. It shows that the degradation of robustness is a linear function of the degree of how the environment changes. Comparing Fig. 10(b) with Fig. 10(a), we find that the inaccurate standard deviations have much less impacts on the robustness than that of the inaccurate means.



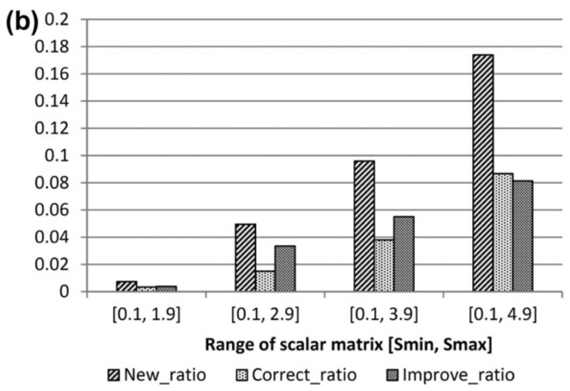


Fig. 10. (a) New_ratio, correct_ratio and improve_ratio when changing the mean; (b) new_ratio, correct_ratio and improve_ratio when changing the standard deviation.

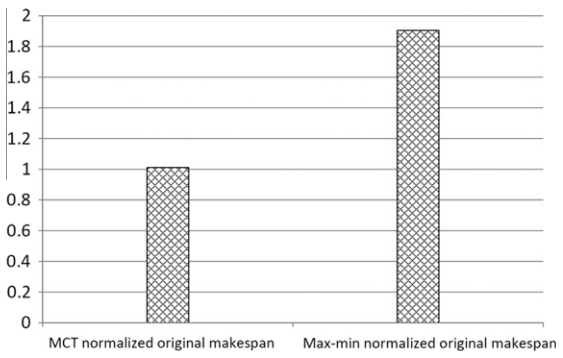


Fig. 11. The original makespan when changing the mean and the standard deviation with a fixed scale parameter.

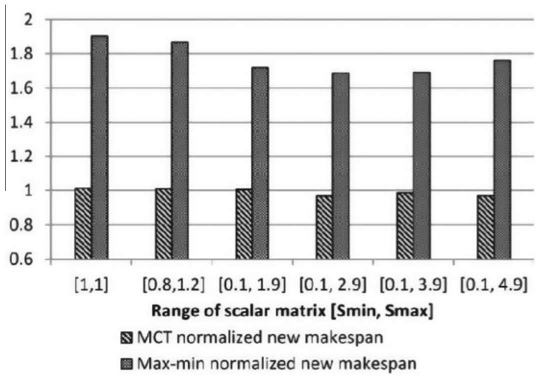


Fig. 12. The normalized new makespan when changing the mean and the standard deviation with a fixed scale parameter.

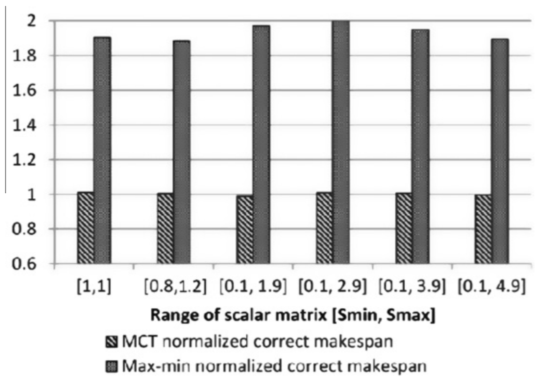


Fig. 13. The normalized correct makespan when changing the mean and the standard deviation with a fixed scale parameter.

6.2.2. Compare the performance of different heuristics

In this part, three different heuristics (Min–min, MCT, Max–min) are compared with their performance when using inaccurate information. In this part, we will keep the standard deviations fixed and change mean values. To compare the performance of these heuristics, normalized makespans of MCT and Max-min are introduced.

- Max–min normalized original makespan

$$MN_o(Max\text{—}min) = \frac{M_o(Max\text{—}min)}{M_o(Min\text{—}min)} \tag{5}$$

- Max–min normalized new makespan

$$MN_n(Max\text{—}min) = \frac{M_n(Max\text{—}min)}{M_n(Min\text{—}min)} \tag{6}$$

- Max–min normalized correct makespan

$$MN_c(Max\text{—}min) = \frac{M_c(Max\text{—}min)}{M_c(Min\text{—}min)} \tag{7}$$

- MCT normalized original makespan

$$MN_o(MCT) = \frac{M_o(MCT)}{M_o(Min\text{—}min)} \tag{8}$$

- MCT normalized new makespan

$$MN_n(MCT) = \frac{M_n(MCT)}{M_n(Min\text{—}min)} \tag{9}$$

- MCT normalized correct makespan

$$MN_c(MCT) = \frac{M_c(MCT)}{M_c(Min\text{—}min)} \tag{10}$$

In the respect of the three ratios (Nnew_ratio, correct_ratio, and improve_ratio), Figs. 14–16 show that the Max-min is least impacted by the inaccurate information. However, in Figs. 11–13, Max–min has the longest new makespans and the longest correct makespans among these three heuristics. It means that the Max–min generates the least robust schedules in the environment with or without inaccurate information, even though the inaccurate information has smallest impacts in the Max–min. So the Max–min performance is the worst among these three heuristics. The performance of MCT is very close to the performance of Min-min with respect to the original makespan. Furthermore, MCT outperforms the Min–min in the new makespan. It means that MCT is less impacted by the inaccurate information and performs close to the Min–min in the original makespan, and it performs the best in the new makespan even though the difference between these two heuristics is not significant.

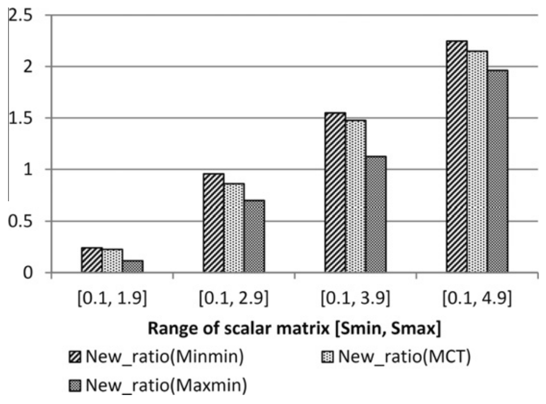


Fig. 14. The new_ratio of three heuristics when changing the mean and the standard deviation with a fixed scale parameter.

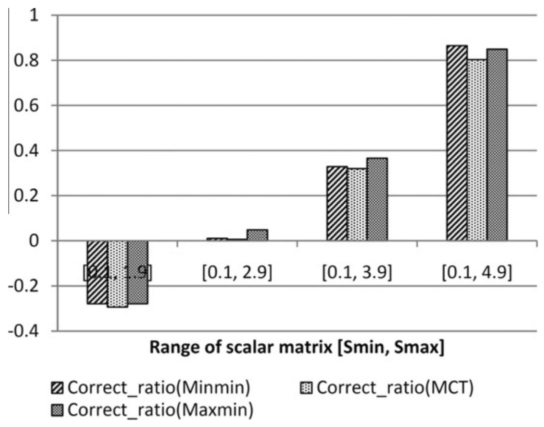


Fig. 15. The correct_ratio of three heuristics when changing the mean and the standard deviation with a fixed scale parameter.

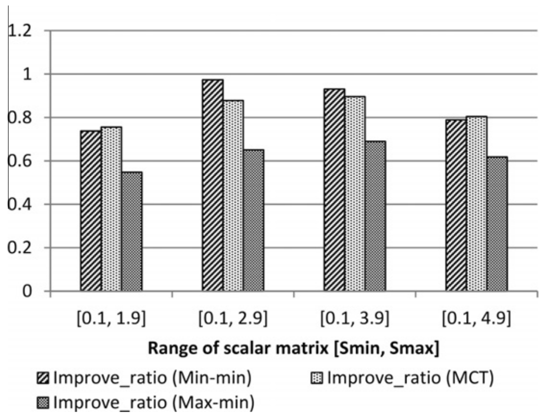


Fig. 16. The improve_ratio of three heuristics when changing the mean and the standard deviation with a fixed scale parameter.

7. Conclusion

We propose a systematic method of measuring the robustness degradation with a stochastic approach. We evaluate impacts of inaccurate information on system robustness in two different scenarios. In our simulation, the makespan is the robustness metric. We find that the makespan with inaccurate information increases proportional to the increase of mean values of task execution time distribution caused by environment changes. Also, 20% to 70% penalty is caused by the inaccurate information used in making scheduling decisions. The impact of environment changes on the robustness is linear to the degree of how much inaccurate information (mainly the shift of means of PMFs) is generated by these environment changes. However, the improvement of re-scheduling with updated information mainly depends on how the task set consists of, not how inaccurate the information is. We also find that the impact of inaccurate means of PMFs is much larger than inaccurate standard deviations.

Among these three greedy algorithms, MCT performs the best under inaccurate information. It generates schedules that are almost as optimal as ones from Min–min where accurate information is used. And inaccurate information has less impacts on schedules from MCT than it does on Min–min. Max–min performs the worst.

Acknowledgements

This work was supported in part by the NSFC 61071061, the Univ. of Kentucky Start Up Fund; SZ-HK Innovation Circle proj.

ZYB200907060012A, NSF GD:10351806001000000, S&T proj. of SZ JC200903120046A; the NSF CNS-0969013, CNS-0917021, and CNS-1018108; the NSF CNS-0915762 (CSR), CCF-08452578 (CAREER), CNS-0917137 (CSR), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0831502 (CyberTrust), CNS-0855251 (CRI), OCI-0753305 (CI-TEAM), DUE-0837341 (CCLI), and DUE-0830831 (SFS), Auburn Start Up grant and Intel gift (2005-04-070); the NSFC 61070001, RFEB Zhejiang Y200803333 and Y200909683, State Key Lab of High End Server Storage Tech. 2009HSSA10, National Key Lab STASI, SFKPC 2009ZX01039-002-001-04, 2009ZX03001-016, 2009ZX 03004-005.

References

[1] M. Levy, T.M. Conte, Embedded multicore processors and systems, IEEE Micro 29 (3) (2009) 7–9.
[2] M. Qiu, E.H.M. Sha, Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems, ACM Transactions on Design Automation of Electronic Systems (TODAES) 14 (2) (2009) 1–30.
[3] S. Ali, A. Maciejewski, H. Siegel, J. Kim, Measuring the robustness of a resource allocation, IEEE Transactions on Parallel and Distributed Systems (2004) 630–641.
[4] E. Coffman, J. Bruno, Computer and Job-Shop Scheduling Theory, John Wiley & Sons, 1976.
[5] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810–837.
[6] T. Braun, H. Siegel, A. Maciejewski, S. Noemix, Static mapping heuristics for tasks with dependencies, priorities, deadlines, and multiple versions in heterogeneous environments, in: International Parallel and Distributed Processing Symposium (IPDPS), Fort Lauderdale, Florida, USA, 2002, pp. 78–85.
[7] M. Eshaghian, Heterogeneous Computing, Artech House Publishers, 1996.
[8] D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Transactions on Software Engineering 15 (11) (1989) 1427–1436.
[9] C. Leangsuksun, J. Potter, S. Scott, Dynamic task mapping algorithms for a distributed heterogeneous computing environment, in: 4th IEEE Heterogeneous Computing Workshop (HCW), Santa Barbara, California, USA, 1995, pp. 30–34.
[10] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, R. Freund, A comparison of dynamic strategies for mapping a class of independent tasks onto heterogeneous computing systems, in: Proceedings of the Heterogeneous Computing Workshop (HCW), Orlando, Florida, USA, 1998, pp. 57–69.
[11] L. Wang, H. Siegel, V. Roychowdhury, A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, Journal of Parallel and Distributed Computing 47 (1) (1997) 8–22.
[12] B. Li, K. Nahrstedt, Qualprobes: Middleware QoS profiling services for configuring adaptive applications, Lecture Notes in Computer Science 1795/2000 (2002) 256–272.
[13] L. David, I. Puaut, Static determination of probabilistic execution times, in: Euromicro Conference on Real-Time Systems (ECRTS), Catania, Italy, 2004, pp. 223–230.
[14] Y. Li, J. Antonio, H. Siegel, M. Tan, D. Watson, Determining the execution time distribution for a data parallel program in a heterogeneous computing environment, Journal of Parallel and Distributed Computing 44 (1) (1997) 35–52.
[15] G. Bernat, A. Colin, S. Petters, WCET analysis of probabilistic hard real-time systems, in: IEEE real-time systems symposium, 2002, pp. 279–288.
[16] M. Qiu, L. Yang, Z. Shao, E.H.-M. Sha, Rotation Scheduling and Voltage Assignment to Minimize Energy for SoC, in: The International Conference on Computational Science and Engineering, 2009, pp. 48–55.
[17] M. Qiu, Z. Jia, C. Xue, Z. Shao, E.H.M. Sha, Voltage assignment with guaranteed probability satisfying timing constraint for real-time multiproceesor dsp, Journal of VLSI Signal Processing Systems 46 (1) (2007) 55–73.
[18] V. Shestak., J. Smith., H.J. Siegel, A.A. Maciejewski, A stochastic approach to measuring the robustness of resource allocations in distributed systems, in: International Conference on Parallel Processing (ICPP), Columbus, Ohio, USA, 2006, pp. 459–470.
[19] J. Smith, E.K.P. Chong, A.A. Maciejewski, H.J. Siegel, Stochastic-based robust dynamic resource allocation in a heterogeneous computing system, in: International Conference on Parallel Processing (ICPP), Vienna, Austria, 2009, pp. 188–195.
[20] M. Iverson, F. Ozguner, L. Potter, Statistical prediction of task execution times through analyticbenchmarking for scheduling in a heterogeneous environment, in: Heterogeneous Computing Workshop (HCW), San Juan, Puerto Rico, 1999, pp. 99–111.
[21] V. Shestak, J. Smith, A. Maciejewski, H. Siegel, Stochastic robustness metric and its use for static resource allocations, Journal of Parallel and Distributed Computing 68 (8) (2008) 1157–1173.
[22] X. Zhang, Z. Wang, N. Gloy, J.B. Chen, M.D. Smith, System support for automatic profiling and optimization, Computer Networks 38 (4) (2002) 393–422.
[23] C. Krintz, Coupling on-line and off-line profile information to improve program performance, in: International Symposium on Code Generation and Optimization, San Francisco, California, USA, 2003, pp. 69–78.
[24] R. Armstrong, D. Hensgen, T. Kidd, The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions, in: 7th IEEE Heterogeneous Computing Workshop (HCW), Orlando, Florida, USA, vol. 5, 1998.
[25] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. Lima, et al., Scheduling resources in multi-user, heterogeneous, computingenvironments with SmartNet, in: IEEE Heterogeneous Computing Workshop (HCW), Orlando, Florida, USA, 1998, pp. 184–199.
[26] S. Ali, H. Siegel, M. Maheswaran, D. Hensgen, S. Ali, Representing task and machine heterogeneities for heterogeneous computing systems, Tamkang Journal of Science and Engineering 3 (3) (2000) 195–208.
[27] The *R* project for statistical computing, 2010. <http://www.r-project.org/>.
[28] G. Bernat, A. Colin, S.M. Petters, WCET analysis of probabilistic hard real-time systems, in: IEEE Real-Time Systems Symposium (RTSS), Austin, Texas, USA, 2002, pp. 279–288.
[29] A. Burns, G. Bernat, I. Broster, A probabilistic framework for schedulability analysis, Lecture Notes in Computer Science 2855/2003 (2003) 1–15.
[30] Y. Li, J.K. Antonio, H.J. Siegel, M. Tan, D.W. Watson, Determining the execution time distribution for a data parallel program in a heterogeneous computing environment, Journal of Parallel and Distributed Computing 44 (1) (1997) 35–52.

Jiayin Li received the B.E. and M.E. degrees from Huazhong University of Science and Technology (HUST), China, in 2002 and 2006, respectively. And now he is pursuing his Ph.D. degree in the Department of Electrical and Computer Engineering (ECE), University of Kentucky. His research interests include software/hardware co-design for embedded systems and high performance computing.

Zhong Ming is a professor at College of Computer and Software Engineering of Shenzhen University. He is a member of a council and senior member of China Computer Federation. His major research interests are software engineering and embedded systems. He led two projects of National Natural Science Foundation, and two projects of Natural Science Foundation of Guangdong province, China.

Meikang Qiu received the B.E. and M.E. degrees from Shanghai Jiao Tong University, China. He received the M.S. and Ph.D. degrees of Computer Science from University of Texas at Dallas in 2003 and 2007, respectively. He had worked at Chinese Helicopter R&D Institute and IBM. Currently, he is an assistant professor of ECE at University of Kentucky. He is an IEEE Senior member and has published more than 100 peer reviewed papers, including 35 journal papers. He has been on various chairs and TPC members for many international conferences. He served as the Program Chair of IEEE EmbeddCom'09 and EM-Com'09. He received Air Force Summer Faculty Award 2009. He won three best paper awards (IEEE Embedded and ubiquitous Computing (EUC'09), IEEE/ACM GreenCom'10, and IEEE CSE'10) and one best paper nomination. His research interests include embedded systems, computer security, and wireless sensor networks.

Gang Quan is currently an Associate Professor with the Electrical and Computer Engineering Department, Florida International University, Miami. He received the B.S. degree from the Tsinghua University, Beijing, China, the M.S. degree from the Chinese Academy of Sciences, Beijing, and the Ph.D. degree from the University of Notre Dame, Notre Dame, IN. His research interests includes real-time system, power/thermal aware design, embedded system design, advanced computer architecture and reconfigurable computing. Prof. Quan received the NSF CAREER award in 2006.

Xiao Qin received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology, Wuhan, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska- Lincoln in 2004. He is currently an associate professor of computer science at Auburn University. Prior to joining Auburn University in 2007, he had been an assistant professor with New Mexico Institute of Mining and Technology (New Mexico Tech) for three years. He won an NSF CAREER award in 2009. His research interests include parallel and distributed systems, real-time computing, storage systems, fault tolerance, and performance evaluation. His research is supported by the U.S. National Science Foundation, Auburn University, and Intel Corporation. He is a senior member of the IEEE and the IEEE Computer Society.

Tianzhou Chen is a professor of computer science at Zhejiang University. His current research interests include computer architecture, multi-core system, on-chip interconnection, embedded system, power-aware computing, hardware/software co-design and security.