TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 07/10 pp72–84
DOI: 10.26599/TST.2019.9010040
Volume 26, Number 1, February 2021

WAEAS: An Optimization Scheme of EAS Scheduler for Wearable Applications

Zhan Zhang, Xiang Cong, Wei Feng*, Haipeng Zhang, Guodong Fu, and Jianyun Chen

Abstract: The rapid development of wearable computing technologies has led to an increased involvement of wearable devices in the daily lives of people. The main power sources of wearable devices are batteries; so, researchers must ensure high performance while reducing power consumption and improving the battery life of wearable devices. The purpose of this study is to analyze the new features of an Energy-Aware Scheduler (EAS) in the Android 7.1.2 operating system and the scarcity of EAS schedulers in wearable application scenarios. Also, the paper proposed an optimization scheme of EAS scheduler for wearable applications (Wearable-Application-optimized Energy-Aware Scheduler (WAEAS)). This scheme improves the accuracy of task workload prediction, the energy efficiency of central processing unit core selection, and the load balancing. The experimental results presented in this paper have verified the effectiveness of a WAEAS scheduler.

Key words: wearable devices; heterogeneous multicore system; big.LITTLE architecture; task scheduling

1 Introduction

There has been a significant involvement of wearable devices in the daily lives of people with the rapid advancement of wearable technologies, and they have been used in a broad range of applications, such as healthcare, entertainment, as well as various military and industrial applications. The prime focus of typical wearable applications is on sensing, collecting, and uploading physiological, environmental, and location data via wireless body area networks[1] or wireless sensor networks[2]. In military, healthcare, and emergency fields, the satisfactory/optimum operation of wearables is needed for a long time to help people cope with all kinds of dangerous situations. As wearables are powered by batteries, energy efficiency has become a key concern.

However, there are still many problems that need to be further addressed, optimized, and improved.

(1) Since the years the first lithium-ion batteries were invented, advancements in battery energy density lag far behind than the changes in hardware performance, and next generation batteries are still under development. Owing to the restricted size of embedded wearables, the battery capacity is very limited. In order to extend the battery life of wearables, it is necessary to reduce the power consumption of wearables in hardware/software coschedule mode.

(2) The usage scenarios of wearables generally include many human-computer interaction operations, which require a real-time response. In order to reduce the response time of wearables, it is essential to continuously increase the power consumption of system. Therefore, the optimization of the system becomes necessary as per the characteristics of wearable applications while extending the battery life of wearables as much as possible.

(3) In wearables, the processor consumes a significant portion of the total energy. Heterogeneous multicore processors, which provide high performance and low-power consumption, have garnered a lot of

• Zhan Zhang, Xiang Cong, Wei Feng, Haipeng Zhang, and Guodong Fu are with the School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China. E-mail: zhangzhan@hit.edu.cn; {congxiang; fengwei, zhanghaipeng, fuguodong}@ftcl.hit.edu.cn.
• Jianyun Chen is with Beijing Information Technology Institute, Beijing 100192, China.
∗ To whom correspondence should be addressed.
Manuscript received: 2019-05-24; accepted: 2019-08-27

attention and development in recent years. Many researches have optimized the scheduling strategies of heterogeneous processors to improve performance and energy efficiency, but few of them had addressed the Energy-Aware Scheduler (EAS) strategies of wearable systems. Due to the difference of usage scenarios between wearables and general-purpose devices, it is necessary to research on the low-power technologies for wearable applications.

The objective of this paper is to study and develop the wearables that provide complex functions and are sensitive to response time and battery life (e.g., wearables in military and emergency fields), and research the task scheduling strategies of heterogeneous multicore processor for wearable applications.

2 Related Work

The selection strategy of a Central Processing Unit (CPU) is an important part of a task scheduler in the heterogeneous multicore systems. As there are a large number of low-load periodic tasks in the wearable application scenarios, the selection of low-power processors for such tasks is the most appropriate thing to do, and low-power coprocessor[3] is an example of them. Priyantha et al.[4] used a low-power coprocessor to coordinate with the main processor, extended the time in low-power idle state of the main processor by assigning the low-load tasks, such as context awareness, into the coprocessor, and reduced power consumption by two orders of magnitude. Lane et al.[5] proposed the ZOE wearable platform, which is based on Intel Edison SoC, using a dual-core 500 MHz CPU as the main processor and a 100 MHz MCU as the low-power coprocessor, and achieved an efficient heterogeneous computing platform. Since the coprocessor does not have all the computing capacity like the main processor, it may not be suitable for carrying out all type of tasks, such as activity recognition, by using the machine learning algorithm[6]. At present, the heterogeneous multicore processor technology[7], which integrates cores with different advantages together to improve system performance, such as big.LITTLE architecture, has emerged as the development trend of wearables. The research works on task scheduling in heterogeneous multicore systems can learn from the coprocessor technology.

For latency-sensitive tasks, multicore scheduling strategy[8] focuses on how to place such tasks on the appropriate CPU core while meeting the performance demand of tasks. Wu and Ryu[9] introduced Best Speed Fit for Earliest Deadline First (BSF-EDF), which is a task assignment method in a heterogeneous processor system based on Earliest Deadline First (EDF)[10]. When assigning tasks, BSF-EDF selects the lowest speed idle CPU whose performance capacity is no less than the utilization of the task to perform the task with the highest priority, instead of selecting the CPU with the highest speed like the traditional scheduling algorithm, which reserves high-speed processors for the following tasks, improves the overall performance, and reduces the power consumption of the system. Khan et al.[11] proposed an offline thermal-aware scheduling algorithm to improve the scheduling strategy in multicore systems. As the high temperature of devices will result in performance degradation and energy waste, task migration is a common method to keep the temperature within thermal constraints in multicore systems, which migrates tasks to the CPU at a lower temperature, thereby improving the reliability and stability of the system and reducing the power consumption of CPUs.

Workload balancing and task migration are also the important features of multicore scheduling algorithm. Cheng[12] proposed a Heterogeneous MultiProcessor Dynamics Balance (HM-PDB) energy-saving scheduling framework based on the HMP scheduler. This framework dynamically adjusts the task migration thresholds on different processors based on the workload change, achieves load balancing and power consumption reduction with guaranteeing performance and fairness, and is more suitable for wearable application scenarios. Huang[13] divided application into three stages (pre-process, task executing, and post-process) from the perspective of hardware and software co-scheduling, coordinated with one master core and four slave cores for task scheduling, and improved the system throughput. Gao[14] predicted the application demand for CPU resource by dynamically sensing the load changes of tasks, thereby assigning an optimal time of the logical CPU to the application, and improved the performance and energy efficiency of the system.

Through the above investigation, we should make full use of the different characteristics, including heterogeneous cores, the adjustable CPU frequency, and the low-power consumption of CPU in idle state, to optimize the task scheduler. This paper address the dynamic task scheduling problem in heterogeneous multicore wearable systems and aim to improve the performance energy efficiency of heterogeneous processors from the following aspects.

(1) Optimization of CPU Core Selection: On the promise of fulfilling the deadline constraints and the demands for performance, tasks should be assigned to suitable CPU cores to reduce power consumption.

(2) Improvement of CPU Frequency Scaling: On the promise of fulfilling the task demands, the scheduler should set CPU to a suitable frequency and decrease the frequency scaling times to reduce the power consumption and response latency brought by frequency scaling.

(3) Optimization of CPU C-State-Setting: Based on the changes of system workload, the scheduler should set suitable CPU cores into the idle state. Also, it needs to set optimal c-state level to avoid unnecessary state switching between active and idle and reduce the unnecessary power consumption generated by CPU wakeup and CPU wakeup latencies.

3 Wearable-Application-Optimized Energy-Aware Scheduler (WAEAS)

We assumed that the wearable platform is equipped with a range of sensors and provides various functions such as GPS positioning, physical health monitoring, and environmental monitoring, etc. The wearable hardware platform is also equipped with heterogeneous multicore processor, RAM, flash memory, internal battery, LCD, communication modules, etc. The processor is based on ARM big.LITTLE architecture and consists of several high-performance cores and several low-power cores. All the cores can be active at the same time and have access to the same memory regions. The platform also implements general applications, such as Graphical User Interface (GUI), audio player, text editor, etc. however, this paper mainly aims on the workloads of wearable applications. Our previous work[15] provides a detailed description of the wearable system.

The most obvious feature of wearable applications is the communication with external devices through the IO interface. Each application starts with sending or receiving data and ends with providing feedback to users after data processing. Therefore, wearable applications can be divided into the following three stages: data collection, filtering and storage, and computing and processing.

The tasks in the data collecting stage are generally periodic, run in background, and only require fewer CPU resources. At the same time, these tasks will invoke external devices to send or receive data and hang themselves until the completion of transmission. The requirement for CPU resources of the tasks in the filtering and storage stage is related to the content of the collected data. For the small data collected by sensors, only a small amount of CPU resources are needed to complete the filtering and storage. For the large data collected by camera or microphone, more CPU resources are needed to complete data filtering as soon as possible. The tasks in the computing and processing stage mainly include picture/video processing, data encryption, and data fusion. Such tasks require a large number of CPU resources. At the same time, in order to respond to users as soon as possible, it is necessary to assign high priority to the task and improve the response speed of the system. Therefore, in the research of low-power technology for wearable applications, the characteristics of application priority, latency sensitivity, periodicity, workload, duration, and external devices invocation should be fully considered.

3.1 EAS scheduler

In order to be more suitable to mobile devices, ARM and Linaro jointly developed a new scheduler: EAS, which was originally designed to reduce the power consumption of the overall system as much as possible without affecting performance.

EAS aims to make power performance management and is applicable to the current popular multicore heterogeneous processors (such as ARM big.LITTLE processor). It combines scheduler, CPUIdle, and CPUFreq modules as shown in Fig. 1. The EAS scheduler instructs the CPUFreq module to scale the frequency of CPU cores. It also instructs the CPUIdle module to set CPU core into the idle state, which aims to make power performance control more centralized

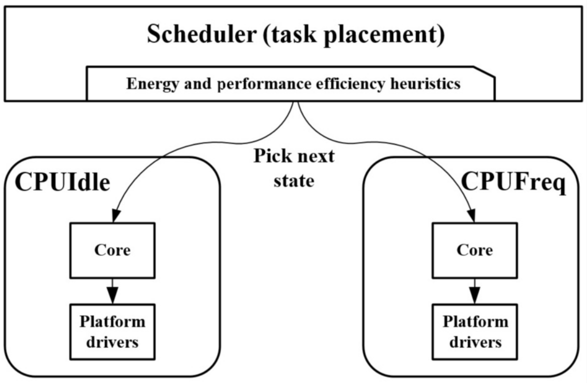
Scheduler (task placement)
Energy and performance efficiency heuristics
Pick next state
CPUIdle
Core
Platform drivers
CPUFreq
Core
Platform drivers

Fig. 1 Architecture of scheduler, CPUIdle, and CPUFreq in EAS.

with the EAS scheduler as the primary driver for power performance decisions.

The architecture of the EAS scheduler is shown in Fig. 2. Since the EAS scheduler is mainly used in the Android systems, on the basis of the traditional Completely Fair Scheduler (CFS), the EAS scheduler introduces some new features for mobile platforms, which are labeled in Fig. 2, including the CPU energy model, Window-Assisted Load Tracking (WALT) algorithm, SchedTune, wakeup path, and Schedutil. Energy model is defined as a set of frequency-power or capacity-power pairs and is an accurate baseline model of the dynamic and static powers used by the CPUs in the system. WALT is a per-task load tracking mechanism. When the system is not overutilized, all tasks will occasionally wake up. Hence, the task wakeup path module selects the CPU core where the task should be executed. SchedTune implements a single (and simple) central tunable controller to balance the energy efficiency and performance boosting. Schedutil is a CPUFreq governor that makes it possible to control CPU clock frequency selection directly from the scheduler. Schedutil works as a shim layer between the scheduler and the CPUFreq framework[16].

In the following, we will optimize the EAS scheduler combined with these new features from four aspects, including workload prediction, CPU selection, batch processing for the associated tasks, and load balancing.

3.2 Basic exponential smoothing-based WALT algorithm

In the original WALT algorithm, there are five 20 ms sliding time windows to count the historical workloads of tasks. The workload is called WALT time and is calculated as following,

$$L = \text{time}_{\text{exec}} \times \frac{\text{freq}_{\text{curr}}}{\text{freq}_{\text{max}}} \times \frac{\text{IPC}_{\text{curr}}}{\text{IPC}_{\text{max}}} \tag{1}$$

where $\text{time}_{\text{exec}}$ is the actual task execution time, $\text{freq}_{\text{curr}}$ is the current CPU frequency, freq_{max} is the max CPU frequency, IPC_{curr} is the current computational efficiency of CPU, and IPC_{max} is the max computational efficiency of CPU.

After the counting of five historical workloads, the WALT algorithm selects $\max\{L_{\text{recent}}, L_{\text{avg}}\}$ (the maximum between the most recent value L_{rescent} and the average value L_{avg}) as the predicted task workload in the next sliding window. As shown in Fig. 3, the predicted value L_{next} is 4 ms calculated with the average values; however, in the practical applicable scenario, 1 ms is a more reasonable predicted task demand.

We propose Basic Exponential Smoothing WALT (BES-WALT) algorithm to resolve the above problem. The purpose of adopting basic exponential smoothing[17] is to increase the weight of short-term loads on the predicted demand. The load calculation formula is shown as following,

$$L_{t+1} = \alpha \times x_t + (1 - \alpha) \times L_t \tag{2}$$

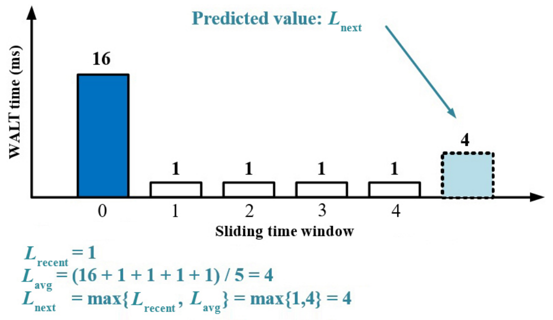

Fig. 3 Example of WALT algorithm.

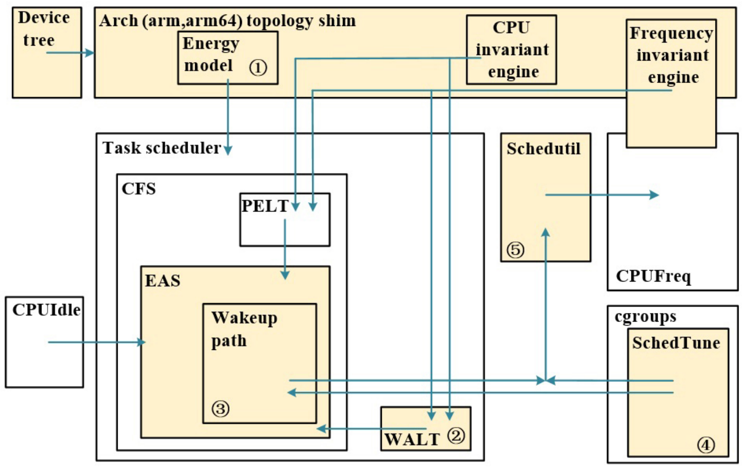

Fig. 2 EAS scheduler architecture diagram.

where L_t represents the predicted demand in the t-th sliding window, x_t represents the real demand in the t-th sliding window, α represents the smoothing factor, where $\alpha \in (0, 1)$. After substituting $L_t, L_{t-1}, \ldots, L_1$ into L_{t+1}, the following formula can be obtained after expanding and sorting,

$$L_{t+1} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + \cdots + \alpha(1-\alpha)^{t-1}x_1 + (1-\alpha)^t L_1 \qquad (3)$$

It can be seen from the above formula that basic exponential smoothing is essentially a moving average with $\alpha(1-\alpha)^k$ as the weight factor. Since the value of α is less than 1, the larger k is, the smaller $\alpha(1-\alpha)^k$ will be, and therefore, the further it is away from the current time, the smaller the workload weight will be. According to the calculation characteristics of geometric series in the PELT[18] algorithm, $\alpha = 0.5$ is selected, and in order to minimize the influence of the initial value L_1 on the predicted demand, L_1 is taken as the average value of the two oldest historical workloads of the task,

$$L_1 = \frac{x_1 + x_2}{2} \qquad (4)$$

After the predicted value L_{t+1} is obtained by basic exponential smoothing, the previous practice is continued, the predicted demand L_{next} in the next sliding window is taken as the maximum of the predicted value and the recent value,

$$L_{\text{next}} = \max\{L_{\text{recent}}, L_{t+1}\} \qquad (5)$$

Figure 4 illustrates the examples of different historical workloads and the prediction results of BES-WALT algorithm. As shown in Fig. 4, when the workload changes are relatively smooth and regular, the prediction results of basic exponential smoothing method will not distinguish from the predicted value of mean value method. When there is a sudden increase in task workloads, BES-WALT selects L_{recent} as the predicted value so that workload prediction can ensure a timely response to a sudden increase in workloads and improves the response speed of system; when task workloads suddenly decrease, BES-WALT ensures that the further it is away from the current time, the smaller the effect of historical task workloads on the prediction value will be, and avoids the effects of preemption and other abnormal operations on the task workloads in the Linux kernel scheduling process; when the task workloads are indeed on a downward trend, compared to mean value method, BES-WALT can more accurately reflect the changes of task workloads.

From above, BES-WALT algorithm can improve the accuracy of workload prediction through increasing the weight of short-term loads on the predicted demand, avoid unnecessary CPU frequency switching, and ensure that the CPU core selection is more reasonable, and the efficiency of CPU utilizing is better than WALT algorithm.

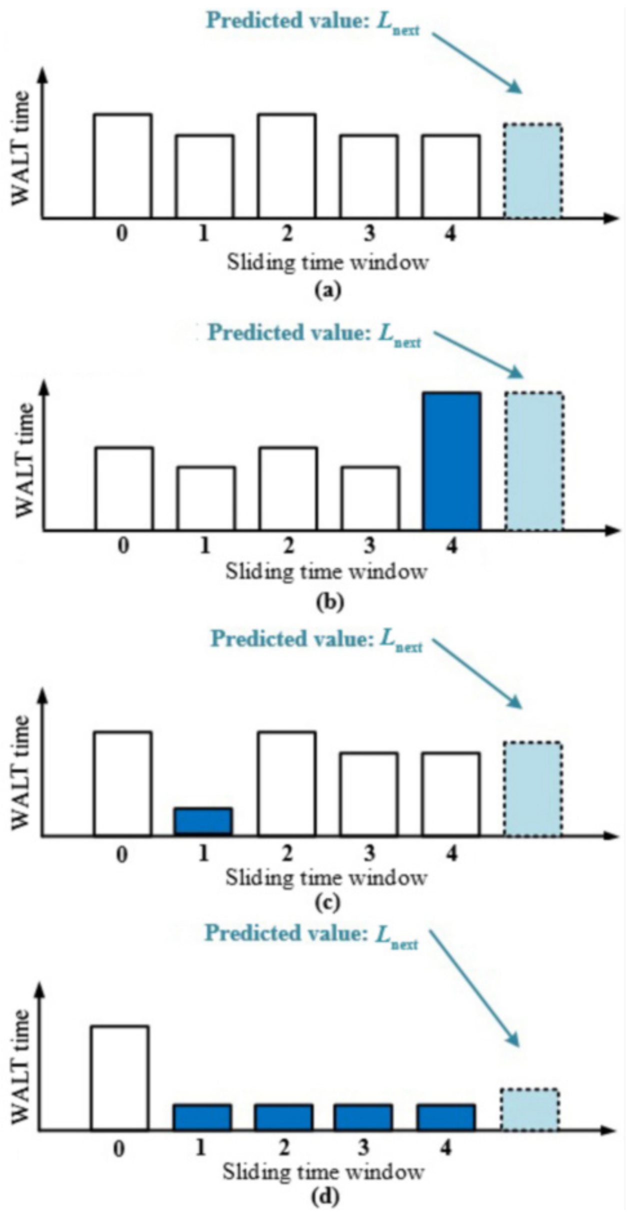

Fig. 4 Examples of using BES-WALT algorithm to predict task workload.

3.3 Energy-aware CPU selection algorithm

When the EAS scheduler performs CPU core selection with the SchedTune mechanism, the latency-sensitive tasks and the non-latency-sensitive tasks will be distinguished from each other. The CPU selection algorithm will be optimized from the perspective of latency-sensitive tasks and the non-latency-sensitive tasks in combination with the actual use scenario.

(1) Latency-Sensitive Task: When the current task is latency-sensitive, in order to complete the task as soon as possible, the original CPU selection algorithm directly traverses all the CPU cores in the system and returns the first retrieved core in the idle state. As shown in Fig. 5, Cores 0, 3, and 4 are in the idle state, Cores 0 and 4 are in the Deep idle state, Core 3 is in the Wait-For-Interrupt (WFI) idle state, the idle state level of Cores 0 and 4 is higher than Core 3. Core 0 is the first retrieved core in the idle state, so that Core 0 is returned by the original CPU selection algorithm as the target core.

Although this approach can vastly improve the searching speed for the idle core, but in big.LITTLE architecture, the power consumption of the big core is more than the little core when it is providing the same computing power. At the same time, because the DVFS Operating Performance Point (OPP) is discrete, the frequency interval of big core is larger than the little core, and the coarse granularity is more likely to cause the waste of computing power when the big core is selected. Embedded multicore processor only consists of a small number of cores and since simplifying the search process brings out improvement in the power consumption and performance, the selection of a suitable CPU core becomes essential.

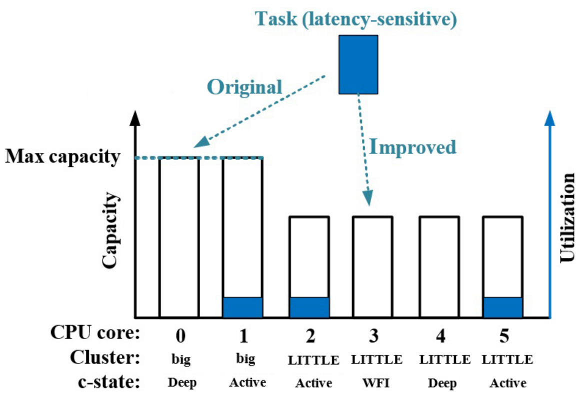

Fig. 5 Example of CPU selection for latency-sensitive tasks.

From above observations, the scheduler does not return the first idle CPU core for latency-sensitive tasks, but it seeks a core with the minimum performance capacity and the lowest idle state along with meeting the performance demand of the task. The specific algorithm is presented in Algorithm 1. In Fig. 5, Cores 0, 3, and 4 are in the idle state, Core 3 is in the lowest idle state level and the performance capacity of Core 3 meets the task demand, so that the optimized CPU selection algorithm selects Core 3 as the target core for the task.

(2) Non-Latency-Sensitive Task: As shown in Fig. 6, the EAS scheduler always assigns the non-latency-sensitive tasks to the active cores in the little-core scheduling subdomain as far as possible. It also assigns the task to Cores 2 and 4 in the little core scheduling subdomain. Due to its relatively lower utilization, Core 4 is selected as the target core.

However, this CPU selection strategy will result in the tasks being distributed relatively evenly in the little-core scheduling subdomain, and the little cores will always be active and cannot switch to the idle state. At the same time, due to the low utilization of each core, CPU cores will run in low frequency and provide low performance capacity that will affect the throughput of overall system.

In military, emergency rescue, disaster relief, and other application scenarios, the wearable platform is equipped with many sensors, such as barometer, thermo-hygrograph, cardio-tachometer, etc.

Algorithm 1 Select the suitable core for the latency-sensitive task

Input: latency-sensitive task
Output: target_core

```
1:  temp_capacity ← null //initialize temp_capacity which represents the performance capacity of target_core
2:  temp_cstate ← null //initialize temp_cstate which represents the idle-state level of target_core
3:  target_core ← null //initialize target_core which is the current optimal CPU core for the latency-sensitive task
4:  for each core in CPU do
5:      if core_capacity $\geqslant$ task_performance_demand $\wedge$ core == Idle then
6:          if core_capacity $\leqslant$ temp_capacity $\wedge$ core_cstate $<$ temp_cstate then
7:              temp_capacity ← core_capacity
8:              temp_cstate ← core_cstate
9:              target_core ← core
10:         end if
11:     end if
12: end for
13: return target_core
```

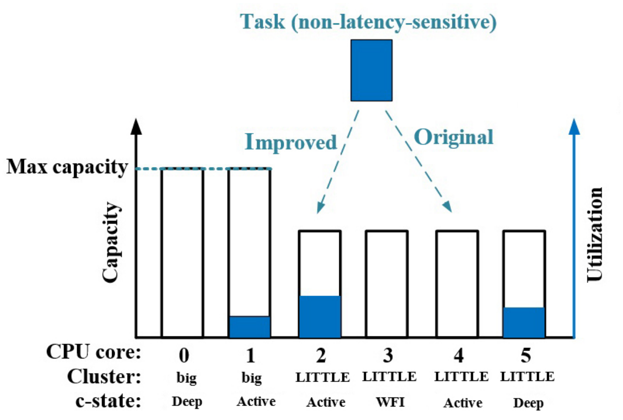
Task (non-latency-sensitive)
Improved
Original
Max capacity
Capacity
Utilization
CPU core: 0 1 2 3 4 5
Cluster: big big LITTLE LITTLE LITTLE LITTLE
c-state: Deep Active Active WFI Active Deep

Fig. 6 Example of CPU selection strategy for non-latency-sensitive task.

The tasks of collecting and processing the data from the above external devices are periodic and of low workload. But they need to be processed continuously, so that the original strategy will avoid the energy waste and longer response latency.

From above observations, we proposed "task packing strategy" for non-latency-sensitive tasks with low workload which are packaged into a process group and placed on a small number of processors. By scaling the frequency, these processors can run at a frequency closer to their computing capacity, while other processors can enter the idle state to save power. Since the system will force global load balancing when the utilization rate of any CPU core exceeds 80%, the packing threshold is set to 80% of the computing power. Meanwhile, since the power consumption of big core is much higher than the little core, the "task packing strategy" is only performed on the little cores. The implementation is presented in Algorithm 2.

After adopting the task packing strategy, the CPU selection for non-latency-sensitive tasks will change, as shown in Fig. 6. In Fig. 6, the EAS scheduler will select CPU Core 4 as the target CPU, because of the least utilization of Core 4 among the active cores in the little core scheduling domain so as to distribute the tasks. However, WAEAS will select Core 2 as the target CPU when the utilization of CPU core satisfies the packing threshold. Core 4 will enter the idle state after the completion of all the tasks, so as to achieve the purpose of putting more cores into the idle state when the system load is low and reduce the power consumption of system. Meanwhile, as the demand for computing power on Core 2 increases, CPUFreq will raise the computing power of Core 2, that will speed up the completion of tasks and reduce the response time of the system.

3.4 Batch processing strategy for the associated tasks

Complementing the above CPU selection algorithm, when there are some cores in the overutilized state and some cores are idle in the system, in order to keep load balancing in the system, the function select_idle_sibling() is called to select the suitable idle CPU core to achieve the goal of assigning tasks. For example, there are Tasks *A* and *B*, where Task *A* is an application that periodically needs to call the I/O device. When Task *A* calls the I/O device, Task *A* wakes up Task *B* and suspends itself; when Task *B* is completed, Task *B* wakes up Task *A*, and so on. Since the original function select_idle_sibling() randomly select a CPU core, Tasks *A* and *B* may migrate randomly onto different CPU cores, resulting in energy waste as shown in Fig. 7.

In wearable systems, many tasks are periodic, and they wake up each other. "Batch processing strategy for the associated tasks" is proposed to carry out such tasks. If the core is idle on which the task was run the

Algorithm 2 Select the suitable CPU core for non-latency-sensitive task

Input: non-latency-sensitive task
Output: target_core

```
1: target_core $\leftarrow$ null //target_core records the current optimal core for the object task
2: temp_utilization $\leftarrow$ 0 //temp_utilization records the new utilization of the current optimal core
3: for each core in CPU do
4:     if core == active $\wedge$ core_available_capacity $\geqslant$ task_performance_demand then
5:         new_utilization $\leftarrow$ non_latency_sensitive_task + core_utilization
6:         if core == little $\wedge$ new_utilization $<$ 80% core_capacity $\wedge$ new_utilization $>$ temp_utilization then
7:             temp_maximum_utilization $\leftarrow$ new_utilization
8:             target_core $\leftarrow$ core
9:         end if
10:    end if
11: end for
12: return target_core
```

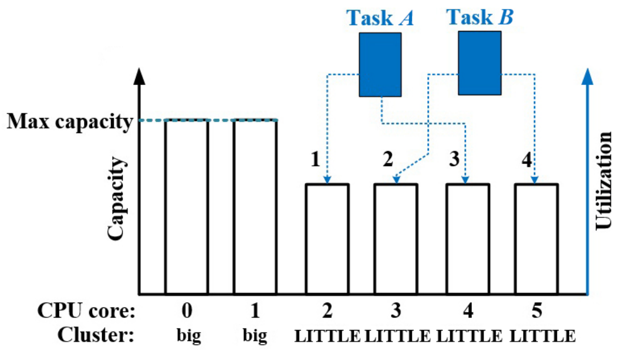

Fig. 7 Example of Tasks *A* and *B* awaken each other in the original algorithm: Task *A* (Core 2) wake up Task *B* (Core 3); Task *B* (Core 3) wake up Task *A* (Core 4); Task *A* (Core 4) wake up Task *B* (Core 5).

last time as the awakener, this task will be placed on the CPU core. In the above example, the process that Tasks *A* and *B* wake up each other is shown in Fig. 8. The proposed strategy records which core the task was running on as the awakener at the last time, and runs the tasks in the system centralized on fewer cores, which not only reduces the migration of tasks in the system, but also makes more cores in the system be in the idle state.

When some cores are in the overutilized state, the other tasks on such cores that do not belong to the category of the associated tasks and will be migrated to another idle core to achieve the goal of load balancing.

3.5 Cluster-based load balancing strategy

In the current load balancing implementation, there is a variable which indicates that the system is overutilized in the root scheduling domain. When any CPU core is overutilized, this variable will be set to true, and the whole system will be considered overutilized.

Then, the global Symmetric Multi-Processor (SMP) load balancing is performed, in which all workloads are shared among all the cores in the system, as shown in Fig. 9, and all the tasks are shared among all the cores.

SMP load balancing is mainly designed for homogeneous multicore processor; however, the embedded heterogeneous multicore processor is divided into the cluster of big cores and the cluster of little cores, which computing power is different from the other. The fairness of task scheduling will not be guaranteed in the equal assigning of tasks to the cores due to the gap in computing capacity; meanwhile, because as long as any CPU core is in the overutilized state, the load balancing will be performed and the frequent load balancing will cause a waste of system performance and energy.

According to the above observations, we proposed "cluster-based load balancing strategy" that sets an overutilized indicator in the big core scheduling subdomain and the little core scheduling subdomain, respectively, thereby indicating that whether the domain is overutilized or not. When a scheduling domain is overutilized, the current domain state is set to be overutilized first, and the load balancing in the current domain is given a priority. This is shown in Fig. 10.

Meanwhile, in order to ensure that the local load balancing will not destroy the balance of the whole system, the root scheduling domain should be set to overutilized, and load balancing over all the scheduling domains of the system should be performed if the following conditions arise.

(1) When a task running in a scheduling subdomain is not suitable to continue running in the current scheduling subdomain as time goes by.

(2) When the demand for computing power of all the tasks in some scheduling subdomain is greater than the computing power that the current subdomain can

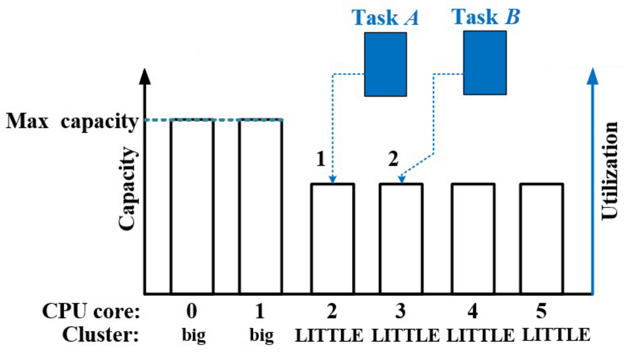

Fig. 8 Example of Tasks *A* and *B* waking up each other in the optimized algorithm: Task *A* (Core 2) wake up Task *B* (Core 3); Task *B* (Core 3) wake up Task *A* (Core 2); Task *A* (Core 2) wake up Task *B* (Core 3).

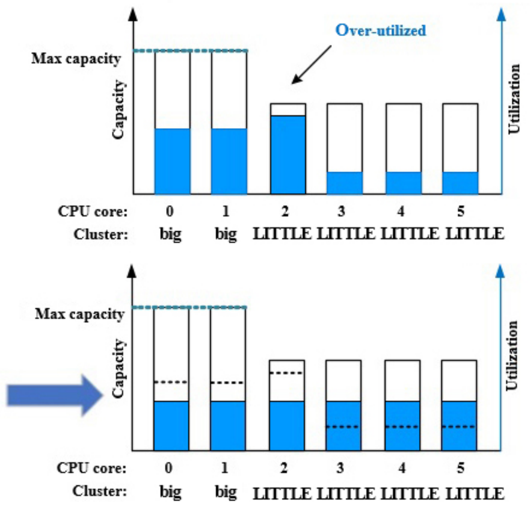

Fig. 9 Example of the original load balancing in EAS scheduler.

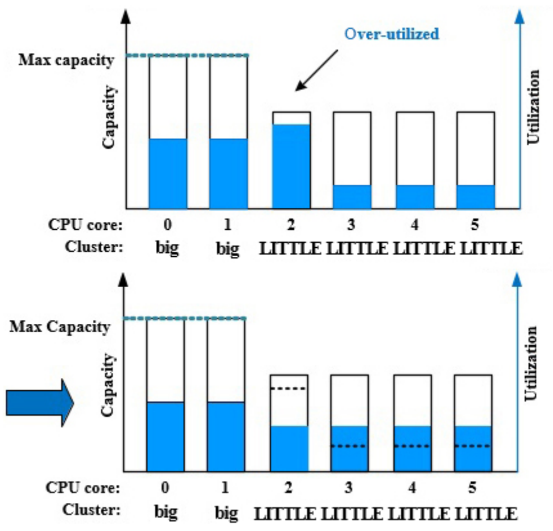

Fig. 10 Example of the cluster-based load balancing strategy.

provide.

From the above observations, there are a lot of aperiodic high-load applications among the wearable applications. When such applications appear, it will lead to unbalanced load in the little core or big core scheduling subdomain. Simply performing global load balancing will only result in energy waste and performance degradation. Cluster-based load balancing strategy reduces the meaningless task migrations in the big core and the little core scheduling subdomains, thereby reducing the system power consumption on the basis of guaranteeing the system performance.

4 Experimental Analysis

4.1 Experimental environment of software and hardware

The experiment is based on the Rockchip RK3399 processor, which employs ARM big.LITTLE architecture with dual Cortex-A72 big core (main frequency up to 2.0 GHz), quad Cortex-A53 little core (up to 1.5 GHz), 2 G LPDDR3 RAM, and 8 G EMMC ROM. The information of software platform is shown in Table 1.

4.2 Basic performance and fairness analysis of the WAEAS scheduler

In order to better assess the basic performance of the WAEAS scheduler, we use Perf[19] to perform a system performance test. Perf is one of the most common tools used for performance counter profiling on Linux, and it consists of several performance testing tools, including Sched message and Sched pipe, which are involved in basic performance testing and fairness testing of the scheduler.

(1) Basic Performance Testing: Sched message is a migration from the classic benchmark Hackbench[20], and it is primarily used to measure the performance of the scheduler. By starting N reading tasks and writing tasks, Inter-Process Communication (IPC) performs parallel reading/writing operations to measure the performance of the scheduler. Comparing before and after of the performance improvement operations, the experimental results are shown in Fig. 11.

The analysis of experimental results: when the parameter N is small (less than 60), the system workload is low. Due to the task packing strategy adopted in the CPU selection algorithm, some tasks are assigned to the same CPU core, and the performance is slightly reduced. When parameter N is greater than or equal to 60, there is a gradual increase in the system workload, the WAEAS scheduler achieves reasonable CPU selection

Table 1 Information of software platform.

Software platform	Version
Linux kernel	Linux Kernel 4.4.83
OS	Android 7.1.2
Application compiler	aarch64-linux-gnu-gcc 5.4.0

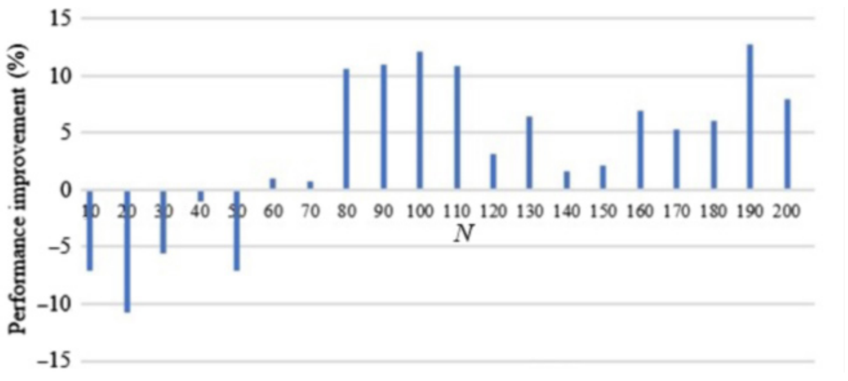

Fig. 11 Basic performance improvement in percentage after improvement.

and load balancing, and the system performance has been improved to a certain extent. As the influence of workload and the OPP of CPU cores, the additional amount of performance improvement fluctuates within the range of N from 120 to 180.

(2) Scheduling Fairness Testing: Sched pipe is mainly used to evaluate the fairness of the scheduler. Its basic principle is to create two tasks: Tasks A and B, the two tasks send N integers to each other through pipe, both are mutually dependent. If the scheduling policy is unfair by showing a preference for a particular task, the whole running time will be longer. The experimental results are shown in Fig. 12.

Experimental results analysis: when parameter N is equal to 100 000, there is a slight decrease in the fairness of WAEAS scheduler. When parameter N is greater or equal to 200 000, the overall fairness of WAEAS scheduler is slightly better than that of EAS scheduler by up to 2%.

4.3 Performance and power consumption analysis of scheduler in multithreaded applications

ParMiBench[21] is an open source benchmark based on MiBench[22] and targeted for multiprocessor-based embedded systems. ParMiBench consists of parallel implementations of seven compute-intensive algorithms in four categories: automation and industry control, network, office, and security. Experimental results are shown in Table 2. The performance and power consumption improvement is calculated as following,

$$\text{Perf}_{\text{improv}} = \frac{\text{Perf}_{\text{WAEAS}}}{\text{Perf}_{\text{EAS}}} - 100\% \qquad (6)$$

$$\text{Power}_{\text{improv}} = 100\% - \frac{\text{Power}_{\text{WAEAS}}}{\text{Power}_{\text{EAS}}} \qquad (7)$$

Table 2 Performance and power consumption improvement in percentage of each program in ParMiBench.

(%)

APP	Performance	Power consumption
Bitcount	2.44	4.04
Susan	0.01	4.68
Basicmath	0.81	6.33
Patricia	46.20	48.51
Dijkstra	1.92	8.09
Stringsearch	1.23	4.38
SHA-1	–13.87	–10.16

Experimental results analysis: among the seven applications, Patricia shows the most obvious improvements in performance by 46% and a decrease in power consumption by 48%. When Patricia is running, the overall utilization of CPU is low, indicating that when the system workload is low, the WAEAS scheduler has advantages for CPU selection and load balancing. For Bitcount, Susan, Basicmath, Dijkstra, and Stringsearch, as compared with EAS scheduler, the performance of WAEAS scheduler is slightly improved, and the power consumption of these 5 applications is improved by an average of 5%, which verifies that the WAEAS scheduler has more advantage in power consumption. However, the results of SHA-1, which is a low-load application, are not satisfactory. SHA-1 is repeated 100 times in a single experiment to reduce errors that result in SHA-1 running at a high density. This is done because WAEAS will give low-load tasks priority to choose a little core for task packing, and it does not give full play to the performance advantages of big cores, resulting in the performance degradation of SHA-1. At the same time, because SHA-1 has been running on little cores, due to insufficient computing

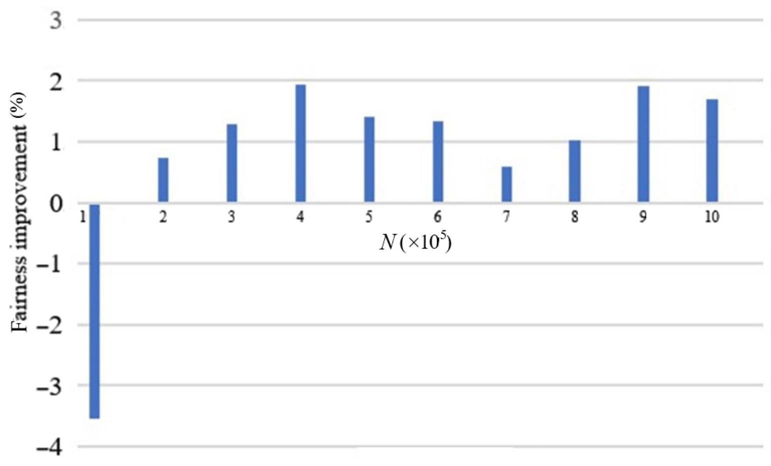

Fig. 12 Scheduling fairness improvement in percentage.

power of little cores, the running time is too long, and the power consumption increases with the running time too.

4.4 Experimental design of practical wearable application scenarios

In order to verify the efficiency of the optimized scheduling strategies in a real scenario, we designed three practical wearable application scenarios and five wearable applications on the experimental platform, as presented in Table 3.

The Rockchip RK3399 development board serves as the kernel module that connects thermo-hygrograph, barometer, GPS, and cardio-tachometer via bluetooth, respectively. The APP installed on the development board controls the start and end of testing. The testing scenarios are designed as following.

(1) A low system workload is simulated by separately starting each external device, and the power consumption of the development board is recorded within 1 minute after the data collection process is stable.

(2) A heavy system workload is simulated by simultaneously starting all external devices and the power consumption of the development board is recorded within 1 minute after the data collection process is stable.

(3) The transmission of large data files is simulated by transmitting a 4 GB file from Seafile client to Seafile server via WI-FI and the power consumption of the development board is recorded within 1 minute after the file transmission is stable.

4.5 Experimental results and analysis in practical wearable application scenarios

We tested the power consumption of Completely Fair Scheduler (CFS) scheduler, EAS scheduler, and WAEAS scheduler in the practical wearable application scenarios, the experimental results of which are shown in Fig. 13. The power consumption of CFS scheduler is obviously higher than that of the EAS scheduler, and the WAEAS scheduler achieves the lowest power consumption. The power consumption improvement in the percentage of EAS scheduler and WAEAS scheduler is shown in Table 4.

Analysis of Experimental Results: there is an improvement of about 10% in the power consumption of the development board when it is connected with a single external device in low sampling frequency (GPS, barometer, and thermo-hygrograph). Similarly, the power consumption of the development board improves by about 5% when it is connected with a single external device in the high sampling frequency (cardio-tachometer); and its power consumption is reduced by 8% when it is connected with all the external

Table 4 Power consumption improvement in percentage of EAS and WAEAS. (%)

Workload	Power consumption
GPS module	9.90
Barometer	10.97
Thermo-hygrograph	15.40
Cardio-tachometer	4.93
All devices	8.05
Large file transmission	4.52

Table 3 Design of the practical wearable application scenario.

External device	Sampling frequency (Hz)	Data type	Computing process
GPS	1	Latitude and longitude	Display real-time location information
Barometer	5	Floating number	Display real-time altitude information translated from air pressure
Thermo-hygrograph	20	Floating number	Display real-time temperature and humidity information
Cardio-tachometer	40	Integer	Display real-time heart rate information and threshold warning
File server	-	4 GB binary file	Simulate the transmission of large files

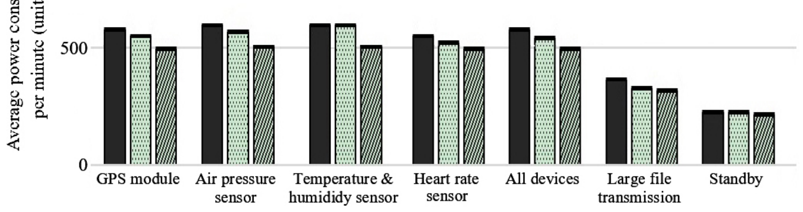

Fig. 13 Power consumption of CFS, EAS, and WAEAS.

devices. Hence, the power consumption of a large file transmission is improved by 4.5%. These experiments verify the efficiency of the WAEAS scheduler in the practical wearable application scenarios.

5 Conclusion

This paper analyzed performance energy efficiency in the dynamic scheduling of latency-sensitive and non-latency-sensitive tasks for heterogeneous multicore wearable systems. By aiming to improve the energy efficiency of workload prediction, CPU core selection, and load balancing without affecting performance, the authors further optimized the EAS scheduler and proposed WAEAS, which is an optimization of the EAS scheduler for wearable applications and includes several optimized algorithms and strategies, such as basic exponential smoothing-based WALT algorithm, energy-aware CPU selection algorithm, batch processing strategy, and cluster-based load balancing. The experimental results have verified the effectiveness of WAEAS scheduler. First, the performance and fairness of EAS scheduler and WAEAS scheduler were tested by using the Perf tool in the Linux kernel, and the optimized EAS scheduler showed an improvement of 5% when the workload was high. Secondly, the performance and energy efficiency of EAS scheduler before and after the improvement conditions were tested with ParMiBench, and the experimental results showed an improvement in the performance and power consumption tested by the other six benchmark programs except SHA-1. Finally, the authors tested the power consumption in practical application scenarios, and the experimental results showed that the power consumption was improved by about 8%. The next step will focus on testing the WAEAS scheduler on more complex platforms with more practical wearable applications and making further improvement of performance and energy efficiency.

References

[1] Y. Liu, D. Liu, and G. Yue, BGMM: A body Gauss-Marko-based mobility model for body area networks, *Tsinghua Science And Technology*, vol. 23, no. 3, pp. 277–287, 2018.

[2] G. Li, S. Peng, C. Wang, J. Niu, and Y. Yuan, An energy-efficient data collection scheme using denoising autoencoder in wireless sensor networks, *Tsinghua Science And Technology*, vol. 24, no. 1, pp. 86–96, 2019.

[3] N. K. Govindaraju, J. Gray, R. Kumar, and D. Manocha, GPUTeraSort: High performance graphics co-processor sorting for large database management, in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2006, pp. 325–336.

[4] B. Priyantha, D. Lymberopoulos, and J. Liu, Littlerock: Enabling energy-efficient continuous sensing on mobile phones, *IEEE Pervasive Computing*, vol. 10, no. 2, pp. 12–15, 2011.

[5] N. D. Lane, P. Georgiev, C. Mascolo, and Y. Gao, Zoe: A cloud-less dialog-enabled continuous sensing wearable exploiting heterogeneous computation, in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, New York, NY, USA, 2015, pp. 273–286.

[6] S. Bhattacharya and N. D. Lane, From smart to deep: Robust activity recognition on smart watches using deep learning, presented at 2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops), Sydney, Australia, 2016.

[7] R. Buchty, V. Heuveline, W. Karl, and J. Weiss, A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators, *Concurrency and Computation: Practice & Experience*, vol. 24, no. 7, pp. 663–675, 2012.

[8] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli, Scheduling for embedded real-time systems, *IEEE Design & Test*, vol. 15, no. 1, pp. 71–82, 1998.

[9] P. Wu and M. Ryu, Best speed fit EDF scheduling for performance asymmetric multiprocessors, *Mathematical Problems in Engineering*, vol. 2017, pp. 1–7, 2017.

[10] G. Lipari, Earliest deadline first, http://retis.sssup.it/∼lipari/courses/rtos/lucidi/edf.pdf, 2005.

[11] H. Khan, Q. Bashir, and M. U. Hashmi, Scheduling-based energy optimization technique in multiprocessor embedded systems, presented at 2018 International Conference on Engineering and Emerging Technologies (ICEET), Lahore, Pakistan, 2018.

[12] J. Cheng, Research of energy efficient strategy on big.LITTLE architecture for wearable devices, master degree dissertation, Harbin Institute of Technology, Harbin, China, 2017.

[13] Z. Huang, Cooperative scheduling method with hardware and software for heterogeneous multi-core in embedded system, master degree dissertation, Zhejiang University, Hangzhou, China, 2007.

[14] X. Gao, Fair scheduling on dynamic heterogeneous chip multiprocessors, master degree dissertation, University of Science and Technology of China, Hefei, China, 2015.

[15] X. Cong, Research on the embedded low power consumption technology customized for wearable applications, master degree dissertation, Harbin Institute of Technology, Harbin, China, 2018.

[16] EAS overview and integration guide (r1p6), https://developer.arm.com/tools-and-software/open-source-software/linux-kernel/energy-aware-scheduling, 2018.

[17] E. S. Gardner, Exponential smoothing: The state of the art, *Journal of Forecasting*, vol. 4, no. 1, pp. 1–28, 1985.

[18] P. Turner, CFS per-entity load tracking, https://lwn.net/Articles/504013/, 2012.

[19] A. Melo, The new Linux 'perf' tools, http://www.linux-kongress.org/2010/slides/lk2010-perf-acme.pdf, 2010.

[20] Y. Zhang, Hackbench, http://people. redhat.com/mingo/cfs-scheduler/tools/hackbench.c, 2008.

[21] S. Iqbal, Y. Liang, and H. Grahn, Parmibench, an open-source benchmark for embedded multiprocessor systems, *IEEE Computer Architecture Letters*, vol. 9, no. 2, pp. 45–48, 2010.

[22] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, MiBench: A free, commercially representative embedded benchmark suite, in *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization*, Austin, TX, USA, 2001, pp. 3–14.

Zhan Zhang received the PhD degree from Harbin Institute of Technology (HIT), in 2008. Since 2014, he has been an associate professor with the School of Computer Science and Technology, HIT. His research interests include fault tolerant computer, wearable computer, and system evaluation theory and technology.

Xiang Cong is a master student at the School of Computer Sceince and Technology, HIT. His research interests include wearable computing, cloud computing, big data, and fault tolerant computing technology.

Wei Feng received the PhD degree from Tsukuba University, Japan, in 2011. Currently, he is an engineer at the School of Computer Science and Technology, HIT. His main research interest is wearable and mobile computing technology.

Haipeng Zhang is a master student at the School of Computer Science and Technology, HIT. His research interests include wearable computing, fault tolerant computing, and low-power technology.

Guodong Fu is a master student at the School of Computer Science and Technology, HIT. His research interests include wearable computing, fault tolerant computing, and testing technology.

Jianyun Chen is with Beijing Information Technology Institute. Her research interests include wearable computing, pattern recognition, and systems engineering.