Improved dynamic frequency-scaling approach for energy-saving-based radial basis function neural network

Deguang Li1, Ruling Zhang1, Shijie Jia1, Dong Liu1, Yanling Jin1, Junke Li2
1School of Information Technology, Luoyang Normal University, Luoyang 471934, People’s Republic of China
2College of Computer and Information, Qiannan Normal University for Nationalities, Qiannan 558000, People’s Republic of China
E-mail: lideguang.00@163.com

Abstract: As dynamic voltage and frequency scaling (DVFS) does not consider predicting system behaviour in the future stage, to improve efficiency of DVFS in fine-grained, the authors propose a central processing unit (CPU) utilisation prediction model based on radial basis function neural network. Their model first collects five typical system characteristics related to CPU utilisation during system running, then they use radial basis neural network to fit the non-linear relationship between these system characteristics and CPU utilisation in the next period. According to the predicted CPU utilisation, specific frequency scaling is performed to change frequency in real time. Finally, they evaluate their model with classical DVFS by means of different task sets. Experimental results show that their model could predict CPU utilisation in more fine-grained compared with other models, and changes frequency-scaling effect of traditional DVFS.

1 Introduction

Information and communication technology (ICT) industry as one of the fastest growing industries in the world has accounted for 10% of the total global electricity consumption in the world [1], and its carbon emissions have reached 700 million tons/year, increasing at an annual rate of about 4% [2]. To promote the green, low-carbon and sustainable development of ICT industry, green computing [1–6] has become the consensus of many domestic and foreign researchers. At present, the widely used software energy-saving technologies mainly include dynamic voltage and frequency scaling (DVFS) [7], dynamic power management [8], temperature management and task scheduling under heterogeneous multi-core system [9] etc.

Research on DVFS of energy saving has been carried out for many years. Weiss et al. [10] first put forward the idea of operating system (OS)-level power management and implemented unbounded-delay perfect-future (OPT), bounded-delay limited-future (FUTUR) and bounded-delay limited-past (PAST) algorithms. Govil et al. [11] had also made a great contribution to it by improving the algorithm proposed by Mark. DVFS dynamically adjusts operating voltage and frequency of the chip according to different computing requirements of the running application. Generally, if the current running task needs to perform a large number of calculations, the processor frequency and voltage are increased to provide higher computing power; thus, the running time of the software is shortened and the energy consumption of the task is reduced; if the current running task needs less computing, the frequency and voltage of the processor should be appropriately reduced to reduce the power consumption of the processor. At present, DVFS has been widely used in central processing unit (CPU), light-emitting diode and other components, which can effectively reduce the system power consumption, especially for mobile and hand-held devices with limited energy; thus, CPU manufacturers all provide support for DVFS. For example, Intel's X86 CPU generally has a five-level frequency modulation mechanism, ARM's ARM series CPU generally has a three-level frequency modulation, Intel's SpeedStep and AMD's PowerNow all implement DVFS in their processor.

The key point of DVFS to adjust the frequency and voltage is how to know the CPU utilisation in the next period. Current methods are mainly divided into two categories: one is based on the historical execution path of the task; the other is based on the current or historical system information. Although the above methods have been extensively studied in different aspects of the system, there are still some shortcomings, which do not take into account of the prediction of the system behaviour in the future. In this paper, system behaviour of the future stage is predicted according to five typical system characteristics of the current running system; these characteristics represented the system load, which are number of instructions per cycle (IPC), average cache memory access time (ACMT), number of system interruptions (NSIs), number of context switching (NCS) and branch prediction errors. Then, we use radial basis function (RBF)-CPU utilisation prediction model to calculate CPU utilisation in the next period. Finally, we evaluate our method by different task sets, experimental result show that our proposed method is effective.

Our contributions in this paper are: (i) five typical system characteristics are selected to describe CPU utilisation of the system and quantitative formula for each characteristic are put forward to calculate each value. (ii) An effective frequency-scaling strategy is given to change traditional DVFS, which is simple and fast. (iii) We implement our method in Linux system and detail experiment is designed to test the method, experimental result shows that our method is more effective compared with the previous method with acceptable performance loss.

The organisation of this paper is as follows. Section 2 presents related work of DVFS, and an overview of our model is described in Section 3. In Section 4, five typical system characteristics are described, and experimental results are presented in Section 5; finally, we conclude our work in Section 6.

2 Related works

Weiser et al. [10] first put forward the idea of OS-level power management and the proposed power consumption of processors by using dynamic voltage and frequency regulation. He simulated the trajectory and slack time of task execution, and chooses a new frequency for each task when OS performs task scheduling. They designed popular algorithms of OPT, FUTUR and PAST; however, OPT and FUTUR cannot be realised due to the use of future system information, and then the algorithm PAST that can be realised is presented. The key idea of PAST setting the frequency and voltage of the next time is according to the work left and processor utilisation of the previous time. Govil et al. [11] noted that PAST algorithm is only based on the CPU utilisation information within a single time interval; thus, the algorithm
cannot change frequency speed smoothly. For this reason, he proposed a method based on multi-cycle viewing, also monitoring the cycle mode of processor utilisation and high utilisation in a short time, then the processor frequency is scaled based on this information, which is called PEAK algorithm. He also proposed long-short and other algorithms, the long-short algorithm, which focuses more on prediction, tries to find a golden value between local baseline and other baseline average over a long period of time. The proposed algorithm provides a direction for predicting future system behaviour by different methods.

The voltage scheduler proposed by Pering et al. [12] was responsible for determining the speed and voltage of the processor, when a task is added or deleted in the system, the voltage scheduler is re-evaluated and evaluated periodically during task execution. Literatures [13–15] also designed DVFS algorithm for single-core processor from the perspectives of system components, characteristics of executing tasks and load of tasks. The method proposed by Kobayashi was based on processor direction aware [16], and a time-based frequency adjustment algorithm is proposed for hand-held devices in [17], the algorithm reduces energy consumption by extending task execution time without affecting user experience, the effectiveness of this method depends on the amount of latency user that allows to increase. This algorithm may reduce the performance of the system, but it can significantly reduce energy consumption. Gruwald et al. [18] confirmed the above conclusion, and they took the weighted average of the utilisation rate of the past $N$ time periods as the predicted value of the processor utilisation rate in the next time period, and finally, found that this method saves more energy than the algorithm proposed in [17]. However, the algorithm has another disadvantage, since the utilisation of CPU in the next period needs to consider the utilisation of in the first $N$ periods; it has a delay response to the scenario, where the utilisation of CPU changes rapidly.

Dhodapkar and Smith [19] presented a method of dynamic frequency scaling based on the information of the previous task, also in [20], which uses track changes in task behaviour to monitor repeated execution paths. A DVFS decision mechanism for energy acquisition system is proposed by Liu et al. [21], which dynamically adjusts the voltage and frequency of the processor based on the residual energy of the system. Isci et al. [22] proposed a DVFS algorithm for each CPU core, which can estimate the global system power consumption under different strategies and optimise the system throughput. Khan et al. [23] proposed a DVFS recovery algorithm based on task relaxation time, and implemented the algorithm on ARM A9 and ARM11 platforms. Liu and Qiu [24] put forward a two-stage discrete DVFS algorithm based on multi-core processor, which optimises the critical path of the task model and allocates tasks to different energy gradients to optimise the system power consumption.

Liu et al. [25] built a DVFS capable CPU/graphics processing unit (GPU)/field-programmable gate array heterogeneous computing infrastructure, which can adjust its hardware characteristics dynamically in terms of the software runtime contexts, so that the applications can be executed efficiently with less time and lower-energy cost. Some other researchers put forward different approaches for energy saving for various application scenarios by improved DVFS such as power management by using DVFS in data centres [26], cyber physical–social systems [27], cloud computing [28] and radio-frequency identification systems [29]. Xie et al. [4] put forward an incrementally scalable and cost-efficient interconnection structure for data centres. Yousra and Samir [30] also pointed that DVFS was a key technique in exploiting configurable characteristics of the processor in power-aware processors and proposed a fuzzy approach to guarantee the balance between the schedule ability of real-time periodic application and its energy consumption optimisation under multi-core architecture. Ma et al. [31] proposed framework to improve soft-error reliability while satisfying a lifetime reliability constraint for soft real-time systems executing on integrated CPU and GPU platforms, also based on each core's executing frequency and utilisation, Ma et al. [32] designed a framework that performs workload migration between high-performance cores and low-power cores to reduce power consumption and improve soft-error reliability. Sorkhpour et al. [33] used scenario-based meta-scheduling for the optimisation of frequency scaling in time-triggered multi-core architectures.

We can conclude that the key point of DVFS is how to know the CPU utilisation rate in the next period based on previous works [17–26]. Many research are based on historical information of the running system and the characteristics of the current task to adjust system frequency. Some studies performed simple linear fitting of the relationship between the operating characteristics of the system and most appropriate frequency; this type of algorithm is easy to implement, but the error between the adjusted frequency and optimal frequency is large. Some research proposed discontinuous function to describe the system frequency, which also has above problems. To describe the relation between system characteristics and CPU utilisation in the next stage more accurately, we propose using RBF neural network (RBFNN) to predict CPU utilisation in the next stage based on system running characteristics.

3 Our approach

To obtain the CPU utilisation in the next period, we assume that there is a non-linear function relation between the CPU utilisation in the next period and system characteristics during system running, as linear relation is a special representation of non-linear relation, non-linear function relation is more accurate to describe the relationship between these unknown relations, the formula of our prediction model is (1). In the model, $Y$ is the CPU utilisation in the next period, $k_1, k_2, …, k_n$ are characteristics of the system during system running, $f$ is the non-linear function relationship between them. To describe the function $f$ more accurately, RBFNN is used to fit it, as it has been proved that RBFNN is a kind of non-linear functional network with fast convergence speed, simple structure and approximation ability. Now, we present detail steps for our method

\[
Y = f(k_1, k_2, k_n) = F(k_1, k_2, …, k_n)
\]  

First, the input and output layers of RBFNN are determined. In this paper, the number of IPC, ACMT, NSIs, the NCS and branch prediction errors are the input of the network, and CPU utilisation in the next period is the output. Second, the data of input and output layers of running tasks in training set are collected, and then we train the RBFNN based on the above data; finally, the trained optimal network structure and related parameters are saved. Finally, five real-time system characteristics are taken as the input of the neural network, and the output of the neural network is the CPU utilisation of the next period.

After obtaining the utilisation rate of each CPU core, it is to decide whether to change the frequency of each CPU core in the next period according to the utilisation of each CPU core. Our adjustment strategy is as follows:

(i) When the predicted utilisation is $\geq 80\%$ and the current running frequency is not the maximum, and the CPU utilisation of previous three periods is $\geq 80\%$, we change the frequency to the highest level; otherwise, we will only turn up the frequency one level.

(ii) When the predicted utilisation is $<30\%$ and the current running frequency is not the minimum, and the CPU utilisation in the first three time periods is $<30\%$, we change the frequency to the minimum level; otherwise, we will only turn down the frequency one level.

(iii) For other situations, our approach does not change system frequency.

4 Characteristics for consideration

The selected system characteristics directly affects the accuracy of our model; after a lot of literature reviews and experimental analysis, there are several system characteristics that directly indicate the utilisation of processor, which are the number of IPC, ACMT, NSIs, NCS and branch prediction errors. The number of IPC is directly reflecting CPU utilisation, which has been used by a
large number of researchers for CPU utilisation prediction, ACMT has a great influence on the performance of the CPU, which is directly affecting execution time of the program, NSIs, NCS and branch prediction errors are all important system events, which are all affecting execution time and CPU speed of the program.

### 4.1 Number of IPC

Computer completes the tasks by executing various instructions through the CPU; the more instructions executed the busier the CPU is. Generally, an instruction cycle consists of several processor cycles and different instructions require different processor cycles. For example, a complex instruction may require many processor cycles to complete. The number of IPC refers to the average number of instructions executed per processor cycle. The larger the number of IPCs, the more instructions processed in a unit clock cycle. As the number of IPCs increases, the processor utilisation will increase. Therefore, the number of IPCs can reflect the utilisation of the processor. The formula of IPC is shown in the equation below:

\[
\text{IPC} = \frac{\text{number( executed instructions )}}{\text{number( processor cycles )}}
\]

Number (executed instructions) is the number of instructions executed and number (processor cycles) is the number of processor cycles consumed by the program.

### 4.2 Average cache memory access time

Memory access of the processor mainly includes access to L1 cache, L2 cache, L3 cache, main memory and disc. The longer the memory access time, the longer the CPU waits and the idler the CPU is. Owing to the contradiction between the CPU speed and memory read/write speed, CPU usually takes a long time to wait for the data to arrive or write the data to the memory. The cache has a great influence on the performance of the CPU, and since the locality principle of the memory access. The cache hit rate is very high (most can reach about 90%); thus, the average memory access time directly affects CPU utilisation. The average cache access time depends on hit time, failure rate and failure overhead; the formula of the average memory access time is as below:

\[
\text{ACMT} = H_{L1} + M_{L2}(H_{L1} + M_{L2}(H_{L1} + M_{L3} \times P_{L3}))
\]

\[
H_{L1}, H_{L2} \text{ and } H_{L3} \text{ are the hit ratios of the L1 cache, L2 cache and L3 cache, respectively; } M_{L1}, M_{L2} \text{ and } M_{L3} \text{ are the failure rates of each memory; and the failure overhead of the third level cache is } P_{L3}.
\]

### 4.3 Number of system interruptions

Usually, the interruption processing process includes request interruption, interruption response, shut down interruption, protect break point and interruption return. These operations will increase context switching, the instruction and memory access time etc., which directly affect the utilisation of CPU. In Linux system, the proc/interrupts file can be used to monitor the number of interrupts in each CPU core, which records the number of interrupts on each CPU core from start-up to current time. If the number of interrupts of each CPU core at each sampling point is obtained by taking two sampling points, the number of interrupts of each CPU core in the time period between two sampling points can be calculated by formula (4), \(\text{Interruption}_{1} \text{ and } \text{Interruption}_{2}\) are the NSIs at two sampling points

\[
\text{NSI} = \text{Interruption}_{2} - \text{Interruption}_{1},
\]

### 4.4 Number of context switching

Context switching usually has to save the context of the previous task, excessive context switching consumes more time on the storage and recovery of data such as registers, kernel stacks and virtual memory; thereby, shortening the actual running time of the process, resulting in a significant drop in overall system performance. The task with more context switching is usually computationally intensive. Context switching includes voluntary context switches \((\text{NSI}_{\text{vswch}})\) and non-voluntary context switches \((\text{NSI}_{\text{nsvwch}})\), the calculation of NSI is shown in (5). \(\text{NSI}_{\text{vswch}}\) refers to the number of voluntary context switches/s, \(\text{NSI}_{\text{nsvwch}}\) refers to the number of involuntary context switches/s. Voluntary context switch refers to the context switch caused by the process's inability to obtain the required resources. For example, when input/output, memory and other system resources are insufficient, voluntary context switching will occur. Involuntary context switch refers to the context switch that occurs when the process is forced to schedule by the system due to the time slice. For example, involuntary context switching will occur when a large number of processes are competing for CPU

\[
\text{NSI} = \text{NSI}_{\text{vswch}} + \text{NSI}_{\text{nsvwch}}
\]

### 4.5 Number of branch prediction errors

Branch prediction is a method used by the processor to improve its execution speed, which predicts the branch flow of a program, read a branch instruction in advance and decode to reduce the time waiting for the decoder. Once a branch prediction error has occurred, the instructions and results that have been pre-loaded into the pipeline must be completely cleared before the correct instructions can be loaded for reprocessing, which will result in additional cycle consumption. Therefore, the branch prediction error will lead to the increase of CPU waiting time, which will lead to less instruction completion in the same cycle. The NBPEs of each CPU core in the time period between two sampling points can be calculated by formula (6), \(\text{NBPE}_{2}\) and \(\text{NBPE}_{1}\) are the NBPEs at two sampling points

\[
\text{NBPE} = \text{NBPE}_{2} - \text{NBPE}_{1}
\]

## 5 Non-linear fitting of RBFNN

As we use training data to train RBFNN, thus the accuracy of the prediction of CPU utilisation depends on whether the training data covers various characteristics of different running tasks of the system. Therefore, for the training set, we select the benchmark program including various typical tasks, which includes CPU-intensive tasks, input & output (IO)-intensive tasks, hybrid tasks etc. Then, we get the five system characteristics of training tasks; these eigenvalues are normalised as inputs to RBFNNs. Next, we get the appropriate frequency of each task in the training set, and the appropriate frequency of the task was normalised as the output of the radial basis neural network. Then, the structure and parameter of RBFNN are determined; important parameters include the number of hidden layer nodes, variance, RBF centre, and the weight of hidden layer to output layer. On the basis of the training set, the RBFNN is trained to obtain the network structure and the optimal parameters of the RBFNN under the predefined mean square error. Finally, five characteristics profiled in real time of the running system are used to predict CPU utilisation rate in the next period.

In this paper, we use a generalised RBFNN to get the relation between the system characteristics and CPU utilisation. Our design of RBFNN is mainly divided into structural design and parameter design. Among them, structural design refers to how many nodes are included in the input layer and output layer, as in our network, there are five input nodes and one output node, and 25 nodes in the model layer, parameter design refers to the solution of network parameters. In this paper, the RBFNN is established by using the toolbox function newrb() in MATLAB, the equation of newrb() is (7), \(P\) is the input matrix of \(R\), \(Q\) is the input matrix of \(S\)

\[
\text{where } Q = \text{number of training samples input, } R = \text{the dimension of input vector and } S = \text{the number of output nodes; goal is the specified mean square error, default value is 0; spread is the diffusion speed of RBF, default value is 1; MN is the maximum number of implicit nodes, default value is } Q; \text{ and } DF = \text{the number}
\]
of additional nodes needed between two displays, default value is 25. Among them, the number of training samples of matrix $P$ is 1560, and the dimension of input vector is 5; the set error is 0.01 and the diffusion speed spreads is 0.6, which is the optimal value obtained in many training tasks

$$\text{net} = \text{newrb}(P, T, \text{goal}, \text{spread}, \text{MN}, \text{DF})$$  \hspace{1cm} (7)

For each type of training set, we randomly selected 50 tasks, and a total of 300 tasks as our training set. For each task, we need to collect the system characteristic data at the current time of each sample data and the CPU utilisation rate at the next time. First, we propose the calculation method of CPU utilisation. As we implement our method in Linux system, CPU utilisation can be calculated by using the /proc/stat file, which contains information about all CPU activities of the system from start-up to current time. For each core, the data can be calculated by formula (8) (Fig. 1).

$T_{\text{user}}$ is the time when CPU runs in user mode, excluding processes with negative nice value. $T_{\text{nICE}}$ is the time occupied by processes with negative nice value. $T_{\text{system}}$ is the time when the CPU runs in the core state. $T_{\text{idle}}$ is the time waiting time other than IO waiting time. $T_{\text{irq}}$ is the hard interrupt time of CPU. $T_{\text{softirq}}$ is the soft interrupt time of CPU. $T_{\text{stealstolen}}$ is the time spent on other OSs while running in a virtual environment. $T_{\text{guest}}$ is the time spent on virtual CPU control when accessing guest OSs, excluding processes with negative nice value and $T_{\text{guest nice}}$ is the virtual CPU control time spent on processes with negative nice value when accessing guest OS. For each task, we first obtain CPU time data of two sampling points, and then the CPU core utilisation rates between two sampling points can be calculated by formula (9). Next, to ensure the validity of sampling, we collect ten sampling points to train the RBFNN for each task

$$\begin{align*}
T_{\text{cpu}} &= T_1 + T_2 \\
T_1 &= T_{\text{user}} + T_{\text{nICE}} + T_{\text{system}} + T_{\text{idle}} + T_{\text{iowait}} \\
T_2 &= T_{\text{irq}} + T_{\text{softirq}} + T_{\text{stealstolen}} + T_{\text{guest}} + T_{\text{guest nice}} \\
I_{\text{cpu}} &= 1 - \frac{(T_d - I_d) \cdot (T_{\text{cpu}} - T_{\text{cpu}})}{T_{\text{cpu}} - T_{\text{cpu}}}
\end{align*}$$ \hspace{1cm} (8)

After getting the utilisation of each CPU in a sample period, we need to get the five system characteristics during this period. In the Linux system, there is a system call called perf_event_open, which can be used to collect hardware event data. Owing to the high real-time requirement of dynamic frequency modulation, we need to implement our method in the kernel module, while the system calls cannot be directly used in the kernel module. After analysing the implementation process of system call, we know that each system call corresponds to a system call service routine in the kernel, but the name of the service routine is prefixed with a 'sys_', so we can use the service routine to complete system call in the kernel module. When collecting training data, it is necessary to collect each CPU frequency. It is necessary that the acquisition time of each CPU frequency is longer than that of different types of tasks in the training set. Only in this way can the data collected by each CPU frequency cover the runtime characteristics of various tasks in the system. Next, for each task, we first get the frequency supported by the CPU, then set the governor of all CPU cores as user space, so that we can manually set the CPU running frequency; then run the training set, so that we can collect data. Then, the training module of RBFNN is used to train the RBFNN with the collected training data until the trained RBFNN meets the pre-set error requirements. Finally, the system characteristic data at the current time point can be input into the RBFNN, and the predicted CPU core utilisation rate for the next time period can be obtained.

6 Experimental validation and experiment analysis

After training the radial basis network, the utilisation of each CPU core in the next stage can be obtained. Next, the frequency-scaling operation is performed. Before starting the frequency scaling, the adjustable frequency value of the current host CPU must be known. The Cpufreq subsystem in the kernel provides the operation interface, so we can get the frequency data supported by the processor through calling the system interface. Since we need to manually change CPU frequency, we need to set the governor to user space, get the governor data supported by the OS before you start setting up the governor. Then, we could decide to determine the frequency of each CPU core for the next time period according to the utilisation of each core. Our strategy is simple and fast, which is shown as follows: (i) when the predicted utilisation is ≥80% and the current running frequency is not the maximum, and the CPU utilisation of the previous three periods is >80%, we change the frequency to the highest level; otherwise, we will only turn up the frequency one level. (ii) When the predicted utilisation is <30% and the current running frequency is not the minimum, and the CPU utilisation in the first three time periods is <30%, we change the frequency to the minimum level; otherwise, we will only turn down the frequency one level. (iii) For other situations, our approach does not change system frequency.

When the predicted utilisation rate is ≥80% in the next time period, it indicates that CPU will be busy in the next time period, and we should increase CPU frequency. If CPU utilisation of the first three points in the record is >80%, it indicates that CPU resource requirement is very large at this stage, and the frequency needs to be set to the highest directly to meet the requirements of the task. Otherwise, we only need to raise the frequency by one level to meet the task requirements. When the predicted CPU utilisation is <30% for the next time period, it indicates that CPU will be idle during the next time period, and we should lower the CPU frequency. If the CPU utilisation of the first three points in the record is <30%, it indicates that the CPU resource requirement is very small at this stage, and the CPU frequency needs to be set to the minimum directly to achieve maximum energy saving. Otherwise, we only need to lower the CPU frequency by one level.

The threshold of the above policy refers to the Ondemand policy in the Linux kernel, the upper and lower thresholds of the Ondemand are 30 and 80% by default.

After setting the governor to user space, we can manually set the CPU frequency. At this point, a scaling_set speed file recording frequency will appear under the folder/sys/devices/system/cpu/cpuX/cpufreq/. This file shows the frequency value of each core. We only need to write the corresponding frequency value to the file to implement the core frequency setting. Since each core has its own frequency setting file, it is necessary to write the corresponding frequency value to the scaling_set speed file of each core.

We use KA3005P power supply to provide regulated power, and HOIKI 3334 to get the instantaneous and cumulative power consumption during system running. The hardware and software configuration of the system is shown in Table 1. Three types of tasks are selected: CPU-intensive task, IO-intensive task and hybrid task, and the characteristics of tasks are shown in Table 2. To verify the effectiveness of the proposed DVFS_RBF method for dynamic frequency scaling, comparative analysis is carried out.
with other three strategies of Performance, Ondemand and PAST, next we discuss the operating frequency, energy consumption and performance of different tasks under these four strategies. Among them, Performance is the energy-saving strategy of the Linux kernel, Ondemand is the default energy-saving strategy of the Linux kernel, and PAST is the classical strategy proposed by Weiser et al. [10].

Fig. 2–4 show CPU frequency changes in periods of Task1, Task2 and Task3 under Performance, Ondemand, PAST and RBF_DVFS algorithms, respectively. Since Performance strategy always works at the highest frequency regardless of workload changing, no matter what type of task we run, CPU always runs at the highest frequency, which is shown in these three figures.

For CPU-intensive task Task1, since CPU-intensive task needs a large amount of CPU resources during runtime, this will lead to CPU at a high frequency for most of the time. Besides the Performance strategy, the system frequency difference under the other three strategies is not large, which is within 5%. RBF_DVFS is slightly better than the other two strategies because of the prediction of system tasks in advance.

For IO-intensive Task2, the RBF_DVFS strategy in this paper is obviously superior to other strategies. When running this type of task, each core will distribute tasks equally, resulting in the average utilisation rate of each core, and it is not high, but the frequency changes quickly, often around 30%. When Performance strategy is working, CPU still runs at the highest frequency, as Ondemand strategy is based on the CPU utilisation rate of the previous period of time to frequency modulation, so there are two quite different CPU utilisation rates, which lead to the response delay. Our RBF_DVFS improves the prediction of CPU in advance, so the frequency variation granularity and response time of the system are short.

For mixed task Task3, the RBF_DVFS strategy is also superior to other strategies. As the execution time of the task is limited, the frequencies of Ondemand strategy and PAST strategy are higher than RBF_DVFS strategy. Also, the CPU utilisation of this type task is in a floating state, Ondemand strategy and PAST strategy delay the judgement of CPU utilisation in the next period, which affects the timely adjustment of system frequency. At the same time, PAST strategy increases or decreases the CPU frequency one level; it will cause the delay of frequency adjustment.

Fig. 5 shows the energy consumption ratio of Performance, Ondemand and PAST compared with RBF_DVFS. The positive ratio shows that this method consumes more energy than RBF DVFS, while the negative value shows the opposite result. As can be seen from this figure, these methods have different energy consumption behaviours. For CPU-intensive Task1, Performance strategy, Ondemand strategy and PAST strategy consume 9.5, 5.3 and 7.6% more energy than DVFS RBF, respectively. For IO-intensive Task2, our proposed method is better than the other three methods; the other three methods consume more energy, and the proportions are 9.1, 3.9 and 6.6%, for mixed Task3, the proportion is 24.2, 4.5 and 8.4%. From the results of the above tasks, it can be concluded that RBF DVFS has average energy savings of 17.6, 4.5, and 7.5% over Performance.
the performance loss is within acceptable range, which proves that working at the highest frequency, so the time consumed by the CPU strategy need more time to finish the task. For IO-intensive Task2, the proportions are 3.24, 2.68 and 1.68%; on average, it consumes 2.61% more time than three methods. This is because RBF DVFS reduces the system runtime saved for energy saving while adapting the CPU frequency, but reduces the overall power consumption. Especially for the performance strategy, because it keeps the CPU working at the highest frequency, so the time consumed by the program under this strategy is minimum. Experiments show that the performance loss is within acceptable range, which proves that our RBF DVFS is effective.

7 Conclusion
This paper presents an RBF-DVFS utilisation prediction model, which is implemented by RBFNN. It is used to fit the functional relation between the characteristic related to CPU resources and CPU utilisation of CPU in the next period. Through the fitting of RBFNN, we improve the prediction of CPU utilisation in more fine-grained and the frequency scaling of traditional DVFS. Finally, we evaluate our model by different tasks, especially for the mixed-type task, our method is better the results indicate our model is effective.

8 Acknowledgments
This work was supported in part by the State Key Program of National Natural Science Foundation of China under Grant No. 61802162. The Science and Technology Key Project of Henan province under Grant nos. 182102210105, 182102210434 and 192102210259, Major Special Science and Technology Project of Henan Province (No. 171100110600) and the Science and Technology Foundation of Guizhou Province ([2019]1447).

9 References
[1] Tafidis, P., Bandeira, J.: ‘Inter-regional European cooperation platform to promote sustainable transport through ICT: an overview of best practices’, Proc. Tenth Int. Conf. Pervasive Technologies Related to Assistive Environments, Rhodes, Greece, 2017, pp. 255–260
[2] Averginou, M., Bertoldi, P., Castellazzi, L.: ‘ Trends in data centre energy consumption under the European code of conduct for data centre energy efficiency’, Energies, 2017, 10 (10), p. 1470
[3] Wang, X., Wang, W., Yang, L.T., et al.: ‘A distributed HOSVD method with its incremental computation for big data in cyber-physical-social systems’, IEEE Trans. Comput. Soc. Syst., 2018, 5 (2), pp. 481–492
[4] Xie, J., Deng, Y., Min, G., et al.: ‘An incrementally scalable and cost-efficient interconnection structure for data centers’, IEEE Trans. Parallel Distrib. Syst., 2017, 28 (6), pp. 1578–1592
[5] Kuman, K.: ‘Overview of next-generation green data center’, Fujitsu Sci. Tech. J., 2012, 48 (2), pp. 177–183
[6] Hu, C., Hu, C., Deng, Y., et al.: ‘QoS promotion in energy-efficient datacenters through peak load scheduling’, IEEE Trans. Cloud Comput., 2018, DOI: https://doi.org/10.1109/TCAD.2019.2940681
[7] Yao, F., Demers, A., Shenker, S.: ‘A scheduling model for reduced CPU energy’, Proc. IEEE 36th Annual Symposium of Computer Science, Washington, D.C., USA, 1995, pp. 374–382
[8] Cong, J., Yuan, B.: ‘Energy-efficient scheduling on heterogeneous multi-core architectures’, Proc. 2012 ACM/IEEE Int. Symp. Low Power Electronics and Design, Redondo Beach, CA, USA, 2012, pp. 345–350
[9] Oh, S., Kim, J., Kim, S., et al.: ‘Task partitioning algorithm for intra-task dynamic voltage scaling’, 2008 IEEE Int. Symp. Circuits and Systems, Seattle, WA, USA, 2008, pp. 1228–1231
[10] Weisser, M., Welch, B., Demers, A., et al.: ‘Scheduling for reduced CPU energy’, Mobile Computing, Boston, MA, 1994, pp. 449–471
[11] Govil, K., Chan, E., Wasserman, H.: ‘Comparing algorithm for dynamic speed setting of a low-power CPU’, Proc. First Int. Conf. Mobile Computing and Networking, Berkeley, CA, USA, 1995, pp. 13–25
[12] Pering, T., Burd, T., Brodersen, R.: ‘Voltage scheduling in the IA32 microprocessor system’, Proc. 2000 Int. Symp. Low Power Electronics and Design, 2000 ISLPED, IEEE, Rapallo, Italy, 2000, pp. 96–101
[13] Choi, K., Lee, W., Soma, R., et al.: ‘Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dynamics’, Proc. Conf.- Computer-aided design IEEE Computer Society, San Jose, CA, USA, 2004, pp. 29–34
[14] Choi, K., Soma, R., Pedram, M.: ‘Dynamic voltage and frequency scaling based on workload decomposition’, Proc. 2004 Int. Symp. Low Power Electronics and Design, 2004, pp. 174–179
[15] Pedram, M., Cheng, W.C., Danu, K., et al.: ‘Frame-based dynamic voltage and frequency scaling for a MPEG decoder’, IEEE/ACM Int. Conf. Computer-Aided Design, USA, 1996, pp. 66–69
[16] Kobayashi, N., Enomoto, T.: ‘A low power 90 nm CMOS motion estimation processor implementing dynamic voltage and frequency scaling (DVFS) and fast motion estimation’, 2008 IEEE Int. Symp. Circuits and Systems 2008 ISCAS, Seattle, WA, USA, 2008, pp. 1672–1675
[17] Pering, T., Burd, T., Brodersen, R.: ‘The simulation and evaluation of dynamic voltage scaling algorithms’, Proc. 1998 Int. Symp. Low Power Electronics and Design, USA, 1998, pp. 68–75
[18] Grunwald, D., Morrey lli, C.B., Lewis, P., et al.: ‘Policies for dynamic clock scheduling’, Proc. Fourth Symp. Operating Systems Design & Implementation, Berkeley, CA, USA, 2000, vol. 209, pp. 83–95
[19] Bhodapak, A.S., Smith, J.E.: ‘Managing multi-configurable hardware via dynamic working set analysis’, ACM Sigarch Comput. Archit. News, 2001, 30 (2), pp. 233–244
[20] Isci, C., Martonosi, M.: ‘Detecting recurrent phase behavior under real-system variability’. 2005 Int. Proc. IEEE Workshop Characterization Symp., Austin, TX, USA, 2005, pp. 13–23
[21] Liu, S., Qiu, Q., Wu, Q.: ‘Energy aware dynamic voltage and frequency selection for real-time systems with energy harvesting’. Date Proc. Conf. Design Automation & Test in Europe, Munich, Germany, 2008, pp. 236–241
[22] Isci, C., Buyukyel, Y., Naz, C.Y.: ‘On-line energy efficient multi-core global power management policies: maximizing performance for a given power budget’. 2006 39th Annual IEEE/ACM Int. Symp. Microarchitecture MICRO-39, Orlando, FL, USA, 2006, pp. 347–358
[23] Khan, J., Bilavan, S., Belady, C.: ‘Energy analysis of a DVFS-based power strategy on ARM platforms’. 2012 IEEE Fault Tolerance on Computing Systems (FTTC), Paris, France, 2012, pp. 1–4
[24] Liu, S., Qiu, M.: ‘A discrete dynamic voltage and frequency-scaling algorithm based on task graph unrolling for multiprocessor system’. 2009 Int. Conf. Scalable Computing and Communications; Eighth Int. Conf. Embedded Computing, Dalian, People’s Republic of China, 2009, pp. 3–8
[25] Liu, X., Liu, P., Hu, L., et al.: ‘Energy-aware task scheduling with time constraint for heterogeneous cloud datacenters’. ‘Concurrency and Computation: Practice and Experience’, 2019
[26] Koeun, P.J., Mushalb, M.E.: ‘DVFS-power management and performance engineering of data center server clusters’. 2019 15th Annual Conf. Wireless On-demand Network Systems and Services, 2019, pp. 91–98
[27] Wang, X., Yang, L.T., Xie, X., et al.: ‘A cloud-edge computing framework for cyber-physical-social services’, IEEE Commun. Mag., 2017, 55 (11), pp. 80–85
[28] Ghonoodi, A.: ‘Green energy-aware task scheduling using the DVFS technique in cloud computing’. J. Adv. Comput. Res., 2019, 10 (10), pp. 1–10
[29] Wang, X., Yang, L.T., Li, H., et al.: ‘NOQA: a nested anti-collision algorithm for RFID systems’, ACM Trans. Embedded Comput. Syst. (TECS), 2019, 18 (4), p. 32
[30] Youssar, N., Samir, B.A.: ‘Intelligent online configuration for DVFS multiprocessor architecture: fuzzy approach’. Int. Conf. Industrial, Engineering and Other Applications of Applied Intelligent Systems, Cham, Graz, Austria, 2019, pp. 616–627
[31] Ma, Y., Zhou, J., Chantem, T., et al.: ‘On-line resource management for improving reliability of real-time systems on ‘big/ little’ type MPPSoCs’, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., 2018, 39 (8), pp. 88–100
[32] Ma, Y., Zhou, J., Chantem, T., et al.: ‘Improving reliability of soft real-time embedded systems on integrated CPU and GPU platforms’, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., 2019, DOI: https://doi.org/10.1109/TC.2018.2886187
[33] Sorkhpour, B., Obermaisser, R., Bebawy, Y.: ‘Optimization of frequency scaling in time-triggered microcontroller architectures using coordinated meta-scheduling’. 38th 2019-Automotive micro- Electronics Tenth GMM-Symph.VDE, 2019, pp. 1–10

This is an open access article published by the IET under the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0/)