J Electr Eng Technol.2016; 11(5): 1289-1298
http://dx.doi.org/10.5370/JEET.2016.11.5.1289

ISSN(Print) 1975-0102
ISSN(Online) 2093-7423

A Modeling Approach for Energy Saving Based on GA-BP Neural Network

Junke Li*, Bing Guo†, Yan Shen, Deguang Li* and Yanhui Huang***

Abstract – To cope with the increasing scale of scientific data and computational complexity of daily data, more and more cores have been integrated into GPU(Graphic Processing Units) and its working frequency is continually upgrading, which makes it being widely used in general computing for assisting CPU to accelerate program. While GPU offers powerful computing capability, the problem of the energy consumption becomes particularly prominently and it has become one of the important issues hindering development of GPU. For the purpose of solving this problem, DVFS (Dynamic Voltage Frequency Scaling) becomes an effective solution. Because the previous works only focus on single component and use linear relationship to do DVFS without considering energy saving of other units in system at software runtime, therefore we propose an energy saving model (CDVFS) of considering the characteristics of both GPU and memory at software runtime based on GA-BP (Genetic Algorithm-Back propagation) neural network to make better use of the relationship between components for energy saving. Firstly, the model assumes that functional relation between the software runtime characteristics of GPU and memory and the appropriate frequency which corresponds to the GPU and memory as nonlinear. Secondly, we extract five characteristics and use GA-BP neural network to fit the nonlinear functional relation. At last, experiments demonstrate the effectiveness of the approach and reasonableness of assumption, and also show that CDVFS can get average energy savings of 17.06% compared with previous works within acceptable performance loss.

Keywords: Energy saving, Model, Software runtime characteristics, GA-BP neural network, DVFS

1. Introduction

The ICT (information and communication technology) industry is growing particularly fast in the world, and its carbon emissions are still in constant growth. A report of Gartner in 2007 points that the ICT industry accounts for approximately two percent of global carbon dioxide emissions and this number will be doubled by 2020. Energy conservation and emission reduction in the field of ICT has become an urgent issue to be faced by all the countries in the world.

In 21st century, increasing scale of data puts forward higher requirement on computing speed than ever before. The data scale of simulation for nuclear explosion, satellite image data processing, weather forecast, molecular dynamics and genetic engineering is in TB (Tera-bytes, 2^{40} bytes) or even PB (Peta-bytes, 2^{50} bytes) orders of magnitude and this is also the same in games and high-definition video decoding in our daily life. These data need to be processed by processor with the computing power of trillion times per second or more. With the development of semiconductor technology, the improved performance of GPU has far exceeded than that of CPU. In order to take full advantage of powerful computing resources of GPU, the programmable pixel processing module is added to graphics pipeline to make the GPU gradually be widely used in the field of general purpose computing area. The enhancement of GPU computing power, however, is at the cost of increasing energy consumption. While the integration of GPU increases rapidly, energy consumption has gradually become one of the main factors that restrict GPU to be applied in the wide application. To alleviate the problem, reducing power consumption on the premise of guaranteeing performance has become the focus of attention.

Current energy consumption of CMOS (complementary metal oxide semiconductor) circuit is mainly composed of static and dynamic energy consumption. The latter occupies the dominant position in the total energy consumption and it has approximately exponential relationship with running frequency. Therefore dynamically adjusting the operating frequency based on status of program can significantly reduce the energy consumption. DVFS is an energy saving mechanism by dynamically scaling frequency and voltage, changing and switching the state of memory bank and I/O device so as to make hardware provide appropriate performance based on the state of program. Its appearance provides a new opportunity for researching energy saving

† Corresponding Author: College of Computer Science, Sichuan University, China. (guobing@scu.edu.cn)
* College of Computer Science, Sichuan University, China. (ljk2006ljk@163.com)
** School of Control Engineering, Chengdu University of Information Technology, China. (sheny@cuit.edu.cn)

Received: November 10, 2015; Accepted: April 27, 2016

on GPU. The rest of paper is organized as followings. In section 2, the related work will be presented. Section 3 shows the energy saving model in details. The measurement of software runtime characteristics and the fitting method of GA-BP neural network are described respectively in Section 4 and Section 5. Section 6 gives experimental results. Finally, conclusions about our work will be showed in Section 7.

2. Related works

DVFS provides a new way for researching energy saving and a number of techniques have been proposed in the past. These previous DVFS-related works can be classified into the following groups.

The first category of these technologies uses deadline of the task to implement algorithms. [2] proposes a DVFS approach under constraint of task deadline. [3] reduces the global energy dissipation by proposing the DVFS policy under the constraint of the end-to-end delay. [4] identifies different traffic distribution of time slot in NoC (network on a ship) by using the worst-case packet deadline to do DVFS in order to reduce the energy consumption of the NoC. These policies know the task arrival times and deadlines in advance and scale frequency to reduce energy dissipation while meeting real-time deadlines. The second category uses compiler or application assisted information to guide DVFS scheduling for energy saving. In [5], the author proposes energy management of application layer by using information provided by the application. In [6], authors propose the compiler assisted DVFS and insert relevant DVFS instruction into the program to realize the purpose of saving energy depending on the detected type of the program that compiler provided. This group of methods needs additional code added to the program before it is executed on the system or compiler support for performing DVFS. The third category uses software runtime characteristics or statistics to guide DVFS for energy saving. [7] proposes a DVFS approach that uses the relation between IPC (instructions per cycle) information and frequency during the software runtime to reduce energy dissipation under the condition of the memory stall. [8] proposes a DVFS scheduling approach of multicore CPU that uses CPI, MAPI and the service level agreement (SLA) request information between the client and the server. In [9], DVFS is formulated into a multiple choice knapsack problem to minimize total energy consumption by giving MPI (last level cache misses per instruction) distribution of the program, the corresponding energy consumption and other statistics. [10] proposes a regression-based DVFS model by distinguishing the tasks on-chip or off-chip. An online DVFS is proposed in [11] based on the distribution of memory stall time, behavior of program phase and computational workload at software runtime. A DVFS scheduling approach is proposed in [12] using relationship between memory frequency and the threshold. [13] uses the relationship between frequency and weight vector that can be calculated by the parameter CPI and μ to implement the DVFS algorithm. For memory power consumption, [14] indicates that the memory power consumption accounts for 30% of the system so that the research on memory has a certain extent of significance for energy saving. [15] proposes the DVFS strategy based on memory power consumption, which illustrates that reducing memory frequency will not produce significant impact on the performance of the task when the memory bandwidth of program requirements is not very big. [16] scales memory frequency through minimizing system SER (system energy ratio) by using performance counters and memory power consumption performance model. Although using the runtime characteristics of program to guide the DVFS can achieve the aim of energy saving, these studies only take runtime characteristics of single component into consideration and neglect the energy saving of coordinated GPU, memory and I/O.

Due to researches on coordinated component DVFS are relatively rare and they mainly use linear approaches, in this paper, we mainly discuss the problem of coordinated GPU and memory DVFS for energy saving. Inspired by the software runtime characteristics [7-15](reducing processor frequency in memory intensive program to save energy will not affect the overall system performance and decreasing memory frequency in compute-intensive programs to save energy will not affect the overall system performance), we propose a model(CDVFS) of coordinated GPU-memory energy saving approach based on GA-BP neural network, which jointly scales the GPU and memory frequency to achieve the goal of energy saving by the key characteristics that can be extracted from hardware at software runtime.

3. Energy Saving Model

It is widely accepted by scholars that using the runtime characteristics to guide DVFS can save energy. But these studies that guided by characteristics mainly use linear approach and regard characteristics and the corresponding frequency of hardware as N linear relationship [5, 7-13]. The linear relationship can be described as the following equation:

$$F = N_1 P_1 + N_2 P_2 + \cdots + N_n P_n \tag{1}$$

where, F denotes the current frequency. P_n indicates the value of software runtime characteristics. N_n is the weight of each characteristic. Although using the model can get the result, it has the following deficiencies, Such as 1: Hypothesis of the linear relationship between runtime characteristics and frequency lacks practical support. 2: single indicator will not fully reflect frequency demand of out-of-order execution, data dependence and branch prediction phenomena. Current approaches are not systematically considered into the frequency relevance between the GPU and memory at software runtime [7-16]. Based on what have been analyzed above, we argue that there is a nonlinear relationship (Linear functional relationship can be considered as a special nonlinear functional relationship) between appropriate frequency and software runtime characteristics. By analyzing them, we can get the following relationship model:

$$\begin{aligned} F(Comp1, Comp2) &= f(SRC) \\ &= f(AI, GMCR, MAPI, CPI, FL) \end{aligned} \tag{2}$$

where, *F*(*Comp1*, *Comp2*) indicates the appropriate frequency of each component (*Comp1*,*Comp2*) at software runtime. *SRC* is the metric of the running software, *f* stands for the nonlinear functional relation. In dealing with nonlinear relationship, BP neural network is able to compute mathematical relationship with high accuracy no matter how complicated it would be. Therefore, in this paper, BP neural network is used as *f* function. *AI*, *GMCR*, *MAPI*, *CPI*, *FL* and its corresponding concrete measurement will be discussed in the next section. The energy saving model based on GA-BP neural network can be divided into the following five steps:

(1) Suppose there is a non-linear function relationship between appropriate frequency and characteristics at software runtime.
(2) Measure characteristic quantities that are related to the frequency of GPU and memory at software runtime.
(3) Obtain appropriate frequency at software runtime by running benchmarks through adjusting frequency.
(4) Preprocess the characteristic quantities to fit the input of the model.
(5) Fit the nonlinear function *f* through the GA-BP neural network. The input of GA-BP neural network is characteristics at software runtime and the output is the appropriate frequency of GPU and memory at software runtime.

4. Measure the characteristic quantities

Selecting characteristics of GPU and memory can affect the result of the model, thus this paper selects five metrics (arithmetic intensity, global memory to computation cycle ratio, memory access per instruction, instruction per cycle, frequency level) related to appropriate frequency of hardware from the perspective of running software behavior. From the view of macroscopic, executing program goes through the process that the programs are loaded into memory from the peripheral and fetch the instruction from the memory to run. From the view of microcosmic, software has feature of program locality at software runtime. Therefore, the five metrics can comprehensively reflect the intensive degree of GPU and memory at software runtime and they can be used to reflect the appropriate frequency related to the workload. Characteristic and its measurement will be made concrete analysis in the following.

4.1 Arithmetic intensity

AI (Arithmetic intensity) is first proposed in [17] to construct the roofline model and evaluate performance of program. It provides the optimization direction for programmers to judge whether the program is compute intensive or memory intensive. It is the ratio of floating point operations per second to accessing bytes per second from memory, which is shown in Eq. (3).

$$AI = \frac{flops}{bytes/s} \tag{3}$$

where, *flops* (floating point operations per second) indicate current peak floating point during the software running. *B ytes/s* is the memory bandwidth used of the program. Eq. (3) directly expresses the executing program focus on which components at software runtime. The large value of AI indicates the program is emphasized on the processor.

4.2 Global memory to computation cycle ratio

To verify whether the program is compute intensive or memory intensive, [18] use a rule of thumb named “Global memory to computation cycle ratio (GMCR)" to measure the density of memory access during program running. The value of the GMCR can be obtained by Eq. (4).

$$GMCR = \frac{Number(Global\ Memory\ Transactions)}{Number(Computation\ Instructions)} \tag{4}$$

Number (*Global Memory Transactions*) and *Number* (*Computation Instructions*) in Eq. (4) respectively indicate number of executing memory access and number of executing computations instruction per unit time. Like arithmetic intensity, GMCR metric reflects the intensive degree of the GPU and memory from the perspective of instruction.

4.3 Memory access per instruction

The operations related to the memory are completed by the instruction during the program execution. The more times the memory is being accessed by the program in a unit time, the more intensive the memory resources are being exploited. MAPI (memory access per instruction) metric in [19] is able to quantify such relationship. Therefore, it can reflect the demand on memory bandwidth of program at the software runtime. [8] uses MAPI to change the frequency of memory and it can be calculated by using Eq. (5).

$$MAPI = \frac{Number(Memory\ Access)}{Number(Executed\ Instructions)} \quad (5)$$

where, *Number* (*Memory Access*) indicates the Number of memory access, and *Number* (*Executed Instructions*) indicates the number of executed instructions.

4.4 Instruction per cycle

For the same program the shorter execution time is, the more instructions are executed per unit time. When IPC is increased, the utilization rate of processor increases, too. This property is used by [8] to indicate the intensive degree of the program in the processor. IPC is the average number of executed instructions per clock cycle and it can be expressed by Eq. (6).

$$IPC = \frac{Number(Executed\ Instructions)}{Number(processor\ Cycles)} \quad (6)$$

In Eq. (6), *Number* (*Processor Cycles*) and *Number* (*Executed Instructions*) respectively denotes cycles that processor used and the number of instructions executed.

4.5 Frequency level

Due to the above metrics of the same program are influenced by the current frequency of GPU, so it should be considered as a characteristic. After the GPU frequency is determined, the frequency of memory can be defined by the above metrics. In this paper, the level of current frequency is measured by Eq. (7).

$$FL = f_{present} / max(f) \quad (7)$$

where, $f_{present}$ represents the current frequency of GPU. *max*(*f*) is the maximum running frequency of the GPU.

5. Nonlinear Fitting of GA-BP Neural Network

There is a nonlinear relationship between the runtime characteristics and the corresponding appropriate frequency of GPU and memory, and rational expression of this relationship can guide DVFS at software runtime. Therefore, how to express this kind of nonlinear is particularly important. BP neural network is a kind of numerical approximation method without establishing mathematical equation. It can approximate any nonlinear function and has good fitting ability by learning the input vector and output vector. In order to express the relationship more accurately, we use the genetic algorithm to optimize the initial weights and threshold values of BP neural network. Therefore, we adopt the GA-BP neural network to fit this relationship.

5.1 Nonlinear fitting procedure of GA-BP neural network

Steps of using GA-BP neural network to realize nonlinear fitting are detailed as followings:

(1) Adjust the frequency under the guidance of software runtime characteristics, measure five software runtime characteristics from samples, and record the corresponding frequency of the GPU and memory.
(2) Preprocess those five characteristics and the corresponding frequency and use them as the input and output value of the BP neural network respectively.
(3) Design the structure of the BP neural network (layers, number of node and transfer function).
(4) Use the genetic algorithm to optimize weights and threshold values of BP neural network and determine the correlation between input and output.
(5) Put the collected characteristics within software runtime into the established BP neural network and then get the output value. What's more, we compare the processed output values with actual value to verify the validity of the model.

5.2 Design GA-BP neural network

GA-BP neural network refers to using genetic algorithm to optimize values of the weights and threshold to make BP neural network achieve the best fitting effect. The structure of BP should be firstly determined (number of hidden layer, number of nodes in hidden layer, transfer function of each layer) and then use the genetic algorithm (coding method, fitness function, selection, crossover operation and mutation operation) to optimize the values of weight and threshold. What's more, approximation error, convergence speed and learning rate of neural network are factors that need to be considered.

5.2.1 Determine the number of hidden layer

The number of hidden layer will affect the prediction accuracy of network. If too much, it will lead to long training time and over fitting phenomenon. Robert Hecht-Nielson in [20] proves that BP network of single hidden layer can approximate continuous function in any closed interval. So BP neural network of single hidden layer can finish the mapping of *n* to *m* dimensions.

5.2.2 Determine the number of nodes in hidden layer

The number of nodes in hidden layer will also affect the prediction accuracy of BP neural network, but how many nodes in hidden layer lacks the guidance of the scientific

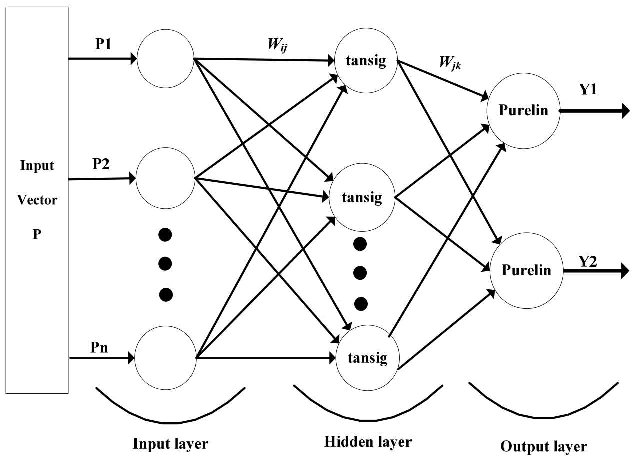


Fig. 1. Structure of BP neural network

method. Generally, the following empirical formula is always used to get the best numbers of nodes *l*:

$$l \leq \sqrt{n+m}+a \tag{8}$$

where, *l* is the number of nodes in hidden layer. *n* is the number of nodes in input layer. *m* is the number of nodes in output layer. *a* is constant number between 0 ~ 10. In this paper, there are five inputs(AI, MAPI, GMCR ,IPC and FL) in the input layer and two outputs (the appropriate frequency of GPU and memory) in the output layer, so the scope of *l* is: 4~13.

5.2.3 Determine transfer function of each layer

Using different transfer functions will have different effects on accuracy and learning rate, moreover, there is a variety of options for the transfer function of hidden layer and output layer, such as *hardlim*, *hardlims*, *purelin*, *tansig*, *logsig*, etc. Through the experiment, we can see that using the *tansig* and *purelin* as the transfer function can achieve satisfactory results in convergence speed and error. Therefore, we use the *tansig* and *purelin* as the transfer function for hidden layer and output layer. The structure of the BP neural network is shown in Fig. 1, where *P* is the input vector, w_{ij} is the weight between input layer and hidden layer, w_{jk} is the weight between hidden layer and output layer, *Y* is the output vector, *tansig* and *purelin* is respectively the transfer function of hidden layer and output layer.

5.2.4 Population coding

The population coding uses real number coding. Size and composition of coding will be given after the structure of the network is determined. Population coding consists of four parts: the weights between input layer and hidden layer, thresholds of hidden layer, thresholds of output layer and the weights between hidden layer and output layer. Population initialize randomly after population coding is determined.

5.2.5 Fitness function

The fitness function determines the optimized direction of the algorithm. In this paper, we use sum of absolute value between the predicted output and the desired output as the individual fitness value *Fit*. It can be calculated from the Eq. (9).

$$Fit = \sum_{i=1}^{n} abs(y_i - o_i) \tag{9}$$

where, *n* is the number of output nodes. y_i is the expectation output of the *i*th node. o_i is the predicted output of *i*th node.

5.2.6 Select operation

In this paper, we use roulette method which is the strategy based on the proportion of the fitness values as the selection operation in the genetic algorithm. The selection probability P_i of each individual *i* is got by Eq. (10).

$$P_i = \frac{1/Fit_i}{\sum_{j=1}^{n}(1/Fit_i)} \tag{10}$$

where, Fit_i is the fitness value of individual *i*, and *n* is the number of individuals.

5.2.7 Crossover operation

Crossover operation is to choose two individuals from population to produce new excellent individual through exchanging and recombining them. We use the real number crossover method as crossover operation. The crossover operation on *j* bit between the *k*th individual a_k and the *l*th individual a_l is as follows:

$$\begin{cases} a_{kj} = a_{kj}(1-b) + a_{lj}b \\ a_{lj} = a_{lj}(1-b) + a_{kj}b \end{cases} \tag{11}$$

where, *b* is a random number on interval [0, 1].

5.2.8 Mutation operation

Mutation operation means choosing an individual from the population and selecting one gene in individual to generate better individual by mutation. The method of mutation operation is as follows:

$$a_{ij} = \begin{cases} a_{ij} + (a_{ij} - a_{max}) * f(g) & r > 0.5 \\ a_{ij} + (a_{min} - a_{ij}) * f(g) & r \leq 0.5 \end{cases} \tag{12}$$

where, a_{ij} is the jth gene of ith individual. a_{max} is the upper bound of the gene. a_{min} is the lower bound of the gene. $f(g)=k(1-g/G_{max})^2 a_{ij}$, where, k is a random number, g is the number of current iteration. G_{max} is the largest number of evolution. r is a random number on interval [0,1].

After the structure of BP neural network is determined, we use genetic algorithm to optimize parameters of network. For better measuring characteristics, the typical benchmarks of CUDA are selected to test, such as batchCUBLAS, BlackScholes, FastWalshTransform, Matrixmultiplication, Matrixblas and MatrixTranspose. For our convenience, we use the abbreviation batch, BS, FWT, MT, MB and MTP to represent them respectively. That the software runs in compute intensive, the memory intensive and mixed phase has the heuristic significance for the frequency adjustment, so the selected software should contain these phases as many as possible. From this aspect, we select the MT as compute intensive program, choose MTP as memory intensive program, and select FFT as mixed program. We use GT740M GPU platform to run these typical software, and record their characteristics, corresponding frequency of GPU and memory through the performance tuner. In order to better approximate real frequency, we get the optimal weight and threshold of each layer under the following parameter settings: population size is 20, evolution time is 100, crossover probability is 0.55 and the mutation probability is 0.18. Fig. 2 shows the weight and threshold values of BP neural network under the mean square error of 10^{-5} that obtained by genetic algorithm, where $w1$ is 13*5 weight matrix between input layer and hidden layer and it is corresponding to the w_{ij} in Fig. 1. 13 and 5 are respectively the number of nodes in hidden layer and the number of nodes in input layer. $b1$ is the 13*1 threshold matrix of hidden layer and the 13 is equals to the number of nodes in hidden layer. $w2$ and $b2$ are the same meaning as the $w1$ and $b1$. The only difference is that $w2$ is the weight matrix between hidden layer and output layer and it is corresponding to the w_{jk} in Fig. 1 and $b2$ is the threshold matrix of output layer.

$$w1=\begin{Bmatrix} 0.5883 & 1.3225 & -1.1545 & 1.1878 & 0.7132 \\ -1.3576 & -0.6234 & -1.1273 & 0.1947 & -1.3055 \\ -0.2407 & -1.9552 & 0.2505 & -1.1372 & -0.2142 \\ -1.3234 & 0.0054 & -1.4226 & 0.9281 & 0.9680 \\ 1.4158 & 0.3088 & 1.3755 & -1.4514 & 0.7003 \\ -0.2938 & 1.4919 & 1.4984 & 0.7132 & 0.2230 \\ -1.3041 & -0.8196 & -0.1019 & 2.6129 & -0.0627 \\ -1.7926 & 1.4652 & 0.5621 & -0.4694 & -1.3399 \\ -0.8409 & -0.5375 & -1.5280 & 1.4812 & -0.4792 \\ 1.3156 & 1.0435 & -1.0652 & -0.8883 & -1.2973 \\ -1.2733 & 1.5167 & 0.0205 & -0.0620 & 1.7068 \\ 1.4195 & 0.7486 & -0.6260 & 2.0819 & -1.2282 \\ -1.3513 & -1.4744 & 0.6694 & -0.4943 & 0.4230 \end{Bmatrix}$$

$$b1=\begin{Bmatrix} -2.4676 \\ 2.0607 \\ 1.7795 \\ 1.4624 \\ -1.0585 \\ 0.3018 \\ 0.2624 \\ -0.1716 \\ -0.7601 \\ 0.6165 \\ -1.6005 \\ 1.8967 \\ -1.8619 \end{Bmatrix},\quad w2=\begin{Bmatrix} -0.3117 & 0.5314 \\ -0.3360 & 0.3345 \\ -0.4433 & 0.8032 \\ 0.4533 & -0.5769 \\ 0.2170 & 1.0678 \\ -0.8210 & 0.1001 \\ 0.1011 & 1.7410 \\ 0.2062 & 0.4254 \\ -0.3831 & -0.8418 \\ -0.8066 & -0.2981 \\ -0.2881 & -0.7466 \\ -0.2686 & -0.7748 \\ -1.3802 & -0.7709 \end{Bmatrix}, b2=\begin{Bmatrix} -0.7756 \\ 0.1047 \end{Bmatrix}$$

Fig. 2. Optimized weight and threshold value of neural network

6. Nonlinear Fitting of GA-BP Neural Network

In order to verify the validity of the CDVFS on energy saving, this paper compares and analyzes the results through the implement of CDVFS and actual running the approach of [13] and [16] , where [13] is used for scaling frequency of processors and [16] is used for adjusting frequency of memory. For fair comparison, we combine approach of [13] and [16] to illustrate results. Because the existing approach lacks coordinated GPU and memory DVFS, we call the combination approach as traditional DVFS, which we use TDVFS for convenience. The parameter in [13] is set as follows: the value of CPI_{avg} is got by running compute-intensive benchmark at each frequency and CPI_{base} is the actual CPI value. We set the performance constraint to 2% in [16] compared with non-DVFS. [13] and [16] are mainly for single component, so we set frequency of the memory in [13] to 980MHZ and the GPU in [16] to 980MHZ.

As the runtime characteristics used in this paper has difficulties to obtain, we stimulate the CDVFS proposed in this paper after getting the experimental data from the GT740M platform and recording characteristics parameter at software runtime through the visual profiler. We use [21] to get the energy consumption of actual program execution and the simulated program execution. Fig. 3-Fig. 8 is the

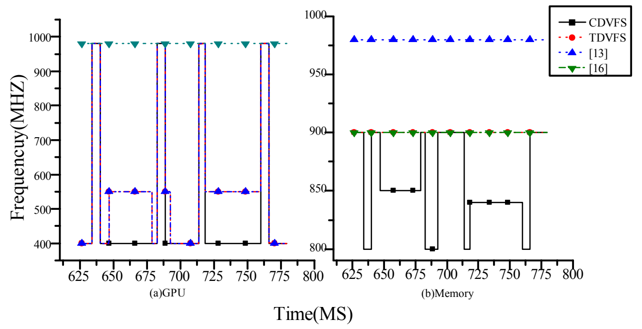


Fig. 3. Frequency variation of batch

effect of frequency variation by CDVFS, TDVFS, [13] and [16] under the program of batch, BS, FWT, MB, MT and MTP. The black solid line with square, red dot line with circle, dash blue line with upper triangle and dash dot olive line with lower triangle represent CDVFS, TDVFS, [13] and [16] in each figure respectively. The left part and right part of each figure show respectively the frequency of GPU and frequency of memory under above approaches.

Frequency of processor behaves as follows under each approach. Because [16] is only for scaling frequency of memory, we set the frequency of processor at the constant number 980MHZ for each benchmark under it in the left part of each figure. During the execution of MB in Fig. 6, there is no difference in frequency change among CDVFS, TDVFS and [13]. This also appears at execution of MT and MTP in Fig. 7 and Fig. 8. These phenomena show that each approach has the same effect on type of MT, MB and MTP program. Runtime frequency of BS in Fig. 4 under CDVFS is slightly lower than that under TDVFS and [13]. This phenomenon indicates these approaches can adapt and meet the demand on frequency of the BS-type program. FWT program in Fig. 5 has the same phenomenon with the BS program and the difference between them is that the frequency generated by CDVFS is higher than that of the TDVFS and [13] in three periods. This shows that CDVFS is more sensitive in the aspect of satisfying the frequency of the FWT-type program. During execution of the batch program in Fig. 3, the frequency of CDVFS, TDVFS and [13] coincides with about two-thirds of total time. In the time that frequency doesn't overlap, the BS-liked phenomenon account for most of the time, this shows that batch has the BS-liked effect under CDVFS. The reason for phenomena above lies in that these approaches have different principles in adjusting frequency. From results got by all types of applications above, we can see that these approaches can scale frequency of processor at runtime and they have similar trends of frequency change in most cases.

Frequency of memory behaves as follows under each approach. As [13] is only for adjusting frequency of processor, frequency of memory is ran at the constant number 980MHZ for each benchmark in right part of each figure. MB and MTP in Fig. 6 and in Fig. 8 show the similar trend of frequency change. Frequency of MTP is the same under CDVFS, TDVFS and [16]. This also happened in MB except for 4% of the total time that frequency of CDVFS is lower than that of TDVFS and [16]. This phenomenon shows that these approaches for MB and MTP on frequency adjustment of memory have almost the same effect. Runtime frequency of BS and MT in Fig. 4 and in Fig. 7 shows that CDVFS is slightly lower than those under TDVFS and [16]. The frequency of FWT in Fig. 5 under CDVFS is slightly higher than that of TDVFS and [16]. These phenomena indicate these approaches can adapt and meet the demand on frequency of the BS-type, MT-type program and FWT-type program. At the batch program in Fig. 3, the frequency obtained by CDVFS changes frequently while the frequency got by TDVFS and [16] is constant. The time of frequency overlapped accounts for about 14.54% of the total time compared with the TDVFS and [16]. The time that is not overlapped accounts for most of the time and the frequency of CDVFS is lower than that of TDVFS and [16] in this period. Thus it

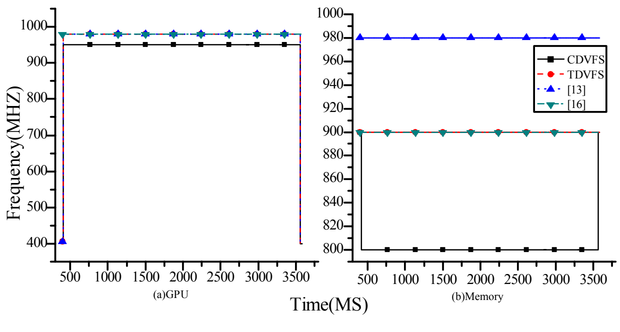


Fig. 4. Frequency variation of BS

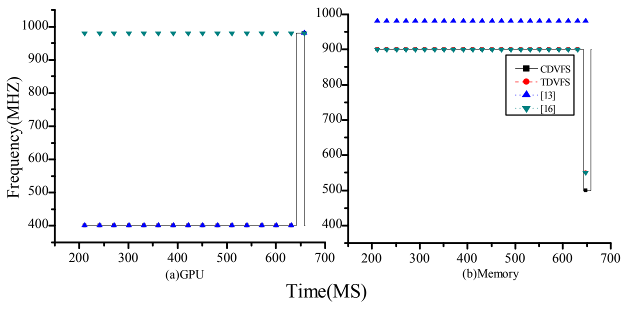


Fig. 6. Frequency variation of MB

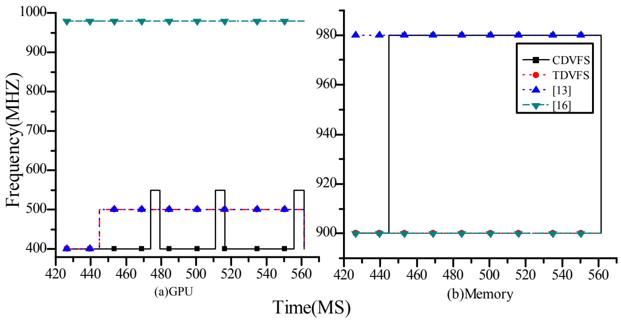


Fig. 5. Frequency variation of FWT

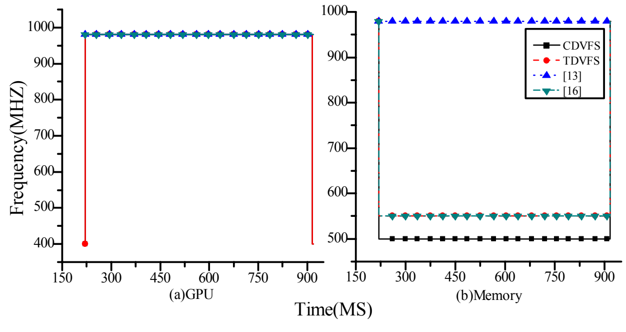


Fig. 7. Frequency variation of MT

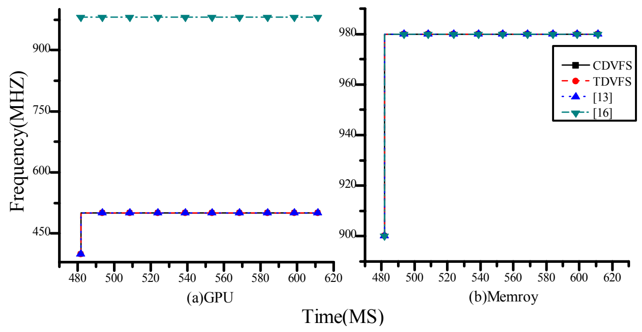


Fig. 8. Frequency variation of MTP

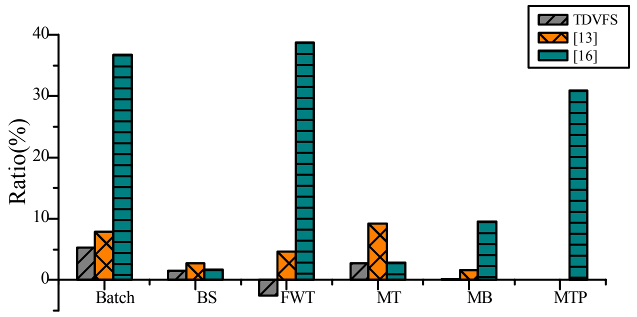


Fig. 9. Energy comparison

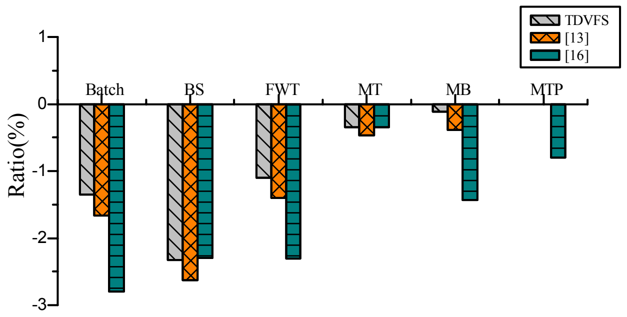


Fig. 10. Performance comparison

indicates that batch has the same effect as BS under the CDVFS. This is because these approaches have different principles in adjusting frequency of memory. From above results got by all types of applications, we can see that these approaches can scale frequency of memory at runtime.

Fig. 9 shows energy consumption of TDVFS, [13] and [16] compared with CDVFS, and we use ratio to express. The ratio of positive value shows that the approach consumed more energy than CDVFS and the negative value illustrate opposite result. From the figure, we can see these approaches have different behaviors of energy consumption. TDVFS respectively consumes more energy about 5.32%, 1.53% and 2.72% than that of CDVFS in Batch, BS and MT. It saves energy about 2.58% compared with that of CDVFS in FWT and has almost equal energy compared with that of CDVFS in MB and MTP. [13] respectively consumes more energy about 7.88%, 2.7%, 4.66%, 9.16% and 1.62% than that of CDVFS in Batch, BS, FWT, MT and MB. It has equal energy compared with that of CDVFS in MTP. [16] respectively consumes more energy about 36.74%, 1.73%, 38.72%, 2.78% 9.5% and 30.92% than that of CDVFS in Batch, BS, FWT, MT, MB and MTP. Consequently, CDVFS respectively gets average energy saving of 2.38 %, 8.68%, 40.14% than TDVFS, [13] and [16]. The average total energy saving is 17.06%. The main reason lies in that [13] and [16] only take single component into consideration, ignoring energy saving of other component. TDVFS considers energy saving of both GPU and memory, but each component has its own principle and ignores the relationship of energy saving between them. CDVFS treats GPU and memory as potential energy saving components and it finds the relationship that can get better energy saving effect between them.

For DVFS, the performance should be also taken into consideration. Fig. 10 is the performance of TDVFS, [13] and [16] compared with CDVFS, and we also use ratio to express. The meaning of negative and positive value is the same as described in Fig. 9. From the figure we can see the time for CDVFS is more than that of TDVFS, [13] and [16]. CDVFS consumes more time than TDVFS in Batch, BS, FWT, MT and MB respectively with 1.34%, 2.32%, 1.1%, 0.34% and 0.12%. It consumes almost equal time as TDVFS in MTP. It also spends more time than [13] in Batch, BS, FWT, MT and MB respectively with 1.65%, 2.62%, 1.39%, 0.46% and 0.38%. It has almost equal time compared with [13] in MTP. [16] spends less time than CDVFS in Batch, BS, FWT, MT, MB and MTP respectively with 2.79%, 2.29%, 2.3%, 0.34%, 1.42% and 0.8%. CDVFS respectively consumes more time than TDVFS, [13] and [16] in average with 1.74%, 2.17%, 3.32%. The average total time saving is 2.41% compared with CDVFS. This is because CDVFS scales both frequency of GPU and memory to save energy and then slow down the speed of each component. From the experiment, we can see that the performance loss is kept at acceptable value, which shows the CDVFS is effective.

7. Conclusion

General-purpose computing based on GPU attracts widely attention from researchers for its performance and energy efficiency. In order to better tap the energy saving effect of GPU in the general-purpose computing, we propose a modeling approach for energy saving called CDVFS. It has following advantages over existing approaches. Firstly, our approach is easy to be understood. Secondly, it can well reflect the demand on frequency of program. Finally, CDVFS can get average energy savings of 17.06% compared with previous works within acceptable performance loss. Experiments validate our model for energy saving and the results indicate that our approach is effective and the assumption is reasonable.

Acknowledgements

This work was supported in part by the State Key Program of National Natural Science Foundation of China under Grant No. 61332001, the National Natural Science Foundation of China under Grant No. 61272104 and 61472050, the Applied Basic Research Program of Sichuan province under Grant No. 2014JY0257, 2015GZ0103 and 2014-HM01-00326-SF. We also acknowledge group of embedded real-time system for their effective advice and constructive suggestions.

References

[1] Top 500 Supercomputer Sites Webpage, November 2015. http://www.top500.org.

[2] V. Hanumaiah and S. Vrudhula, "Temperature-aware DVFS for hard real-time applications on multicore processors," IEEE Trans.Computers., vol. 61, no. 10, pp. 1484-1494, Oct.2012.

[3] T. Horvath, T. Abdelzaher, K. Skadron, et al. "Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control," IEEE Trans. Computers., vol. 56, no. 4, pp. 444-458, Apr. 2007.

[4] J.Zhan, N.Stoimenov, J.Ouyang, et al, "Optimizing the NoC Slack Through Voltage and Frequency Scaling in Hard Real-Time Embedded Systems," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems., vol. 33, pp. 1632-1643, Nov. 2014.

[5] Z. Zong, A. Manzanares, X. Ruan, et al. "EAD and PEBD: two energy-aware duplication scheduling algorithms for parallel tasks on homogeneous clusters," IEEE Trans.Computers., vol. 60, no. 3, pp. 360-374, Mar.2011.

[6] Q. Wu, M. Martonosi, D. W. Clark, et al, "A dynamic compilation framework for controlling microprocessor energy and performance," in Proc.micro, 2005, pp. 271-282.

[7] R. Kotla, S. Ghiasi, T. Keller, et al, "Scheduling Processor Voltage and Frequency in Server and Cluster Systems," in Proc.IPDPS, 2005.

[8] Z.Zhang, J.M.Chang, "A cool scheduler for multi-core systems exploiting program phases," IEEE Trans. Computers., vol. 63, no. 5, pp. 1061-1073, May. 2014.

[9] X.Chen, C.Xu, R.P. Dick, "Memory access aware on-line voltage control for performance and energy optimization," in Proc.ICCAD, 2010, pp. 365-372.

[10] K.Choi, R.Soma, M.Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance tradeoff based on the ratio of off-chip access to on-chip computation times," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems., vol. 24, no. 1, pp. 18-28, Jan.2005.

[11] J. Kim, S. Yoo, C. M. Kyung, "Program phase-aware dynamic voltage scaling under variable computational workload and memory stall environment," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems., vol. 30, no. 1, pp. 110-123, Jan.2011.

[12] S. Kim, H. Eom, HY. Yeom, et al, "Energy-centric DVFS controlling method for multi-core platforms," Computing, vol. 96, no. 12, pp. 1163-1177, Dec.2014.

[13] G. Dhiman and T. S. Rosing, "Dynamic voltage frequency scaling for multi-tasking systems using online learning," In Proc. ISLPED, 2007, pp. 207-212.

[14] K. Li, X. Tang, K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," IEEE Trans. Parallel and Distributed Systems., vol. 25, no. 11, pp. 2867-2876, Nov.2014.

[15] H. David, C. Fallin, E. Gorbatov, et al, "Memory power management via dynamic voltage/frequency scaling," In Proc.ICAC, 2011, pp. 31-40.

[16] Q. Deng, L. Ramos, R. Bianchini, et al, "Active low-power modes for main memory with memscale," IEEE Micro., vol. 32, no. 3, pp. 60-69, May/June. 2012.

[17] S.Williams, A.Waterman, D.Patterson, "Roofline: an insightful visual performance model for multicore architectures," ACM. Commun., vol. 52, no. 4, pp. 65-76, Apr.2009.

[18] S. Ryoo, CI. Rodrigues, S. S. Baghsorkhi, et al. "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," In ACM Proc. PPoPP, 2008, pp. 73-82.

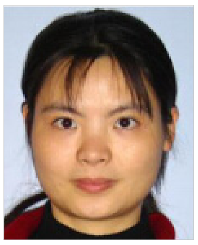
[19] Intel Corp., Intel-64 and IA-32 Architectures Software Developers Manual, vol. 3B: System Programming Guide, Part 2, Nov. 2008.

[20] A. Sarajedini, R. Hecht-Nielson, "The best of both worlds: Casasent networks integrate multilayer perceptrons and radial basis functions," in Proc.IJCNN, 1992, pp. 905-910.

[21] H. Nagasaka, N. Maruyama, A. Nukada, et al, "Statistical power modeling of GPU kernels using performance counters," in Proc. Green Computing, 2010, pp. 115-122.

Junke Li He received his BS degree in Computer Science from the Henan Polytechnic University in 2010, and he received his MS degree in Computer Science from Southwest University in 2013. He is currently a PhD candidate in the School of Computer Science, Sichuan University. His research interests include parallel computing and green computing.

Bing Guo He received his BS degree in Computer Science from the Beijing Institute of Technology in China, and MS and PhD degrees in Computer Science from the University of Electronic Science and Technology of China, China, in 1991, 1999, and 2002, respectively. He is currently a Professor in the School of Computer Science at the Sichuan University, China. His current research interests include embedded real-time system and green computing.



Yan Shen She received her MS degree in Mechatronics Engineering and PhD degree in Measuring and Testing Technology and Instruments from University of Electronic Science and Technology of China in 2001 and 2004 respectively. Currently she is a Professor in the Control Engineering College, Chengdu University of Information and Technology. Her main research interests include distributed measurement systems, embedded system development, wireless sensor networks, robotics.

Deguang Li He received his BS degree in Computer Science from the PLA Information Engineering University, in 2010, and he received his MS degree in Computer Science from Northeastern University, in 2012. He is currently a PhD candidate in the School of Computer Science, Sichuan University. His research interests include embedded real-time system and green computing.

Yanhui Huang He received his BS and the MS degree in Radio Electronics and Computer Science from Sichuan University in 1997 and 2002 respectively. Currently he is a lecture in the school of computer science at Sichuan University. His current research interests include embedded real-time system and green computing.