Contents lists available at ScienceDirect

Future Generation Computer Systems

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

A task-efficient sink node based on embedded multi-core soC for Internet of Things

Tie Qiu[a,*], Aoyang Zhao[a], Ruixin Ma[a], Victor Chang[b], Fangbing Liu[a], Zhangjie Fu[c]

[a] *School of Software, Dalian University of Technology, Dalian 116620, China*
[b] *IBSS, Xi'an Jiaotong Liverpool University, Suzhou 215123, China*
[c] *School of Computer and Software, Nanjing University of Information Science and Technology, Nanjing 210044, China*

HIGHLIGHTS

- TESN processes tasks in parallel. It has heterogeneous architecture, including one master core and seven slave cores.
- TESN adopts WLC to allocate tasks, which balances the load of each slave core and improves the tasks processing speed obviously.
- Compared with single core system, TESN effectively reduces the congestion situation.
- The performance of WLC based on multi-core processor is better than RR when the number of soft cores is more than two.

ARTICLE INFO

Article history:
Received 29 February 2016
Received in revised form
20 October 2016
Accepted 17 December 2016
Available online xxxx

Keywords:
Internet of Things
Sink node
Multi-core
Task allocation strategy
Load balance

ABSTRACT

With the increase of collected information, the computing performance of single-core sink node for Internet of Things (IoTs) cannot satisfy with the demand of large data processing any more. Therefore, the sink node which based on embedded multi-core SoC for IoTs and maximizing its computing performance has brought into focus in recent years. In this paper, we design a multi-core Task-Efficient Sink Node (TESN) based on heterogeneous architecture and the Weighted-Least Connection (WLC) task schedule strategy has been proposed to improve its efficiency. There are two types of cores in the sink node, master core and slave cores. The master core deals with tasks allocation and the seven slave cores deal with data processing. All of the cores are communicating with each other through mailbox. By considering each core's real-time processing information and computing performance, the proposed WLC can balance each core's load and reduce network congestion. The *Xilinx V*5 platform is used to evaluate the performance of WLC and Round-Robin (RR) algorithms for multi-core sink node. The experiment results show that the WLC strategy improves the processing speed obviously, achieves load balance and avoids large scale congestion of sink node in the sensor networks of IoTs.

1. Introduction

Internet of Things (IoTs) is a huge integration of many fields [1,2], including sensor networks, embedded systems, intelligent control [3,4], data processing and fusion, task scheduling and allocation [5,6] etc. The emerging application in IoTs are often based on networks and sensing. Such as smart city and smart home. In order to sense environmental factors, the large-scale nodes are deployed in the distributed areas, include sink node and sensor nodes [7]. Sensor nodes can collaborate monitoring, sensing and collecting environment information, which cover the geographic region, and send collected data to sink node through multi-hop [8,9]. Sink node is responsible for collecting and analyzing the sensing data from other sensor nodes. It needs more powerful computing and data processing generally. In recent years, multi-core system has been used in dealing with larger data. Each core runs synchronously and they coordinate with each other to complete tasks. The way of parallel computing [10] greatly improves the efficiency of sink node. Therefore, multi-core system is essential for real-time system in IoTs.

With the development of innovative applications, IoTs [11,12] has been applied in the field of industrial automation, transport systems, cities etc. For example, in healthcare, IoTs plays a vital role in collecting thousands of patients' data and monitoring the health state of patients. To keep tabs on the location of medical devices,

* Corresponding author. Fax: +86 411 62274416.
E-mail address: qiutie@ieee.org (T. Qiu).

http://dx.doi.org/10.1016/j.future.2016.12.024
0167-739X/© 2016 Elsevier B.V. All rights reserved.

patients, and personnel, many hospitals use IoTs. In the future, the IoTs will be widely used in smart home, smart city, wearable devices etc., which enables people to manage their daily lives with more efficient monitoring and control.

Embedded multi-core SoC contains multiple executing cores, which are integrated into a processor chip. Therefore, it is a parallel processor with high computing performance. Embedded multi-core SoC computes parallelly and synchronously, it can execute complex computing [13,14] and dealing with big data processing. Each core in embedded multi-core SoC is an independent processing unit. These cores are tightly coupled, they communicate with each other through Shared-bus [15] or Data Hiway[16].

Compared with traditional single-core processor, multi-core processor [17,18] can greatly improve the computing performance. Especially for large-scale IoTs, the sink node with single-core cannot satisfy the requirement of applications. Therefore, we design a Task-Efficient Sink Node (TESN) based on embedded multi-core SoC for IoTs. It can parallelly execute process and achieve better load balance [19]. There are two architecture types of multi-core processor: homogeneous [20] and heterogeneous [21]. Considering the better scalability and functional diversity, the heterogeneous multi-core processor is widely used for IoTs. However, all of the cores must be cooperating to complete the tasks. In order to optimize the performance of multi-core processor, the effective strategy for task allocation and scheduling [22,23] is a significant part to further improve the performance of multi-core sink node and prolong its life-time [24].

For TESN, when allocating task to processing cores, we must consider three performance metrics [25]. The total time, load balance and congestion. Total time is the time to finish all the tasks. Each core should have same or similar task load to ensure TESN achieves load balance. By the way, we also need to consider the congestion situation of each core. In order to improve the performance of sink node, we need to achieve the following three objectives: to minimize the total processing time of all tasks, to balance the load situation of each execution core and to control each core's congestion. On one hand, it increases the overhead of communication between executing cores to achieve balanced load and low congestion. On the other hand, if only one core dealing all tasks, there are no communication overhead, but other cores will be idle, which reduces the processing efficiency of sink node greatly. Therefore, we need to establish a proper task allocation scheme [26], which aims at minimizes tasks' total processing time and the overhead of all cores. And also, the scheme should make each core with low congestion.

In recent years, researchers have presented many related algorithms and scheduling strategies on the application of TESN and task scheduling for multi-core processors in IoTs. Munir et al. [27] propose a heterogeneous hierarchical multi-core embedded wireless sensor networks architecture. The more efficient computational performance of multi-core embedded sensor nodes benefit compute-intensive tasks. But it lacks of proper tasks scheduling policy to achieve better performance. Zhou et al. [28] use multi-core processors into WSNs and propose a cache allocation algorithm which to provide suitable cache for parallel tasks. It improves the performance and utilization effectively of WSNs. But this algorithm does not consider the requirements of load balance. Muneeswari et al. [29] incorporates the load balancing agent for the multi-core processor based on the hard–soft processor affinity scheduling algorithm. It achieves the process migration between cores, although load balancing is reached, the communication consumption is too much. It has negative impact on the life-time of sink node. Zhang et al. [30] propose scheduling strategy in IP-core level. The authors use driftwood algorithm to build an optimized global scheduling strategy. But the main purpose of this work is to achieve the maximum use of testing resources to test the SoC. Useful real-time scheduling algorithms are introduced in [31,32]. Zheng et al. [31] propose a "per-priority" basis analysis scheme which computes the total time window at each priority level instead of each traffic-flow, and further propose a task mapping and priority assignment algorithm, in such a way that the hard time bounds are met with a reduced overhead of sink node. Lee et al. [32] propose adaptive scheduling algorithm for embedded multi-core real-time systems. The scheduler assigns tasks dynamically and crosses over global and local scheduler. It calls for the hardware support for scheduler. Digalwar et al. [33] present a real time scheduling algorithm for mixed task set on homogeneous multi-core platform. But the heterogeneous cores are not considered. These two works mainly focus on real-time strategy without considering the load balance and each core's congestion. Round-Robin (RR) task allocation algorithm and conventional Least-Connection (LC) task allocation algorithm are mentioned in [34,35]. The way how to allocate tasks to server between RR and LC is different. In RR, tasks are allocated to servers in sequence. However, LC will choose the server which has the least number of tasks to allocate. They are used for LVS (Linux Virtual Server).

In this paper, we design a Task-Efficient Sink Node (TESN) based on embedded multi-core SoC for IoTs. There are eight soft cores working in the sink node. One of them is the master core and others are slave cores. Each core is an independent execution unit. The master core assigns tasks and balances load for slave cores, and slave cores execute the accepted tasks from master core. The Weighted-Least Connection (WLC) task allocation strategy for master core of TESN which is proposed in our previous work [36]. Jiang et al. [37] present two multiplex network-adapted models of task allocation with load balancing: network layer-oriented allocation and agent-oriented allocation. They considers both link types and agents, and substantially extends the existing work by highlighting the effect of network layers on task allocation and load balancing. Guo et al. [38] proposed a trust dynamic task allocation algorithm. They designed a discrete particle swarm optimization (PSO) to generate a structure of the parallel coalitions and payoff functions to minimize the execution time of the tasks. Brutschy et al. [39] present a self-organized tasks allocation method, which is based on the delay experienced by the robots working on one subtask when waiting for input from another subtask. It does not require the robots to communicate. Yang et al. [40] proposed a modified version of binary particle swarm optimization (MBPSO) for the task allocation problem to obtain the best solution. It adopts a different transfer function and a new position updating procedure with mutation. In Dynamic weighted-least connection algorithm (DWLC), the master core also collects the real-time load information of each slave core. The information will be used to avoid congestion. In order to facilitate the communication between soft cores, the Shared-bus and mailbox architecture are designed for the communication between master core and salver cores. The *Xilinx V5* platform is used as the multi-core sink node and *MiCroBlaze* soft cores are used as processors in the experiments.

The main contributions of this paper are as follows:

(i) We design the architecture of sink node in IoTs based on eight cores. TESN uses the master and slave structure, it can greatly enhance the ability of processing tasks.
(ii) The proposed algorithm WLC, allocates the coming tasks properly. On one hand, it keeps the load balance of multi-core system and helps TESN to play the role of parallel execution. On the other hand, WLC reduces the communication between cores and ensures the majority of resource using for processing tasks.

(iii) We test the performance of TESN and WLC through both software simulations and hardware experiments. The results show that, TESN improves the computing performance of sink node significantly and WLC satisfies the requirements of the load balancing. TESN is suitable for large-scale data acquisition and processing in IoTs.

The remainder of this paper is organized as follows. The problem statement and motivation are described in Section 2. We introduce the hardware design of TESN with shared-bus and mailbox architecture in Section 3. In Section 4, TESN scheduling strategy is described. We propose WLC for heterogeneous architecture. In Section 5, we present algorithm design for our strategy. The algorithms include the multi-core communication algorithm, the multi-core data processing algorithm and DWLC algorithm. In Section 6, the performance results of our algorithms are evaluated with different core numbers and different task numbers on *Xilinx V5* platform compared with RR. We also use NS2 simulation environment to test the computing performance and load balance of TESN. Section 7 concludes this paper.

2. Problem statement and motivation

In IoTs applications, devices with different capabilities are interconnect for data transfer and data collection. With the amount of information and data keep increasing, the capabilities of computing [41] and resource of devices are limited in the applications. When we use the sink node with single-core, all of the data sent by other nodes must be collected and processed by the only one processing core. If the processing ability of single-core sink node cannot satisfy with the demand of large data, task delay and data missing may occur frequently, which impacts the performance of system seriously. On the other hand, the development of SoCs is limited in the ability to process large number of data. The improvement of the performance of SoC is limited by the technology and processes. The computing performance of sink node cannot be enhanced unlimitedly. Therefore the performance bottleneck of single-core sink node is becoming more and more prominent. Large-scale data collection and processing is crucial for IoTs. In order to improve the poor computing performance of single-core system, multi-core system has been widely used gradually. The aim is to speed up the processing time and enable the network make their resources available. In multi-core system, the collection and processing of data is allocated to multiple computing cores. Each core can perform tasks independently and cannot be affected by others. All tasks can be executed in parallel [42] at different cores. Without improving the single-core performance, the processing speed of the system can still be significantly improved, so as to satisfy the needs of large-scale data. Considering the size of a distributed IoTs network, the tasks allocation issue with the aim of improving network performance is essential, which is particular necessary for the real-time system.

Multi-core sink node can solve the problem about poor computing performance of single-core sink node. However, when tasks are executed on a multi-core system, they must be allocated and scheduled uniformly. Otherwise it may lead to the large load gap between cores, even lost tasks or re-execute tasks. If the load between cores cannot be balanced, the execution of the task is still similar to the serial execution. As a consequence, multi-core system cannot take full advantages of parallel execution. This phenomenon reduces the system performance greatly. On the other hand, in order to allocate tasks uniformly, communication between different cores is necessary. Moreover, with the number of cores increasing, the communication between cores becomes more and more frequent and consumes more resources, which does not exist in single-core system. Thus the energy consumption of multi-core sink node is significantly larger than that of single-core sink node. The extra energy consumption is coursed by the parallel execution of tasks and cores communication. Excessive power consumption will shorten the lifetime of multi-core sink node. Therefore, it is crucial to design a task scheduling algorithm based on multi-core sink node. It needs to ensure multi-core load balancing by allocating tasks reasonably as well as minimize the inter-core communication to reduce energy consumption.

To deal with the problem stated above, an eight cores sink node architecture named TESN and a task scheduling strategy named WLC are proposed.

3. TESN architecture

TESN processes tasks in parallel. It has eight executing cores and satisfies the requirements for processing large amounts of data. We realize TESN architecture on *Xilinx* platform. It is a Field-Programmable Gate Array (FPGA) chip which has several *MicroBlaze* processors. We can use necessary exterior devices to realize the embedded programming SoC of IoTs and improve the design flexibility. Fig. 1 shows the architecture of TESN.

All of the cores in TESN are realized by soft cores. As can be seen in Fig. 1, we use *MicroBlaze* to realize the eight executing cores, MB_0 to MB_7. The master core is MB_0, MB_1 to MB_7 are seven salver cores. MB_0 allocates tasks and balances the load for salver cores. And MB_1 to MB_7 are responsible for the cooperation executing tasks. When a new task arrives, it will be assigned to one of the seven slave cores according to the DWLC by master core. Once the task is allocated to a core, it should be finished on it. There is no task migration among cores in order to decrease multi-core communication cost.

In order to support synchronous work, some kinds of *IP-cores* are used to steady the architecture. We use *xps_timer* as the timer module to control the global time. plb_0 to plb_7 are eight local bus for each core. All of them are realized by *LogiCORE IP Processor Local Bus* (*PLB*) *v*4.6. They are responsible for the transfer instruction and information. We use Shared-bus architecture to reduce the energy consumption of multi-core communication. The seven mailbox modules, $mbox_0$ to $mbox_6$, which are realized by *mailbox*, are in charge of communication between salver cores and master core. $Bram_0$ to $Bram_7$ are the memory of each executing core, they are realized by *Bram_block*. MPMC is the conflict control module in the architecture, which is realized by *mpmc*. It controls the visiting of memory from soft cores through eight ports. The register data are stored in an independent BRAM. We use *Mdm* module for debugging, *Network Controler* module for wireless network communication.

Fig. 2 shows the IoTs architecture which TESN applied. TESN has powerful computing performance and it is responsible for collecting and processing data as the sink node. All of the sensor nodes send data to TESN by wireless. TESN connects to the IoTs through the wireless module named *Network Controler*. The master core receive tasks from the wireless module and allocates them to appropriate slave core. TESN implement the parallel execution of all tasks by multi-core. Its computing performance significantly improved. However, the communication between cores will consumption a part of CPU resources. With the increase of cores, the proportion of communication CPU resource consumption also rise quickly. Therefore, when the cores reaching a certain number, the computing performance of TESN will has no more obvious improvement. On the contrary, it will increase the power consumption greatly. Considering what has mentioned above, we adopt eight cores to build the architecture of TESN based on a lot of simulations.

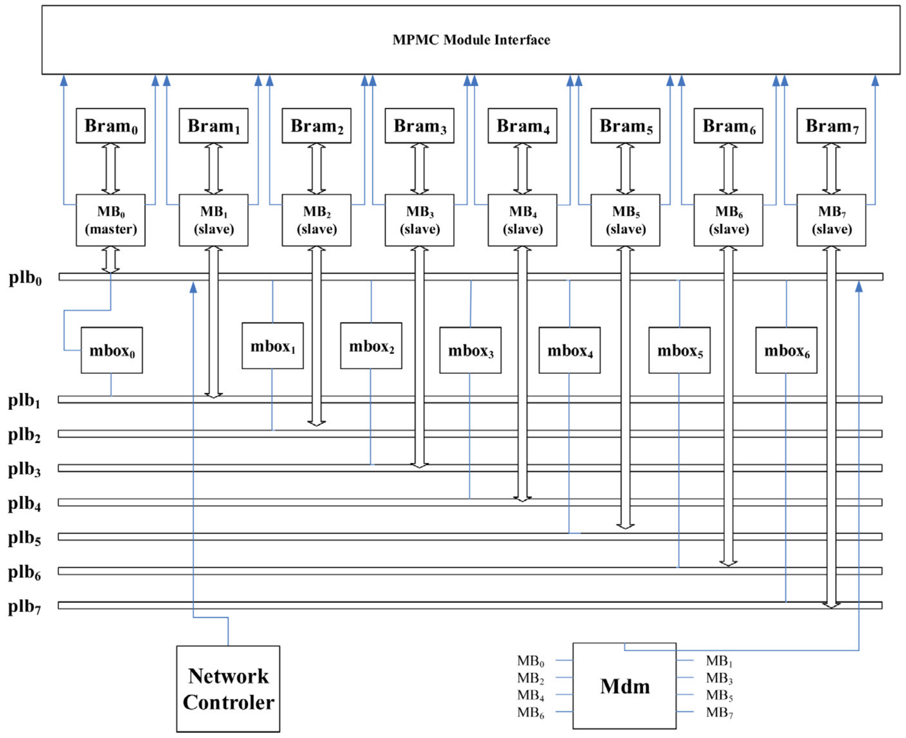

Fig. 1. Architecture of TESN.

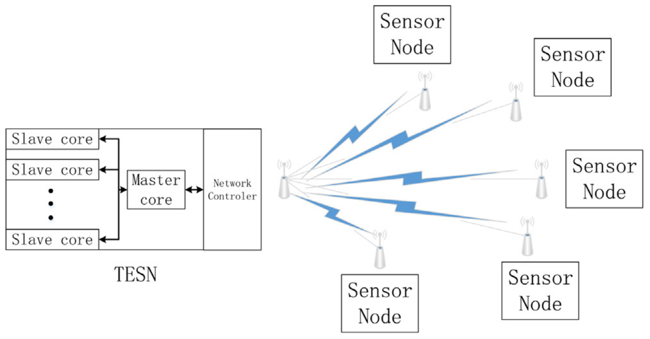

Fig. 2. IoTs architecture of TESN.

4. TESN scheduling strategy

Multi-core processor is designed to improve the thread-level parallelism for sink node. In order to make TESN processing speed faster and get higher computing performance, we should consider multi-core task allocation strategy, threads scheduling of slave cores and multi-core communication strategy. Next we introduce each part carefully.

4.1. *Multi-core task allocation strategy*

TESN is a heterogeneous architecture. The master core is used to receive tasks and allocate tasks to slave cores. Slave cores only responsible for dealing with tasks. For multi-core task allocation, tasks are allocated to each core's local task waiting queue by WLC.

We assume that $C_0, C_1, \dots, C_{N-1}$ are N soft cores in the multi-core processor. $T_0, T_1, \dots, T_{M-1}$ are M threads which need to be allocated. M_f is the number of finished threads. Every salve core writes its finished tasks number in the mailbox. Then, the master core reads the information from $mbox_0$ to $mbox_6$ and modifies the finished tasks number of all slave cores. The current tasks number of soft core i can be calculated by Eq. (1).

$$task_c[i] = task_g[i] - task_f[i] \qquad (1)$$

where, $task_g[i]$ is the allocated tasks number of soft core i, and the finished tasks number of soft core i is stored in $task_f[i]$. $taks_c$ is the current tasks number which need to be execute on each slave core. We assume that the total number of tasks is $tasks_n$, it can be described as Eq. (2).

$$tasks_n = task_g[i] + task_g[2] + \cdots + task_g[i] \quad (i = 1, 2, 3 \dots). \qquad (2)$$

Otherwise, the master core dynamically changes the value M with the total number of tasks in the multi-core processor. $load[i]$ is used to describe the load situation of soft core i. $congestion[i]$ reflects the congestion situation of soft core i, $cp[i]$ is the computing performance of soft core i.

We consider both current tasks number and computing performance of the slave core to allocate the coming tasks. The computing performance is proportional to the clock frequency. We assume that the clock frequency of soft core MB_1 is 150 MHz and the clock frequency of soft core MB_2 is 75 MHz. Then, the computing performance of MB_1 is twice then MB_2. The least computing performance will be stored as 1. We can get that $cp[1] = 2$ and $cp[2] = 1$. Therefore, the weighted value of each slave core is described in Eq. (3). The soft core which has the least value of $weight[i]$ is the proper slave core to deal with the coming tasks.

$$weight[i] = task_c[i]/cp[i]. \qquad (3)$$

4.2. *Threads scheduling of slave cores*

There are two main threads running on the slave core. One is communication thread and the other is data processing thread. Also there are two main threads running on the master core. One thread is communication thread and the other one is the task allocation thread. The thread status of the master core and slave

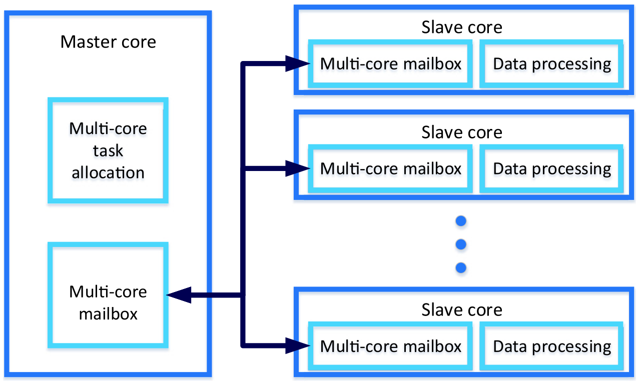

Fig. 3. Working threads of the master core and slave cores.

cores is shown in Fig. 3. We use timer to control the finishing moment of communication thread, which sends each slave core's finished task number and status to the master core.

In order to improve the performance of TESN, multi-core weighted-least connection task allocation strategy is proposed. Each core must be assigned a timer to provide a core's start tick for the purpose of responding to the interrupt to execute the process schedule. Xilkernel's executing unit supports the POSIX standard. As creating and destroying a thread costs less than scheduling process, threads are used to deal with data and communicate with other cores. We consider each core's processing ability and the number of unfinished tasks on each core to allocate the coming tasks.

4.3. *Multi-core communication strategy*

For the purpose of cooperative completing the tasks, all of the soft cores must communicate with each other. Fixed Access Unit (FAU), Mailbox mechanism and message queue are three commonly used multi-core communication strategies. In this paper, the multi-core communication strategy is realized by Mailbox mechanism. A soft core sends information to its corresponding mailbox and the receiving core can read the information from the mailbox. Both reading and writing operations are unblocked.

5. Scheduling algorithm design

According to Section 3, the schedule strategy can be divided into the following three modules:

(i) Timer interrupt module.
(ii) Multi-core thread switch module.
(iii) Multi-core weighted-least connection task allocation module.

5.1. *Timer interrupt module*

Xilkernel provides the timer function to deal with the time-related situation. This module is configurable. We may set it to input or output mode. According to internet data, load balance should be adjusted every 200 ms. If the frequency is too low, some cores can be idle. If the frequency is too high, communication can consume much resource and decrease processing speed. The load situation can be calculated in Eq. (4) and the congestion situation can be calculated in Eq. (5).

$$load = length\_queue/max\_queue \tag{4}$$

$$congestion = task\_queue/max\_queue \tag{5}$$

wait_queue is the tasks number in wait queue, *task_queue* is the length of current tasks number for this slave core and *max_queue* is the total length of tasks queue. We set the same timer for every soft core. the time of all soft cores should be synchronized. *idle* is the state of slave core. If the value of *idle* is set to 1, it means that this slave core finishes its current tasks. The state of this slave core will be send to the *mailbox* and the slave core waits the master core to allocation new tasks. When the timer expires, all of the slave cores send their own load situation to the *mailbox*. Then the master core collects these information and allocates the coming tasks. Timer interrupt module is an important part for WLC and DWLC.

Algorithm 1 Timer interrupt module

```
Input: $timer$, $data$, $type$, $task_f$ and $tasks_g$
Output: Task results
 1: procedure timer interrupt module
 2: while $timer > 0$ and $idle == 0$ do
 3:     while $task_f < tasks_g$ do
 4:         read the $type$ of data
 5:         if $type == 1$ then
 6:             The data will be executed on computing-intensive thread
 7:         else
 8:             The data will be execute on computing-general thread
 9:         end if
10:         $task_f ++$
11:         Check $timer$
12:     end while
13:     Reset $idle = 1$
14: end while
15: Sends load situation to mailbox
16: Master core reads the load situation
17: Master core calculates the weights
18: Master core allocates the coming tasks
19: Reset the value of timer
20: end procedure
```

All threads have the same permission to access resources. POSIX thread API is the basic interface that can be used by users. Xilkernel also provides extra interfaces. The threads can be operated through threads identifiers. The operating system supports the priority-driven scheduling and round-robin scheduling mechanism. Considering that thread switching resource cost is much lower than process switching, and Xilkernel supports POSIX well, we create lots of threads on a core, which have different functions.

Data processing thread and multi-core communication thread are two types of threads in slave cores. Data processing thread responsible for the task processing and calculating the performance of soft cores. Multi-core communication thread is used to send relevant cooperative information between master core and slave cores.

There are two types of data processing thread in WLC. computing-intensive thread and computing-general thread. The type value of computing-intensive thread is 1. It means that this task needs to be executed with more CPU time slices. The other type tasks will be executed with less CPU time slices. Algorithm 1 shows the implementation process of timer interrupt model in detail.

In Algorithm 1, we input the following params *data*, *type*, $task_f$ and $tasks_n$. We assume that *timer* is t and $tasks_n$ is n. Each slave core deals with its own tasks before the *timer* expired (Lines 2–3). When the slave cores dealing with tasks, they will read the *type* of each task firstly, and decide to use computing-intensive or computing-general thread for this task (Lines 4–12). After all of the current tasks finished, slave core reset *idle* to 1 (Line 13). Then, master core allocates tasks for seven slave cores (Lines 15–20). Therefore, the time complexity of this algorithm is $O(t * n)$.

5.2. Multi-core thread switch module

Algorithm 2 Multi-core communication

```
Input: $task_g$ and $task_f$
Output: $task_c$
1: procedure multi-core communication module
2: Initialize $nbyte = 0$
3: Initialize $sent = 0$
4: Initialize $num = 0$
5: while (($nbyte < TOTAL\_BYTE\_SIZE$)&&($num < TOTAL\_NUM\_SIZE$)) do
6:     Read data from mailbox by function $mbox\_read()$
7:     update $sent$
8:     $nbyte+ = sent$
9:     $num + +$
10: end while
11: if $task_f > 0$ then
12:     $task_c = task_g - task_f$
13: end if
14: end procedure
```

We update the global variables such as $task_c$, $task_f$, end, $idle$ etc. during the data processing thread. The master core collects the values of these variables to calculate the appropriate weight for task allocation. The multi-core communication thread is shown in Algorithm 2. *TOTAL_BYTE_SIZE* is the maximal bytes in a mailbox. And *TOTAL_NUM_SIZE* is the maximal bytes number in a mailbox.

Algorithm 2 works as follow. At the beginning, we initialize variables *nbyte*, *sent* and *num* (Lines 2–4). *nbyte* is the total reading bytes. *sent* is the reading bytes number in each operation. *num* is the total reading bytes number. The soft core continuously reads data from *mailbox* until the total size (Lines 5–10). Then it updates the current tasks number (Lines 11–13). The time complexity of this algorithm is $O(n)$.

5.3. Multi-core weighted-least connection task allocation module

Algorithm 3 Dynamic weighted-least connection

```
Input: $task_g$, $task_f$ and $cp$
Output: $core_{index}$
1: procedure dynamic weighted-least connection algorithm
2: while !$end$ do
3:     for $i \in [1, 7]$ do
4:         $sum+ = task_g[i]$
5:         $task_c[i] = task_g[i] - task_f[i]$
6:         $weight[i] = task_c[i]/cp[i]$
7:     end for
8:     if $sum >= tasksnumber$ then
9:         $end = 1$
10:    end if
11: end while
12: $core_{index}$=sort(int $task_c[i]$, int $se[]$)
13: end procedure
```

In this hardware platform architecture, each slave core has different computing performance, thus tasks should be allocated according to WLC. Task allocation is shown in Algorithm 3.

This algorithm works as follow. The master core computers the weight of each slave core and cumulate the value *sum* (Lines 3–7). When *sum* is greater than tasks number, master core will end the tasks allocation and sort the tasks for slaves. The time complexity of this algorithm is $O(n)$.

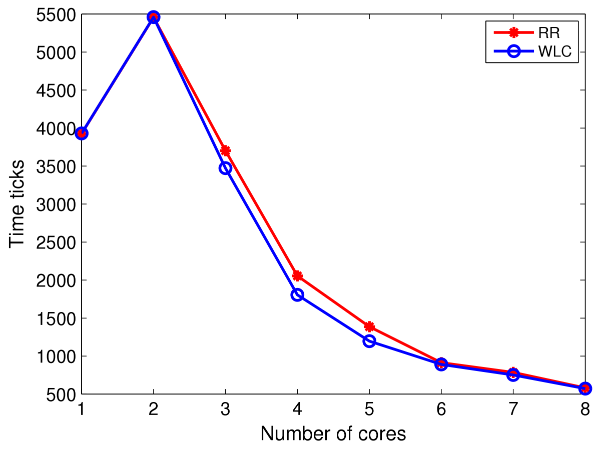

Fig. 4. Ticks time when dealing with 300 tasks using two algorithms.

6. Experiment results

6.1. Assumptions

Every slave core needs 10 ticks to finish a task. The maximum length of each slave cores task waiting queue is 50. The master core collects the load situation of all slave cores every 200 ms.

6.2. Metrics to evaluate processor's performance

ticks is the number of ticks taken to finish all tasks, *wait_len* is the length of the waiting queue, *max_len* is the maximum length of the waiting queue, *N* stands for the total task number, *task_num* is the number of current tasks needed to be dealt.

In order to evaluate our strategy's performance, we use following three metrics:

(i) *ticks*: It is used to show the total time that is needed to finish all tasks. The larger ticks is, the longer time is needed to finish all tasks.
(ii) *avg_load*: It is used to reflect the slave cores's load situation. It can be calculated by Eq. (6).
(iii) *avg_cong*: It is used to reflect the slave core's congestion. It can be calculated by Eq. (7).

$$avg\_load = \sum_{i=1}^{i-N} (wait\_len/max\_len)/N \tag{6}$$

$$avg\_cong = \sum_{i=1}^{i-N} (task\_num/max\_len)/N. \tag{7}$$

6.3. Experimental results

Fig. 4 shows the results of ticks. There are 300 tasks in the simulation. *Y* axis is the total time ticks and *X* axis is the number of cores. In Fig. 4, if only one soft core is active in the processor, the scheduling strategy is unefficient. The performance of RR and WLC are exactly the same. The master core is used to task allocation and other soft cores are slave cores, they are responsible for the task processing. So, if the number of cores is two, we have only on slave core, parallel processing is not available. Therefore, both of WLC and RR need to consume resources for tasks allocation. The time ticks will increase. With the further increase of cores, the scheduling strategy starts to work, more than one slave core deals with tasks in parallel. WLC can keeps better load balance

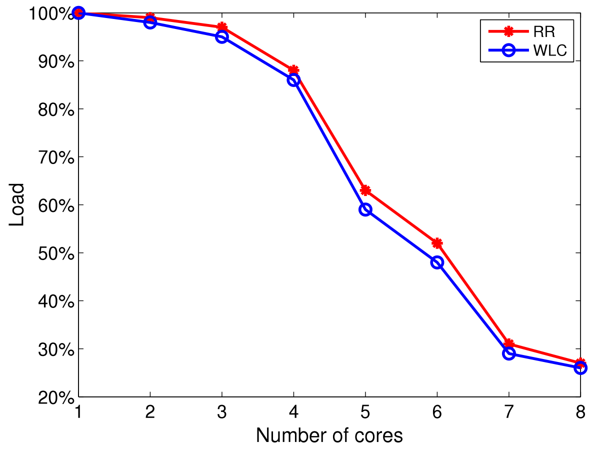

Fig. 5. Load situation based on WLC and RR with 300 tasks.

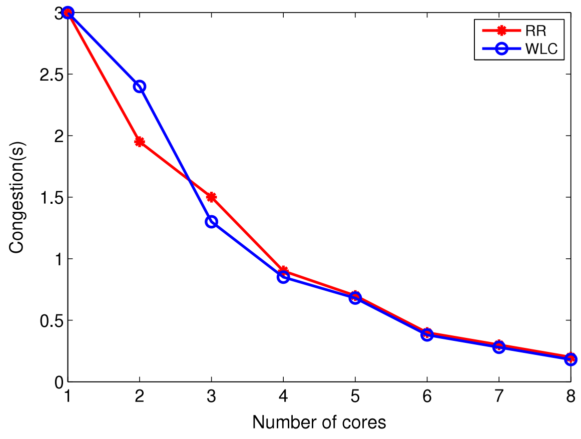

Fig. 6. Congestion situation based on WLC and RR with 300 tasks.

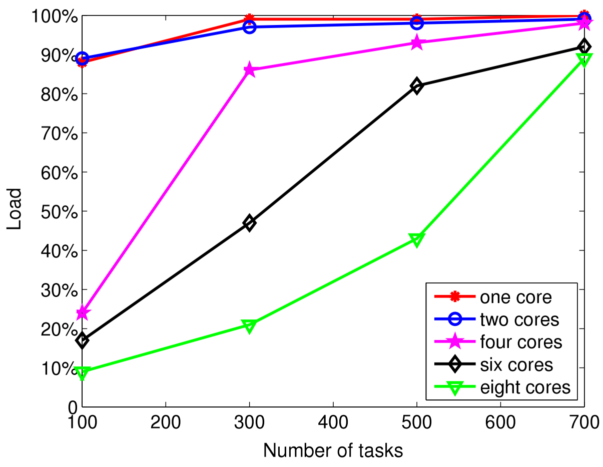

Fig. 7. Load of different cores number when task number is different.

than RR, so the time ticks of WLC will drop a little faster. We can see that when the number of soft cores is in the range of 3–6, the performance of WLC is much better than RR. As can be seen in Fig. 4, when the number of cores is large than 2, the time ticks begin to drop and the drop speed is more and more slowly. The reason of this phenomena is that different cores need to communicate with each other. The more cores TESN has, the larger proportion of resources that communication occupied. It means that, with the increase of cores number, the improvement of performance will be more and more slowly. So the lines of time ticks are more and more flat.

Fig. 5 shows the average load situation of cores when using two strategies. There are also 300 tasks in the simulation. The total number of tasks and the speed of coming tasks are fixed. Therefore, with the increase of cores, we can find that the cores average load decreases. Because more and more soft cores are joined in to share the coming tasks. When the number of cores is less than 4, the number of data processing soft cores is small. The speed of coming tasks is faster. They cannot share the workload for each other obviously. So the average load still keeps in the high level. With the further increase of soft cores, more soft cores are used to deal with tasks. And the speed of coming tasks is fixed. Therefore, the share effect will be more obvious, the average load reduce quickly. By the way, when we use the multi-core system, the communication between different cores occupies certain resources. They also has impact on the load situation. So the drop speed of load situation is also different. The weighted-least connection strategy has lower average load than that of round-robin algorithm. TESN is able to allocate computing resources more reasonable based on weighted-least connection strategy.

The congestion situation of TESN based on WLC and RR is shown in Fig. 6. There are also 300 tasks in the simulation. At the beginning, the system has only on core, differences algorithms has no influence on the tasks running. Therefore, they have the same congestion situation. If the number of soft cores is two, there is only one slave core in the processor. If the number of soft cores is two, there is only one slave core in the processor. In WLC, the salve core needs to calculate and send its load situation to master core and the master core also need to calculate the appropriate allocation decision. Even though it has one slave to choose, the process of calculation cannot be skipped. Both of the communication and calculation need consume much resource. However, we donąŕt need to consider about the information transfer and weight calculation in RR. The tasks are allocated in turn. It will saving a part of resources compared with WLC. So the performance of RR is better. When the number of cores continue to increase, WLC keeps better load balancing than RR, it has lower congestion situation.

Fig. 7 shows the cores' average load of TESN with different cores number. It is the results of the weighted-least connection strategy. The *X* axis is the number of tasks, and the *Y* axis is the average load of soft cores. In Fig. 7, we can see that the average load increases with the increasing task amounts. When the number of soft cores is one or two, there is only one data processing core. The average load keeps in the high level. Then with the increase of soft cores, the average load reduce quickly. Average load of cores is becoming smaller with the increase number of cores when the task number keeps invariant.

Fig. 8 shows the cores' average congestion situation of TESN with different cores number. It is the results of the weighted-least connection strategy. The *X* axis is the number of tasks, and the *Y* axis is the average congestion of soft cores. From Fig. 8, we can see that the average congestion increases with the increasing of task amounts. The single core system has the highest congestion situation. The multi-core system has more than one data processing cores. It has the ability to assign tasks well and reduce the congestion. Congestion load of cores is becoming smaller with the increase number of cores when the task number keeps invariant. As can be seen in Fig. 8, when the number of cores larger then 2, the drop speed of congestion is slower. The reason is that, when the system has more cores, the communication between different cores will be more frequently. TESN need to use more resource to deal with the communication. Therefore,

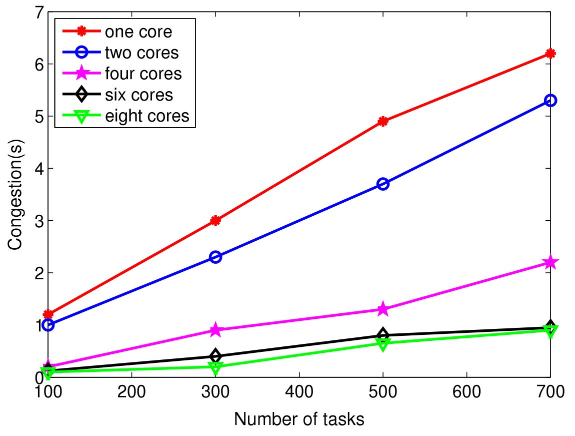

Fig. 8. Congestion of different cores number when task numbers vary.

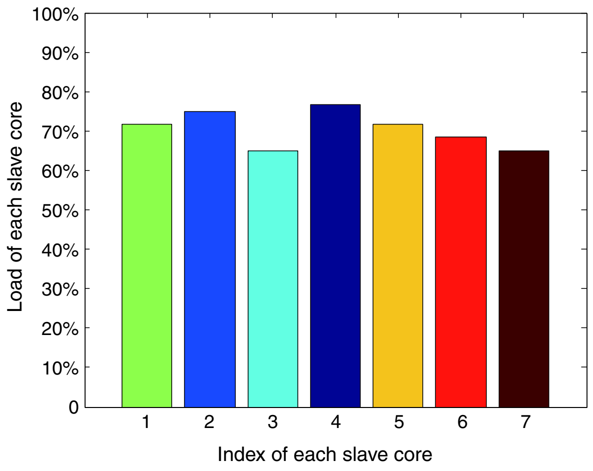

Fig. 10. Load situation of each slave core of TESN. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

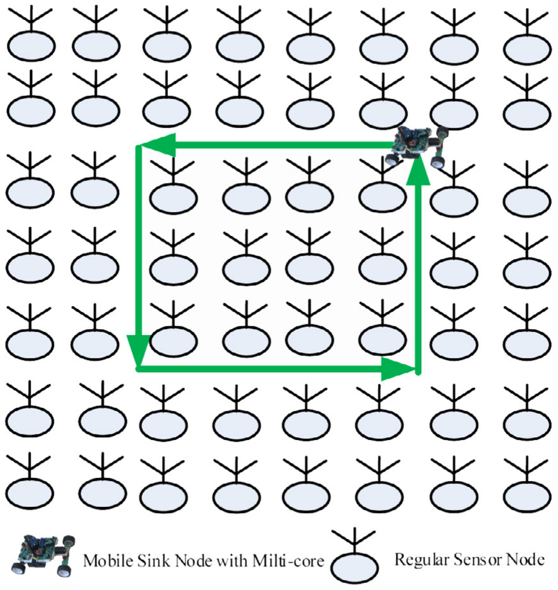

Fig. 9. The network topology for TESN in IoTs.

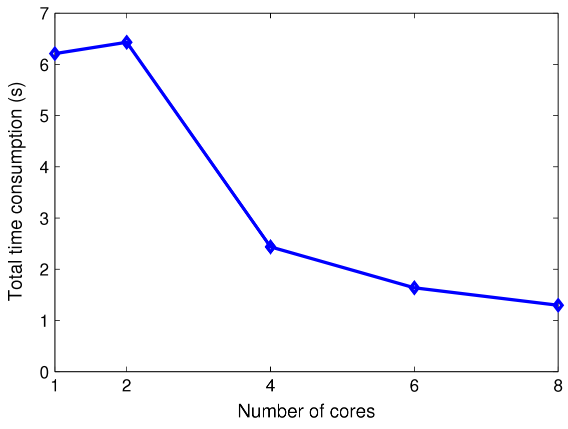

Fig. 11. Total time consumption for different cores number.

when the number of cores up to 8, TESN cannot get more obvious improvement of congestion situation.

In order to evaluate the performance of TESN, we deploy 64 regular sensor nodes which are according to the 8*8 lined up in the topology. The gap between the nodes is 100 m. The communication radius of each node is set to 200 m. The TESN with multi-core is moving along the trajectory of the arrow in the network topology. Each sensor node collects the environmental data and sends them to the sink node. TESN is responsible for collecting all the environmental data from the sensor nodes, which is shown in Fig. 9. We use NS2 to simulate this process and analyze the information. The wireless communication between nodes is based on flooding protocol. In order to simulate TESN, we set up eight cores in the mobile sink node. One of the cores is treated as the master core. Others are slave cores. The master core allocates the packages, which are collected from the regular sensor nodes. We assume that a package represents a task. All slave cores only receive the packages from master core and deal with tasks. Each slave core needs 100 ms to finish a task. A timer is employed to control the moment of task scheduling. When the timer expires, all slave cores send their state information to the master core. Then, the master core calculates the weights of all slave cores and choose the most appropriate core to assign task.

Fig. 10 shows the load situation of 7 slave cores. Different colors represent different soft cores. The total number of tasks is 300. It is achieved by the proportion of the number of current tasks in the queue and the length of queue, when the system runs stability. In Fig. 10, each slave core has the similar load situation. WLC is able to balance the load of each slave core and assign tasks to the appropriate one.

Fig. 11 illustrates the total time consumption for different cores number. The total number of tasks is 300. *X* axis expresses the number of soft cores and *Y* axis expresses the time consumption for running all tasks. In the experiment, we set the number of soft cores to 1, 2, 4, 6 and 8. All experiments based on the WLC task scheduling algorithm. It reflects the CPU efficiency of TESN. The less time consumption is, the higher CPU efficiency is. In Fig. 11, TESN with 8 soft cores adopt the least time to complete all the tasks, which means that it has the fastest speed for tasks processing. The time consumption presents a decline tread with the increase of soft cores basically. When the soft cores number is 2, because of the WLC task scheduling algorithm, TESN has one master core and one slave core. The master core deals with task scheduling and the salver core is responsible for data processing, they also need to communicate with each other. So the time consumption of double cores system is a certain higher than that of single core

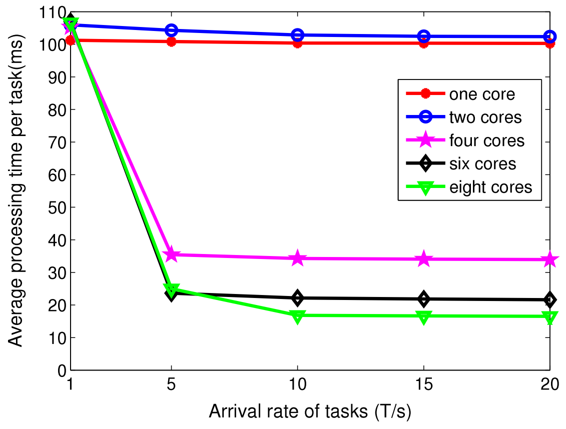

Fig. 12. Task processing rate for different cores number.

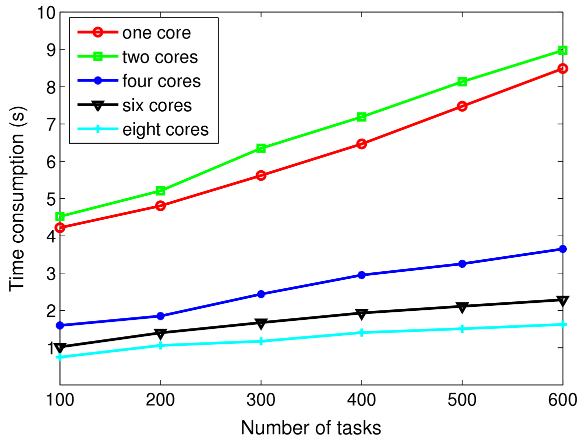

Fig. 13. Task processing time for different cores number.

system. When the soft cores number is 4, there are three slave cores and one master core in the TESN. Because of the communication between soft cores consumes CPU resources, the CPU efficiency is improved closer than three times. So the time consumption is reduced to nearly a third of single core system. Then, with the increase of soft cores, the CPU efficiency also has been further enhanced.

Fig. 12 shows the task processing rate for different cores number. *X* axis expresses the arrival rate of tasks. *Y* axis expresses the average processing time per task. Every soft core needs 100 ms to finish a task. The total number of tasks is 300. In Fig. 12, TESN with eight soft cores has the lowest average processing time per task when it is full load. The single core system only processing the tasks by itself, so the average processing time per task basically remains 100 ms. The double cores system has one master core and one slave core. It also only has one core to process tasks. They also need extra time to communicate with each other, so the average processing time per task is higher than single core system. When the arrival rate of tasks is 1 T/s, it means that there is one task arrive in a second. At this time, all of them do not reach saturation. Compared with single core system, all multi-core systems need to allocate tasks and communicate between cores. They will consume extra time and the average processing time per task is a little higher then 100 ms. When the arrival rate of tasks is 5 T/s, the average processing time per task for more then four cores systems are quickly reduced. All of the slave cores in the four cores and six cores systems are used to process tasks, they are in the full load working condition. So with the increase of arrival rate of tasks, the average processing time per task basically keeps the same. The six cores system has more slave cores than four cores system, so it has lower average processing time per task. However, there are two idle soft cores in TESN. Thus, the load is not full. With the further increase of tasks arrival rate, the average processing time per task will continually reduced and also keeps the same when it reaches saturation.

In Fig. 13, the experiment shows the total time consumption increases with the packet number. *X* axis expresses the number of tasks which ranges from 100 to 600. *Y* axis expresses the time consumption of running tasks. We compared the time consumption of five different numbers of cores in the mobile sink node. It is obviously that TESN with eight cores utilizes the least time to run all the task, compared with time consumption of less cores. Furthermore, four cores and six cores systems speed up the running speed closed to three and five times respectively compared with that of one core system. Due to using multi-core system, the tasks are processed in parallel and the CPU efficiency is increased, thus the processing speed is improved greatly. However, when the number of cores is two, the master core has to take some time to deal with task scheduling and the slave node takes charge of data processing. Therefore, the total time consumption is longer than one core system. With the number of packets increasing, the rate of time consumption in TESN with eight cores is smaller than others. It has the ability to assign tasks well and speed up processing time. Thus, the performance of TESN with eight soft cores is stable and reliable even if the task number is large.

Following conclusions can be reached from the simulation results:

(i) In the master–slave architecture on IoTs, when the number of soft cores is one or two, the congestion situation of multi-core processor is increased, but the processing speed is decreased. The processing time grows larger because of the multi-core communication cost. If there are more than two soft cores in the multi-core processor, the processing time is decreased and the average load situation and average congestion situation is improved obviously.

(ii) The performance of WLC based on multi-core processor is better than RR when the number of soft cores is more than two. It helps to keep load balance well between slave-cores in IoTs.

(iii) TESN significantly promotes the processing speed of sink node in IoTs. It effectively satisfies the needs of large-scale data calculation and processing in IoTs. Therefore, TESN can be widely used in a lot of IoTs applications, such as sensor networks, embedded systems, intelligent control etc.

7. Conclusion

In this paper, we design TESN, a task-efficient sink node based on embedded multi-core SoC for IoTs. TESN has eight soft cores and it has the master–slave architecture. In order to achieve lower congestion and better load balance for the parallel computing, we propose WLC to allocate the tasks for slave cores. WLC considering the processing speed, tasks number and real-time processing situation of each slave core, it obviously improves the computational efficiency of TESN. The experiments are realized on *Xilinx V5* platform. We compares the time ticks, load situation and congestion situation of WLC and RR task schedule strategies, which are aimed at improving the computing performance of TESN. We also test the load balance and CPU efficiency of TESN in NS2 simulation environment. The experiment results show that, the performance of WLC is better than RR when the number of soft sore is more than two and TESN keeps well efficiency and load balance. The implementation of TESN in

IoTs achieves lower congestion, load balance and speeds up the processing time of sensor nodes observably, which improves the network performance and contributes to the multi-core system development of IoTs.

We will further focus on the application of multi-core sink node in sensor networks of IoTs. On one hand, TESN will lead to the increase of power consumption and high temperature. Therefore, the appropriate cooling strategy is worth to study. More cores means faster and less cores means low power. In order to achieve the best comprehensive performance, it is important to further research on the number of core for different IoTs applications through software. On the other hand, this paper only considering the task allocation, our next step will try to combine the task allocation with the routing and deployment method to further improve the performance of heterogeneous sensor networks in IoTs.

Acknowledgments

This work is partially supported by Natural Science Foundation of China under Grant (Grant No. 61672131), and the Fundamental Research Funds for the Central Universities (Grant No. DUT16QY27).

References

[1] A. Whitmore, A. Agarwal, L. Da Xu, The Internet of things-a survey of topics and trends, Inf. Syst. Front. 17 (2) (2015) 261–274.

[2] G. Han, J. Jiang, C. Zhang, T.Q. Duong, M. Guizani, G.K. Karagiannidis, A survey on mobile anchors assisted localization in wireless sensor networks, IEEE Commun. Surv. Tutor. 18 (3) (2016) 2220–2243.

[3] D. Zhang, D. Zhang, H. Xiong, L.T. Yang, V. Gauthier, Nextcell: predicting location using social interplay from cell phone traces, IEEE Trans. Comput. 64 (2) (2015) 452–463.

[4] T. Qiu, Y. Lv, F. Xia, N. Chen, J. Wan, A. Tolba, Ergid: An efficient routing protocol for emergency response Internet of things, J. Netw. Comput. Appl. 72 (9) (2016) 104–112.

[5] S. Lu, Z. Wang, Joint optimization of power allocation and training duration for uplink multiuser mimo communications, in: Proceeding of the 2015 IEEE Wireless Communications and Networking Conference (WCNC 2015), New Orleans, LA USA, IEEE, 2015, pp. 322–327. March 9-12.

[6] T. Qiu, L. Feng, H. Jiang, W. Sun, Queueing model analysis and scheduling strategy for embedded multi-core soc based on task priority, Comput. Electr. Eng. 39 (1) (2013) 24–33.

[7] T. Qiu, D. Luo, F. Xia, N. Deonauth, W. Si, A. Tolba, A greedy model with small world for improving the robustness of heterogeneous Internet of things, Comput. Netw. 101 (2016) 127–143.

[8] S. Lu, Z. Wang, Z. Wang, S. Zhou, Throughput of underwater wireless ad hoc networks with random access: a physical layer perspective, IEEE Trans. Wireless Commun. 14 (11) (2015) 6257–6268.

[9] G. Han, J. Shen, L. Liu, L. Shu, Brtco: A novel border line recognition and tracking algorithm for continuous objects in wireless sensor networks, IEEE Syst. J. PP (08) (2016) 1–10.

[10] A. Salama, C. Binnig, T. Kraska, E. Zamanian, Cost-based fault-tolerance for parallel data processing, in: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM, 2015, pp. 285–297.

[11] L.D. Xu, W. He, S. Li, Internet of things in industries: A survey, IEEE Trans. Ind. Inf. 10 (4) (2014) 2233–2243.

[12] J.A. Stankovic, Research directions for the Internet of things, IEEE Internet Things J. 1 (1) (2014) 3–9.

[13] T. Bonald, J. Roberts, Enhanced cluster computing performance through proportional fairness, Perform. Eval. 79 (2014) 134–145.

[14] G. Han, L. Wan, L. Shu, N. Feng, Two novel doa estimation approaches for real time assistant calibration system in future vehicle industrial, IEEE Syst. J. PP (6) (2015) 1–12.

[15] M. Kondo, T. Yokogawa, Y. Sato, K. Arimoto, High-speed performance evaluation of a large-scale shared-bus digital system, Electron. Commun. Japan 98 (10) (2015) 57–66.

[16] V.T. Ravi, G. Agrawal, Performance issues in parallelizing data-intensive applications on a multi-core cluster, in: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID 2009), Shanghai, China, IEEE Computer Society, 2009, pp. 308–315. May 18-21.

[17] S. Bernabe, S. Sanchez, A. Plaza, S. Lopez, J.A. Benediktsson, R. Sarmiento, Hyperspectral unmixing on gpus and multi-core processors: A comparison, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 6 (3) (2013) 1386–1398.

[18] A. Saifullah, J. Li, K. Agrawal, C. Lu, C. Gill, Multi-core real-time scheduling for generalized parallel task models, Real-Time Syst. 49 (4) (2013) 404–435.

[19] C. Chen, S.C. Mukhopadhyay, C. Chuang, M. Liu, J. Jiang, Efficient coverage and connectivity preservation with load balance for wireless sensor networks, IEEE Sens. J. 15 (1) (2015) 48–62.

[20] S. Xie, Y. Wang, Construction of tree network with limited delivery latency in homogeneous wireless sensor networks, Wirel. Pers. Commun. 78 (1) (2014) 231–246.

[21] R. Hu, Y. Qian, An energy efficient and spectrum efficient wireless heterogeneous network framework for 5g systems, IEEE Commun. Mag. 52 (5) (2014) 94–101.

[22] D. Ergu, G. Kou, Y. Peng, Y. Shi, Y. Shi, The analytic hierarchy process: task scheduling and resource allocation in cloud computing environment, J. Supercomput. 64 (3) (2013) 835–848.

[23] B. Jedari, L. Liu, T. Qiu, A. Rahim, F. Xia, A game-theoretic incentive scheme for social-aware routing in selfish mobile social networks, Future Gener. Comput. Syst. (2016) in press.

[24] Z. George, E. Voroshazi, C. Lindqvist, R. Kroon, W. Zhuang, E. Wang, P. Henriksson, A. Hadipour, M.R. Andersson, Improved performance and life time of inverted organic photovoltaics by using polymer interfacial materials, Sol. Energy Mater. Sol. Cells 133 (2015) 99–104.

[25] H. Song, D.B. Rawat, S. Jeschke, C. Brecher, Cyber-physical Systems: Foundations, Principles and Applications, Morgan Kaufmann, 2016.

[26] J. Youn, I. Park, M. Sunwoo, Heuristic resource allocation and scheduling method for distributed automotive control systems, Int. J. Automot. Technol. 14 (4) (2013) 611–624.

[27] A. Munir, A. Gordon-Ross, S. Ranka, Multi-core embedded wireless sensor networks: Architecture and applications, IEEE Trans. Parallel Distrib. Syst. 25 (6) (2014) 1553–1562.

[28] B.H. Zhou, D.P. Yao, Research on multicore node performance in wireless sensor network, in: Proceedings of the Advanced Materials Research, Dalian, China, Vol. 821, Trans. Tech. Publ., 2013, pp. 1429–1433. August 24-25.

[29] G. Muneeswari, K. Shunmuganathan, Agent based load balancing scheme using affinity processor scheduling for multicore architectures, WSEAS Trans. Comput. 55 (3) (2011) 247–258.

[30] J. Zhang, W. Cai, C. Wang, The implementation of the global scheduling strategy, in: Proceedings of the 2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT-2010), Shanghai, China, IEEE, 2010, pp. 1970–1972. November 1-4.

[31] S. Zheng, A. Burns, Schedulability analysis and task mapping for real-time on-chip communication, Real-Time Syst. 46 (3) (2010) 360–385.

[32] L. Lee, H. Chang, W. Luk, An adaptive embedded multi-core real-time system scheduling, in: Proceedings of the 2nd International Conference on Ubiquitous Computing and Multimedia Applications (UCMA 2011), Daejeon, Korea, Springer, 2011, pp. 263–272. April 13-15.

[33] M. Digalwar, P. Gahukar, S. Mohan, Design and development of a real time scheduling algorithm for mixed task set on multi-core processors, in: Proceedings of the 2014 7th International Conference on Contemporary Computing (IC3 2014), Noida, India, IEEE, 2014, pp. 265–269. August 7-9.

[34] W. Chen, Y. Zhang, Z. Xiong, Research and realization of the load balancing algorithm for heterogeneous cluster with dynamic feedback, J. Chongqing Univ. 2 (2010) 014.

[35] H. Wang, Z. Fang, G. Qu, X. Zhang, Y. Zhang, A novel weight distrithm scheduling algorithm, J. Inf. Comput. Sci. 8 (2011) 1235–1244.

[36] T. Qiu, F. Liu, F. Xia, R. Qiao, A new schedule strategy of embedded multi-core soc, in: Proceedings of the 5th FTRA International Conference on Computer Science and its Applications (CSA 2013), Danang, Viet nam, Springer, 2013, pp. 43–49. December 18-21.

[37] Y. Jiang, Y. Zhou, Y. Li, Reliable task allocation with load balancing in multiplex networks, ACM Trans. Auton. Adapt. Syst. (TAAS) 10 (1) (2015) 3.

[38] W.Z. Guo, J.Y. Chen, G.L. Chen, H.F. Zheng, Trust dynamic task allocation algorithm with nash equilibrium for heterogeneous wireless sensor network, Secur. Commun. Netw. 8 (10) (2015) 1865–1877.

[39] A. Brutschy, G. Pini, C. Pinciroli, M. Birattari, M. Dorigo, Self-organized task allocation to sequentially interdependent tasks in swarm robotics, Auton. Agents Multi-Agent Syst. 28 (1) (2014) 101–125.

[40] J. Yang, H. Zhang, Y. Ling, C. Pan, W. Sun, Task allocation for wireless sensor network using modified binary particle swarm optimization, IEEE Sens. J. 14 (3) (2014) 882–892.

[41] V. Mauch, M. Kunze, M. Hillenbrand, High performance cloud computing, Future Gener. Comput. Syst. 29 (6) (2013) 1408–1416.

[42] M. Baklouti, M. Ammar, P. Marquet, M. Abid, J.-L. Dekeyser, A model-driven based framework for rapid parallel soc fpga prototyping, in: Proceedings of the 2011 22nd IEEE International Symposium on Rapid System Prototyping: Shortening the Path from Specification to Prototype, RSP-2011, Karlsruhe, Germany, IEEE, 2011, pp. 149–155. May 24-27.

Tie Qiu received Ph.D. and M.Sc. from Dalian University of Technology (DUT), in 2005 and 2012, respectively. He is currently Associate Professor at School of Software, Dalian University of Technology, China. He was a visiting professor at electrical and computer engineering at Iowa State University in US (Jan. 2014–Jan. 2015). He serves as an Associate Editor of IEEE Access and Computers & Electrical Engineering (Elsevier journal), an Editorial Board Member of Journal of Advanced Computer Science & Technology, a Guest Editor of Computers and Electrical Engineering (Elsevier journal) and Ad Hoc Networks (Elsevier journal), a Program Chair of iThings2016, a TPC member of Industrial IoT15, AIA13, EEC14, EEC15, EEC16 and ICSN16, a Workshop Chair of CISIS13 and ICCMSE15. He has authored/co-authored 7 books, over 50 scientific papers in international journals and conference proceedings. He has contributed to the development of 3 copyrighted software systems and invented 10 patents. He is a senior member of China Computer Federation (CCF) and a Senior Member of IEEE and ACM.

Aoyang Zhao received B.E. from Dalian University of Technology, China, in 2014. He is currently Master Student in School of Software, Dalian University of Technology (DUT), China. He used to participate in "Open source hardware and embedded computing contest 2012" and won the National first prize. He has authored one scientific paper in international conference. He is an excellent graduate student of DUT and has been awarded several scholarships in academic excellence and technology innovation. He is a Student Member of IEEE. His research interests cover embedded system, distributed computing and internet of things.

Ruixin Ma received Ph.D. and M.Sc. from Dalian University of Technology (DUT), in 2003 and 2012, respectively. He is currently Associate Professor at School of Software, Dalian University of Technology, China. He has authored/co-authored 7 books, over 10 scientific papers in international journals and conference proceedings. He has contributed to the development of 10 copyrighted software systems.

Victor Chang is an Associate Professor in Information Management and Information Systems and also a Director of Ph.D. Program at International Business School Suzhou (IBSS), Xi'an Jiaotong Liverpool, China. Dr. Victor Chang was a Senior Lecturer in Computing at Leeds Beckett University, UK and a Visiting Researcher at the University of Southampton, UK. He has been a technical lead in web applications, web services, database, grid, cloud, storage/backup, bioinformatics, financial computing which subsequently have become his research interests. Victor has also successfully delivered many IT projects in Taiwan (place of birth), Singapore, Australia, and the UK since 1998. Victor is experienced in a number of different IT subjects and has 27 certifications with 97% on average. He completed PGCert (Higher Education, University Greenwich, 2012) and Ph.D. (C.S, University of Southampton, 2013) within four years while working full-time, whereby the distance between his work and research is about hundreds of miles away. He has over 100 published peer-reviewed papers, including several high-quality journals up-to-date.

Fangbing Liu received B.E. from Dalian University of technology, China, in 2013. She is currently a master student in Department of Computer Science, Tsinghua University, China. She used to participate in "Open source hardware and embedded computing contest 2011" and won the National Third Prize. She has authored one scientific paper in international conference. She is an excellent graduate student of Dalian University of technology and has been awarded several scholarships in academic excellence and technology innovation. Her research interests cover embedded system, distributed computing and internet of things.

Zhangjie Fu received his Ph.D. in computer science from the College of Computer, Hunan University, China, in 2012. He is currently an Associate Professor at the School of Computer and Software, Nanjing University of Information Science and Technology, China. His research interests include Cloud & Outsourcing Security, Digital Forensics, Network and Information Security. His research has been supported by NSFC, PAPD, and GYHY. Zhangjie is a member of IEEE, and a member of ACM.