Low-power Scheduling Framework for Heterogeneous Architecture under Performance Constraint

Junke Li[1,2], Bing Guo[1*], Yan Shen[3] and Deguang Li[1]
[1] College of Computer Science, Sichuan University
Chengdu, Sichuan,6100675 - China
[e-mail: ljk2006ljk@163.com,guobbing@scu.edu.cn, lideguang.00@163.com]
[2] School of Computer and Information, Qiannan Normal Nniversity for Nationalities
Duyun, Guizhou, 558000 - China
[e-mail: ljk2006ljk@163.com]
[3] School of Control Engineering, University of Information Technology
Chengdu, Sichuan, 610225 – China
[e-mail: Yshen426@163.com]
*Corresponding author: Bing Guo

Received April 14, 2019; revised October 31, 2019; accepted January 21, 2020; published May 31, 2020

Abstract

Today's computer systems are widely integrated with CPU and GPU to achieve considerable performance, but energy consumption of such system directly affects operational cost, maintainability and environmental problem, which has been aroused wide concern by researchers, computer architects, and developers. To cope with energy problem, we propose a task-scheduling framework to reduce energy under performance constraint by rationally allocating the tasks across the CPU and GPU. The framework first collects the estimated energy consumption of programs and performance information. Next, we use above information to formalize the scheduling problem as the 0-1 knapsack problem. Then, we elaborate our experiment on typical platform to verify proposed scheduling framework. The experimental results show that our proposed algorithm saves 14.97% energy compared with that of the time-oriented policy and yields 37.23% performance improvement than that of energy-oriented scheme on average.

Keywords: Energy saving, heterogeneous architecture, integer programming, resource allocation, scheduling framework

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, 61472050 and 61802162. The Applied Basic Research Program of Sichuan province under Grant No. 2014JY0257, 2015GZ0103 and 2014-HM01-00326-SF. The Science and Technology Foundation of Guizhou Province under Grant NO. [2019]1447. the Natural Science Foundation of the Ministry of Education of Guizhou Province under Grant NO. [2019]071, [2019]205, [2018]080, and [2018]439; the Nature Science Foundation of Qiannan under Grant NO. [2018]05; The Nature Science Foundation Qiannan Normal University for Nationalities under Grant NO. QNSY2018BS012, QNSY2018021and QNSYRC201714.

1. Introduction

Multi-core CPU and many-core GPU have become the most important processors in computer systems. As a general-purpose processor, the design of CPU must balance the needs of various tasks; therefore, most of the transistors in CPU are used to make huge caches and complex control logic, and the operating units do not occupy much of the area. At present, increasing the complexity of the control logic, enlarging cache size and increasing the frequency have not much help improving the performance of a single core, so more CPU cores have integrated into a single chip. Meanwhile, due to the features of parallel graphics rendering, GPU also has a large number of parallel processors naturally. How to make rational use of multi-core processing resources of CPU and GPU becomes the focus of current research, therefore, a large number of articles [1], [2], [3] have compared and discussed their advantages and disadvantages.

For achieving high performance, computer architects, programmer and researchers are now moving away from the debate on CPU or GPU which has the better performance toward CPU and GPU collaboration to get the higher performance than single one[4], [5]. The CPU and GPU collaboration refers to intelligently combine the features of CPU and GPU together to achieve higher computing power by rationally allocating each program across them to avoid and reduce idle time of CPU and GPU and then get superior performance compared with that of a single one. This paradigm is called as heterogeneous computing (HC) by most scholars; this system is known as heterogeneous computing system (HCS). In HCS, CPU and GPU are collectively referred to as processing unit (PU) in order to describe convenience. HC has attracted everyone's attention, for example, in TOP500 and Green500, most supercomputers have CPU and GPU [6], [7].

In 2015, the performance of Tianhe2 that ranked Top 1 in Top500 list researches 54.9PFLOPS. Although heterogeneous multi-core system has higher performance, its power consumption is generally higher. The power of Tianhe2 is up to 17.81 MW, after opening the cooling system reaches 24MW, and electricity cost of working one hour is up to more than ten millions [6]. The problem caused by power consumption, such as rising cost of chip packaging and cooling, increasing the probability of IC's (integrated circuit) invalidation under high temperature (if temperature increase every 10 degrees, the system failure rate will be doubled typically), decreasing of the system reliability, has become the important obstacle that blocks the development of high performance. Therefore, it has important significance to build an effective mechanism to reduce the energy consumption of HCS.

In order to reduce the energy consumption of HCS, many scholars have put forward a variety of methods, models to solve it. At present, according to the granularity of program, the problem can be divided into single program allocation on PUs and multi-programs allocation on PUs. The former refers to allocating single program that can be seen as several workloads to each PU by consuming minimum energy (ASP); the latter involves allocating a set of multi-programs that each program can be seen as several workloads to each PU by consuming smallest energy (AMP). In General, ASP is only for a few specific programs and these programs needs to be changed to adapt to each PU processing mode; however, AMP has wide adaptability and only considers the desired goals without changing the program. Although the advantage of AMP is obvious, existing researches also has deficiency, such as assuming the power is constant[33], getting the parameters by pre-running before scheduling[34], achieving energy savings in a performance-optimized manner[32]. In order to effectively alleviate the

above problems as well as make HC be widely used to adapt to the diversity of programs, this article discusses the AMP approach. At present, in the HC field, most scheduling strategies focus on the current tasks, historical tasks and performance improvement performed by PUs, and lack the global allocation of tasks within a given time. In order to improve the relevant theory of HC scheduling method, this paper studies the energy saving of HC under performance constraints from a global perspective. From this point of view, we propose a task scheduling framework with power-aware, low overhead and good portability for HC. It first obtains the number of tasks and corresponing power consumption on each PU, and then formalizes the energy optimization problem as the 0-1 knapsack problem under performance constraints. For getting the number of tasks, we can count the nunmber of task in program, and for getting the power consumption, we can use existing methods[14]-[17]. In solving the global minimization problem, from mathematics perspective, 0-1 programming has been proved to be an effective method to deal with the objective functions and constraints. From the point of view of the computer, 0-1 programming has been considered to be effective and convenient in programming practice. Therefore, our scheduling framework is suitable for tackling this problem. To the best of our knowledge, the energy saving framework is first proposed by us.

This paper makes the following contributions.

- We analysis the current research wroks of heterogeneous computing system and point out their deficiency;
- We propose task scheduling framework for saving energy in heterogeneous system which consists the CPU and GPU;
- We formalize the program scheduling problem into integer programming which takes performance into consideration. After solving the problem, we can get the result which can be run on corresponding PUs;

The rest of paper is structured as follows. Section 2 shows related works; energy saving framework is introduced in section 3; section 4 presents the program scheduling method; our proposed method is verified and compared in section 5; section 6 summarizes the work of this paper.

2. Related Work

There are many works focus on the heterogeneous computing. Some of them improve the performance of system, while others focus on their energy issue. From previous studies related to our work, we conclude these approaches as following groups.

Heterogeneous computing for performance: Augonnet et al. [8] proposes a uniform runtime execution model—StarPU for allocating single program to different PUs. It has five strategies to get higher performance: Greedy policy with support for priorities, Greedy policy without support for priorities, Greedy policy based on Work Stealing, Random weighted by processor speeds and Heterogeneous Earliest Finish Time. Like [8], Luk et al. [9] proposes the Qilin framework for mapping single program on an HCS which includes CPU and GPU. It provides an API for writing parallelizable programs and uses a training phase to create a performance model for each task on each PU. Using different inputs and a linear performance model, optimal workload division can be computed and dynamic compilation is done to instantiate the chosen distribution. Belviranli et al. [10] presents a dynamic load-balancing technique for loop iterations on HCS. Their technique works in two phases. In the first phase, the relative performance of PUs is estimated by experimenting with different task size

allocations to the PU. In the second phase, the remainder and majority of computations are performed based on the relative performance values obtained in the first phase. The second phase utilizes a self-scheduling algorithm to achieve load balancing. The proposed algorithm dynamically resizes blocks to minimize underutilization, thus yielding the shortest execution time. Grewe et al. [11] proposes a machine learning approach to predict optimal processing resource allocation for an Opencl program based on the analytical result of compiler. It uses Clang compiler and PCA to get the features of program and then inputs these features to the two-stage classier that consists of SVM to get the final scheduling results. Boratto et al. [12] uses static scheduling to divide the workload of matrix computation on a CPU and a GPU for solving the problem of landform attributes representation. Bernabé et al. [13] proposes a workload allocation approach for accelerating the 3D-Fast Wavelet Transform. This approach first profiles performance of CPU and GPU and then allocates the workload to them based on the proportion of their performance. Jiménez et al. [14] proposes a scheduling method for performance based on the performance history. It compares the performance of proposed approach with that of Fist-Free, First-Come-First-Served, and the experiments show that the proposed method has higher performance than that of other methods.

Heterogeneous computing for Energy-aware system: Li et al. [17] proposes an energy saving approach for GPU using the BP neural networks to guide the DVFS. Paul et al. [18] relies on coordinating DVFS for both CPU and GPU to realize a coordinated energy management algorithm for integrated CPU-GPU systems. These papers mainly focus on the DVFS technology and don't consider the task scheduling. Wang et al. [19] studies the energy saving of PCM and DRAM memory in the system and proposes a two-phase approach to solve hybrid main memory address mapping problem. Wang et al. [20] studies the energy consumption optimization problem of real-time streaming applications in multiprocessor system. Considering transition overhead, inter-core communication, discrete voltage levels and utilizing the DVS and DPM technology, it proposes a two-phase approach to solve above problem. Liu et al. [21] proposes a technique to improve the efficiency of large scale heterogeneous clusters (including several CPU-GPU nodes) based on the waterfall model. Their technique changes the possible energy states of busy, spare and sleep for each node. In addition using three states to save energy, it also makes use of task scheduling on available nodes to improve performance and adjusts CPU voltage to save energy. Machovec et al. [23] first uses the information, such as the utility functions presented in [22] to express the performance of system, an estimated time for computing time matrix and an average power consumption matrix to establish utility per energy heuristic approach for scheduling, and then designs, analyzes and compares it with four utility-aware heuristic methods, three FCFS-based methods and a random method. At the same time, an energy filtering method is proposed to limit the maximum energy consumption of each PU. Oxley et al. [24] studies the problem of static resource allocation for multiple independent tasks in a heterogeneous cluster system environment. This article define energy robustness as the probability that the energy budget is not violated and makespan robustness as the probability a makespan deadline is not violated and then model the execution time of program by probability density function. Based on the above information, authors design and analyze several energy aware resource allocation methods under energy and performance constraints.

Above approaches are conducted on cluster environment, while our work is on a single heterogonous node which consists of CPU and GPU. Liu et al. [25] proposes a method of allocating the High Performance Linpack program to CPU, GPU and FPGA. It uses linear programming to allocate the workload of each processing resource to achieve the purpose of energy saving after obtaining the performance parameters of program by the profiler and

getting the parameters of PU by pre-running. The proposed method needs to manually rewrite the code of the target PUs after obtaining the assigned load ratio for each PU. Ma et al. [26] proposes an energy management framework based on the characteristics of the single load to be distributed in the CPU and GPU for load balancing and reducing idle time and idling energy for HCS. The framework also scales the frequency and voltage on CPU and GPU respectively to reduce energy consumption. Barik et al. [28] proposes an energy-aware scheduler (EAS) method of allocating a single program to the CPU and GPU for reducing power consumption. This article uses the information, such as the power of program that got by polynomial approximation, the characteristics of program that got by profiling present in [27] and the length of execution time(i.e., memory- or compute-bounded, short or long-burst) to schedule the single program and then adjust the α parameter to achieve the purpose of saving energy. In order to achieve the goal of energy saving, Ma et al. [29] proposes a holistic energy management framework for single program on CPU-GPU architectures. The framework has two layers; the first layer is the dynamic allocation; it distribute single program to CPU and GPU based on the characteristics of the program. The second layer is the frequency scaling layer; it adjusts the frequency of the processor to save energy. In [29], four kinds of dynamic allocation schemes for assigning single program (Simple Heuristic with fixed step size, an Improved Heuristic with adaptive step size, and two binary search-style algorithms) are compared and analyzed respectively and their advantages and disadvantages are analyzed. Totoni et al. [30] proposes a method of assigning a single program to a CPU and a GPU for energy saving. It studies the effect of different mapping policy and demonstrates that software pipelining can really improve the energy efficiency. Kiran et al. [31] studies mapping programs to heterogeneous multicores using different criteria such as performance, power and energy, and proposes an approach to select optimal mapping for a given program based on its profiling result. Wang et al. [32] proposes a power-efficient work distribution method for single application on a CPU-GPU heterogeneous system. Their method could coordinate inter-processor work distribution and scale frequency of per-processor to minimize energy consumption under a given scheduling length constraint, which uses linear method to predict execution time of each PUs. These approaches are designed for single program, while our work focuses on multi-programs.

Based on the [14], Choi et al. [33] points out the shortcomings of Alternate-assignment scheduling(AA), First-Free scheduling(FF) and Performance-History Scheduling methods(PH), and then proposes the estimated-execution-time scheduling(EET) mode after adding the remaining time table. By considering the time of subsequent tasks, the experiments show that the performance of EET scheduling is better than that of AA, FF and PH. Due to the performance improvement makes EET consume less energy than other methods. Hamano et al. [34] presents a dynamic scheduling energy saving method for allocating multi-programs on heterogeneous systems. Their method first computes the speedup ratio; then the energy delay product (EDP) is calculated and the program is allocated to the corresponding processor with the smallest EDP value. The above procedure is repeated until all programs have been allocated. This paper assumes that the power consumption is constant during program execution. Mark et al. [35] demonstrates a method of optimizing energy efficiency for multiple interdependent tasks under HCS. Their method takes the energy consumption of each task on CPU and GPU into consideration, and then uses directed acyclic graph to construct a scheduling method with minimum energy consumption. It is also considered in [35] that both CPU and GPU in the system can be turned off without overhead in idle state. Jang et al. [36] demonstrates joint optimization of both workload and power budget partitioning between the CPU and the GPU for single-programmed workloads and analyzes potential throughput

improvement of adaptive, workload-aware power allocation schemes for multi-programmed workloads on HCS. After that, authors propose a runtime algorithm that can determine optimal or near optimal workload and power budget partitioning for single-programmed workload and determines optimal Voltage/Frequency settings for multiple programs running concurrently. Although Hamano et al. [34] has multi programs allocation, it use simple allocating method and focus on selecting optimal Voltage/Frequency for CPU and GPU.

In summary, the energy saving technologies can be divided into two categories, one for allocating single workload and one for allocating multi-programs. In this paper, we mainly focus our attention on saving energy of allocating multi-programs. For tackling this problem, current existing works pay too much attention to a single task, mainly reduce the execution time, assume constant power for each PU, solve dependence on each program by using DAG and aim at DVFS technology for simple allocation. These works lack directly solving energy consumption problem for real environment under performance constraints. In contrast to above efforts, our work globally considers the goal of energy conservation from the overall tasks, the performance and relatively real power consumption data, avoiding the drawbacks of assumption of constant power consumption, local focus on energy conservation of a single task. Based on our invocation, we propose an approach for saving energy with performance constraint that considers power model, performance model and an online decision to allocate multi-programs, and validate the effectiveness and feasibility of the energy saving framework through experiments.

3. Program Scheduling Framework

The goal of our work is to rationally schedule a set of programs to execute on each PU so as to reduce energy dissipation when these programs are running. In HCS environment, there exist many programs waiting to be processed at any point in time. For achieving the goal, we propose the task scheduling framework. It includes the proposed program scheduling scheme for energy saving. The processing flow of framework is shown in **Fig. 1**.

The main functions of the framework are to profile the performance and power of each program, get the processing sequence of each PU that have the best energy saving effect under performance constraint, and then execute them on its corresponding PUs. If framework determines that current set of program is one, then the framework will use default policy. If framework determines that energy saving effect is beneficial, it will output the program sequence and then executes them on corresponding PU. The scheduling framework can be summarized into three steps.

Step 1. Profiling. Profile all original programs in the waiting pool. We get performance data of each program from CPU performance model, GPU performance model and energy data from CPU power model, GPU power model. For the CPU and GPU power model and performance model have already achieved good effect and GPU performance model is detailed in [15] and [16], therefore, this paper uses previous studies to get the energy and performance data without pre-running.

Step 2. Deciding. In the HCS environment, the number of programs to be processed is often random, when the number of programs is 1, we can allocate the program to the PU which has better energy saving effect. When the number of programs is greater than 1, we will allocate all programs to GPU if execution time of all the programs on the GPU is less than the minimum time of executing a single program on the CPU; otherwise, we will use step 3 to solve it.

Step 3. Integer programming. In allocating resources for multiple programs, we need to compromise performance and power consumption to avoid that getting the energy-saving effect is at the cost of performance. For this reason, performance needs to be used as constraints for the objective function. In most cases, the ending time of each program assigned to each PU is different, so this difference should be added as the constraint. We can get sequence of the corresponding PU by solving the objective function with minimum energy consumption under the constraint of performance.

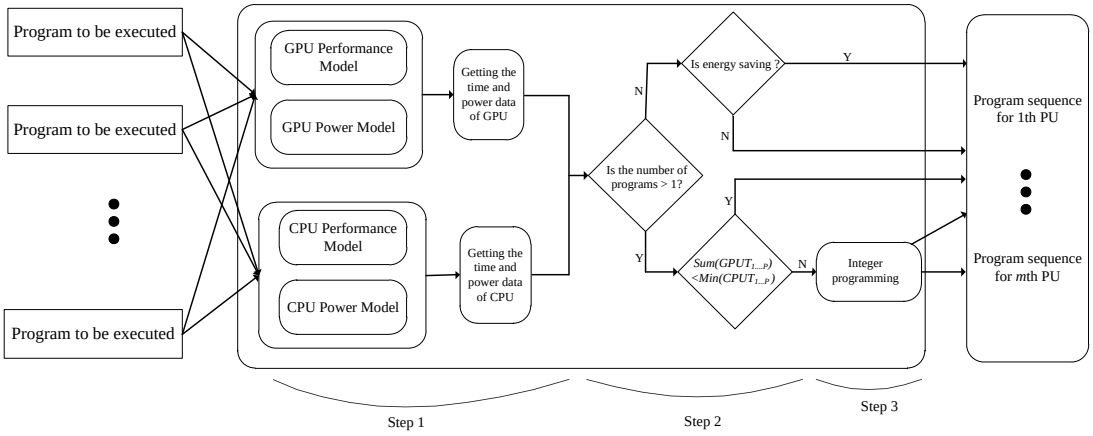


Fig. 1. Program scheduling framework

If the energy is viewed as a resource in a HCS, management of energy in the system can be regarded as the allocation of resources. Similarly, performance can also be considered as a resource in the system. The allocation of system resources are usually based on the demand of target. Therefore, in heterogeneous system environment that can run multi-programs, we can configure different resources to reduce the energy consumption of the system. In the following, we will model the energy saving problem of HCS, and named this optimization model as PCGA (Programs-CPU-GPU-Allocation). By judging how many programs are allocated to CPU and GPU can make the system more energy-efficient under the constraint of performance, PCGA model realizes the coordination of allocating the system resource when the heterogeneous system handles multi-programs. PCGA model is based on integer programming and ultimately formalized into a 0-1 knapsack problem by adding resource allocation constraints.

3.1 Symbol definition

M represents the number of CPU and GPU in systems;
N represents the number of programs to be processed in the system;
E_{ij} represents the energy consumption of the *j*th program running on the *i*th PU (0<=*j*<=*N*; 0<=*i*<=*M*);
T_{ij} represents the consumed time by the *j*th program running on the *i*th PU (0<=*j*<=*N*; 0<=*i*<=*M*);
We let *i* represent PU in system; *j* denote the program to be executed on the system; then assigning *i*th processor to complete the *j*th program has the following expression:

$$x_{ij} = \begin{cases} 1, \text{assign } i\text{th PU to execute } j\text{th program} \\ 0, \text{not assign } i\text{th PU to execute } j\text{th program} \end{cases}$$

3.2 Objective function

The goal of the problem we solve is to choose a suitable combination that minimizes the total energy consumption of the system. Based on the above variable definition, we can obtain the objective function as shown in (1) that represents the total energy consumption.

$$f = \sum_{i=1}^{m} \sum_{j=1}^{n} E_{ij} x_{ij} \tag{1}$$

3.3 Constraints

According to the requirements of the problem, each program has only one processor to run, so we get the processor constraint as shown in (2).

$$\sum_{i=1}^{m} x_{ij} = 1 (j = 1, 2, ..., n) \tag{2}$$

When assigning a processor to a program, performance should be taken into consideration. We can't unlimited reduce performance for saving energy, so we add constraint of performance to the objective function. Due to the randomness of the execution time of the program, it is not suitable for most scenes for allocating programs to the CPU and the GPU respectively in an equal time-consuming manner. In order to better adapt to the real environment, we allow unequal distribution of time on CPU and GPU. In PCGA model, we use the parameter to control running time of different PU. To do this, we use the performance tuning parameter Q to control the time on each processor. In reality, GPU is suitable for dealing with compute-intensive programs and has good performance, therefore, this paper focus on using GPU to execute more programs. Considering above factors, we obtain the time constraint as shown in (3).

$$\begin{cases} \sum_{j=1}^{n} T_{1j} * x_{1j} <= \sum_{j=1}^{n} T_{2j} * x_{2j} \\ Q \sum_{j=1}^{n} T_{2j} * x_{2j} <= \sum_{j=1}^{n} T_{1j} * x_{1j} \end{cases} \tag{3}$$

Taking (2) and (3) into account, the objective function of the problem can eventually be formalized as (4).

The value of Q in (4) is got from our previous experiments (demonstrated in **Fig. 2**) for minimizing the energy consumption under the performance constraint. In order to obtain the appropriate value, we got it through experimental comparison. In each group of experiments, the energy consumption trends obtained by different values of Q are the same as **Fig. 2**; for this reason, we only list one of them. As shown in **Fig. 2**, vertical axis represents the energy consumption of the proposed scheduling approach. Horizontal axis is the different value of Q. As shown in the graph, the value of 0.45 can provide best energy saving effect under performance constraint and the values of 0.45-0.7 can also give comparable effect. For more programs executing on GPU, we set the Q value as 0.45 in the program scheduling.

$$\begin{cases} \min \quad f = \sum_{i=1}^{m}\sum_{j=1}^{n} E_{ij}x_{ij} \\ s.t. \quad \sum_{i=1}^{m} x_{ij} = 1(j = 1,2,...,n) \\ \quad \sum_{j=1}^{n} T_{1j} * x_{1j} <= \sum_{j=1}^{n} T_{2j} * x_{2j} \\ \quad Q\sum_{j=1}^{n} T_{2j} * x_{2j} <= \sum_{j=1}^{n} T_{1j} * x_{1j} \\ \quad x_{ij} = 0或1 \end{cases} \tag{4}$$

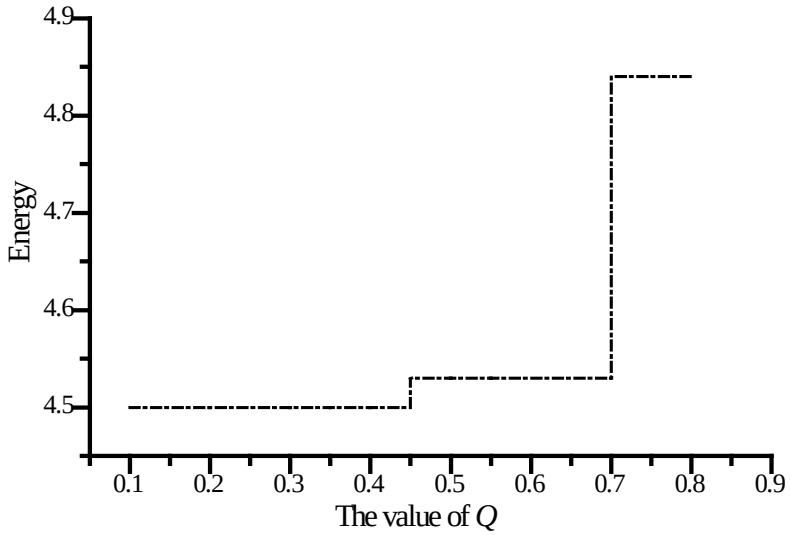


Fig. 2. Energy saving effect on different Q value

The pseudo code for the proposed scheduling scheme is described in **Fig. 3**; it uses the information from the previous power and performance estimation model for CPU and GPU. By using above time and energy information, the scheduler distributes these programs among PUs that is suitable for energy saving under the constraint of performance. The input of our proposed scheme is a set of programs which has *P*. Once started, the algorithm first obtains the prediction execution time of *P* programs using the *GetEstimatedTime()* function; then gets the predicted energy consumption of *P* programs which is obtained by *GetEstimatedEnergy ()* function. After getting above information, algorithm will judge whether *P* is equal to 1 or not; if yes, this program will be sent to the corresponding queue of PU by judging which the power consumption of each PU; if no, the program will be sent to GPU when the total time of *P* programs running on GPU is smaller than minimum time of one program running in CPU. When the total time of *P* programs running on GPU is larger than minimum time of one program running in CPU, the corresponding sequence is produced by the integer programming. In algorithm, integer programming is expressed by *Zero_One_Knapasck()* function which is implemented using equation 4. The outputs of algorithm are the sequence of programs to be executed for each PU. In **Fig. 3**, the parameter *Q* in line 6 is the same value as the *Q* in formula 4. In line 13-15, we only consider the performance factor without considering the energy

consumption factor because we prefer GPU execution when the GPU can get better performance than that of CPU.

Input: The set of P programs to be executed
Output: Sequence to be executed on each PU (SCPU, SGPU)

Algorithm:

```
1. GetEstimatedTime( CPUT_P,GPUT_P);
2. GetEstimatedEnergy( CPUE_P);
3. Using [17] to get the software energy on GPU, GPUE_P;
4. if P ==1
6.     if CPUE_P > GPUE_P && Q * CPUT_P < GPUT_P
7.         SGPU=P;
8.     else
9.         SCPU=P;
10.    end if
11.    return SGPU,SCPU;
12. else
13.    if Sum(GPUT_1....P) <Min(CPUT_1...P )
14.        SGPU=1....P;
15.        return SGPU;
16.    else
17.        Zero_One_knapsack (P); //Implementing the equation 4.
18.        return SGPU,SCPU;
19.    end if
20. end if
```

Fig. 3. Pseudo code for program scheduling

4. Experimental Evaluation

In this paper, all of experiments are conducted on Intel i5-3230M quad-core processors (8 cores in total) and Nvidia's GT740M platform which is the Kepler architecture. This version of graphics card consists of two SMs and 2GB DRAM memory. Each SM contains 192 CUDA cores. The programming environment of GPU is CUDA6.5. To demonstrate effectiveness of our algorithm, we select 44 typical benchmarks from CUDA SDK to conduct typical experiments, such as BlackScholes, fastWalshTransform, matrixMul, sortingNetworks，etc, which are widely adopted by the existing works.

To better evaluate the scheduling policy, we conduct several simulation cases to get favorable comparison such as some same program experiments and some different program experiments for scheduling. The programs we selected are BlackScholes, fastWalshTransform, matrixMul, scalarProd, transposeCoalesced, transposeNaive, Vecadd and their input data rang are respectively 12M-18M, 32M-64M, 200K-800K, 64M-256M, 500K-2M, 500K-2M, 22M-64M. For testing our approach, we give each program two versions, one is for one PU (e.g. CPU) and the other is for another PU (e.g. GPU). In experiments, we select typical scheduling approach for comparison. These approaches are Only-CPU, Only-GPU, Alternate-Assignment (AA), Estimated-Execution-Time (EET) and Energy Optimal(Opt). AA represents using CPU and GPU in round-robin fashion and doesn't consider performance and energy consumption status during selection. EET is to select the device which can

complete the incoming program more quickly by considering the performance information. Opt is to select the device which can save more energy to run the program by considering the energy information. In the experiment, since the performance and power consumption of the AA scheduling scheme are greatly affected by the executing order of program, so this paper use multiple execution to average the value of time and energy. This phenomenon also appears in the EET method, but not significant [31], so the power consumption and performance value of EET are obtained by single measurement method. We also give experiments using only on CPU and GPU scheme for fair comparison. Among them, the energy consumption of Only CPU policy includes the energy of the GPU idle state; similarly, the energy consumption of Only GPU policy includes the energy of the CPU idle state.

Since the execution sequence has some influence on the performance and power consumption of some scheduling methods, in order to better show the differences in performance and power consumption of the various scheduling methods, we select two different sets of experiments to verify. The first group of programs is to avoid the impact of executing order on the performance and power consumption; we choose a different number of the same program to eliminate this effect. In this paper, we select the number of 4, 6, and 7 of the vecadd program to verify the comparison. The second set of programs is designed to better demonstrate the universality of the proposed methods. For this, we select a number of different programs for verification. In this experiment, the number of second sets of programs we selected is the same as the number of programs in the first group; among them, the names of the four programs are BlackScholes, matrixMul, scalarProd, transposeCoalesced; the names of the six programs are BlackScholes, matrixMul, scalarProd, transposeCoalesced, transposeNaive, Vecadd; the names of the seven programs are BlackScholes, matrixMul, scalarProd, transposeCoalesced, transposeNaive, vectorAdd, fastWalshTransform.

Figs. 4 through **7** respectively show the energy consumption and execution time for same program group, energy consumption and execution time for different program group. In these figures, label (a) represents the 4 programs in the experiment; label (b) indicates the 6 programs in the experiment and label (c) denotes the 7 programs in the experiments. As shown in the **Fig. 4** and **6**, Only CPU scheduling consumes more energy than that of Only GPU Scheduling because GPU provides more powerful computation ability. This phenomenon is also happened in the **Fig. 5** and **7** on performances due to GPU has powerful throughput. Although Only GPU policy perform better than Only CPU policy in performance and energy saving, it is not always true for other scheduling scheme.

AA scheduling scheme assigns the programs alternately to the CPU and GPU and its performance and energy largely depends on sequence of incoming programs. Therefore, AA doesn't consider the property of program and this result its worse performance and less energy saving effect compared with Only GPU scheme. Due to it alternately uses the CPU and GPU, its performance is better than Only CPU policy and also has better energy saving than Only CPU in the experiments. Contrary to AA scheduling scheme, EET, Opt and our proposed scheme consider the time and performance information of program and can rational assign the program to each device depending on its purpose.

In the experiment of executing same program, **Fig. 4** shows the energy consumption of each scheduling policy and **Fig. 5** shows performance of these schemes. In **Fig. 4**, we can see that the energy consumption of EET and the Opt scheme are equal to that of Only GPU scheduling under the 4 programs, 6 programs and 7 programs experiment. This is because GPU consumes less energy and higher performance than CPU in executing vecadd program. For EET policy seeks the minimum time in executing 4, 6 and 7 programs, and this purpose will choose all programs to be executed on GPU, therefore the energy consumption of EET

policy is equal to that of Only GPU scheme. For Opt policy pursuits the minimum energy consumption in executing 4, 6 and 7 programs, and this goal will select all programs to be executed on GPU, therefore the energy consumption of Opt policy is equal to that of Only GPU scheme. For our proposed approach, its energy consumption is higher than that of Only GPU, EET and Opt method in **Fig. 4.** This is because our proposed approach has the time constraint and it seeks the minimum energy consumption under this time constraint. The EET, Opt and Only GPU scheduling scheme respectively saves 10.44%, 11.31%, 9.25% energy compared with that of our approach in 4 programs experiment, 6 programs experiment and 7 programs experiment. Due to the EET and Opt policy assign all programs to GPU, the time of these scheme is also equal to that of Only GPU policy shown in **Fig. 5**. Our approach improves the performance by 17.6%, 16.64%, 16.11% compared with EET, Opt and Only GPU scheduling scheme in 4 programs experiment, 6 programs experiment, and 7 programs experiment, respectively. This shows that our approach consumes more energy to get better performance improvement compared with EET, Opt and Only GPU scheme. The experimental result shows that it is worth doing this.

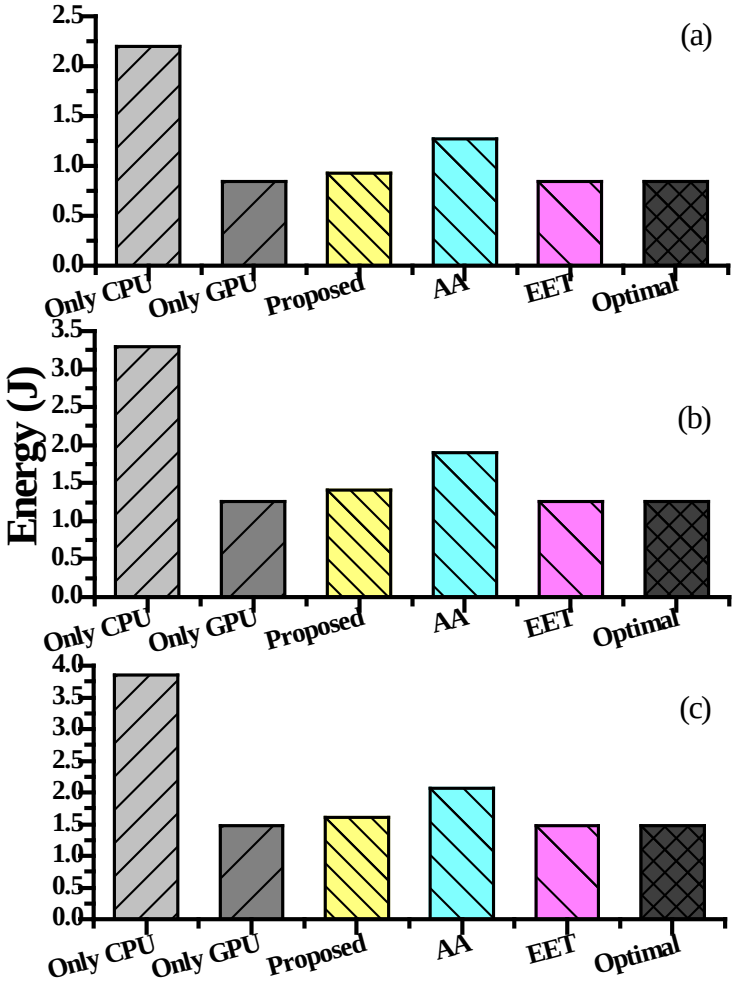


Fig. 4. Energy consumption of each scheme for same tasks

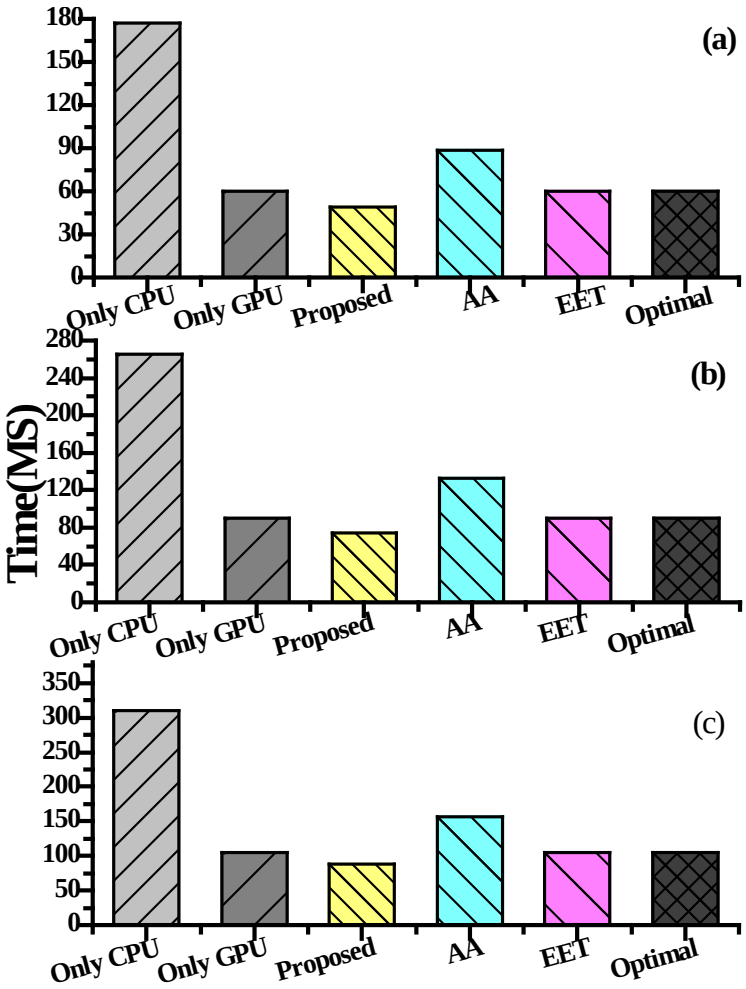


Fig. 5. Performance of each scheme for same tasks

In reality, only executing the same programs is seldom. Therefore, to approach the real environment, we also conduct three experiments which include 4 different programs, 6 different programs and 7 different programs for comparison. In **Fig. 6**(a), we can see our proposed approach consumes more energy than that of Opt policy by 4.47%, and saves 33.15%, 27.09% than that of Only GPU scheme and EET scheme. In **Fig. 7**(a), our proposed approach improves the performance by 29.09% and 32.87% compared with that of the Only GPU scheme and Opt policy, and consumes more time than that of EET scheme by 24.29%. In **Fig. 6**(b), our proposed approach consumes more energy than that of Opt policy by 31.79% and saves 11.75%, 22.45% energy than that of and EET scheme and Only GPU scheme. In **Fig. 7**(b), our proposed approach improves the performance by 45.93% and 40.87% compared with that of the Only GPU scheme and Opt policy, and consumes more time than that of EET scheme by 7.31%. In **Fig. 6**(c), our proposed approach consumes more energy than that of Opt policy by 10.04% and saves more energy than that of Only GPU scheme and EET scheme by 20.55% and 6.08%.In **Fig. 7**(c), our proposed approach improve the performance by 45.12%, 37.96% and 7.58% compared with that of the Only GPU scheme, Opt policy and EET scheme. This is because different scheme has different purpose and then result in different sequence.

The target of EET scheme is to achieve the minimum execution time of the scheduling programs; therefore it schedules the program to execute on corresponding PU which has the minimum execution time. The goal of Opt policy is to get the minimum energy consumption during execution of scheduling programs. It schedules the programs based on the energy information to get minimum energy consumption. Our proposed approach considers the energy and the performance systematically and then can balance each metric. On average, our algorithm saves 14.97% energy compared with that of the EET policy and yields 37.23% performance improvement than that of Opt scheme. From above comparison, we can see that our scheme spend more energy to get more performance improvement and can achieve the balance of the two approaches. Experiment also shows that it is worth doing this.

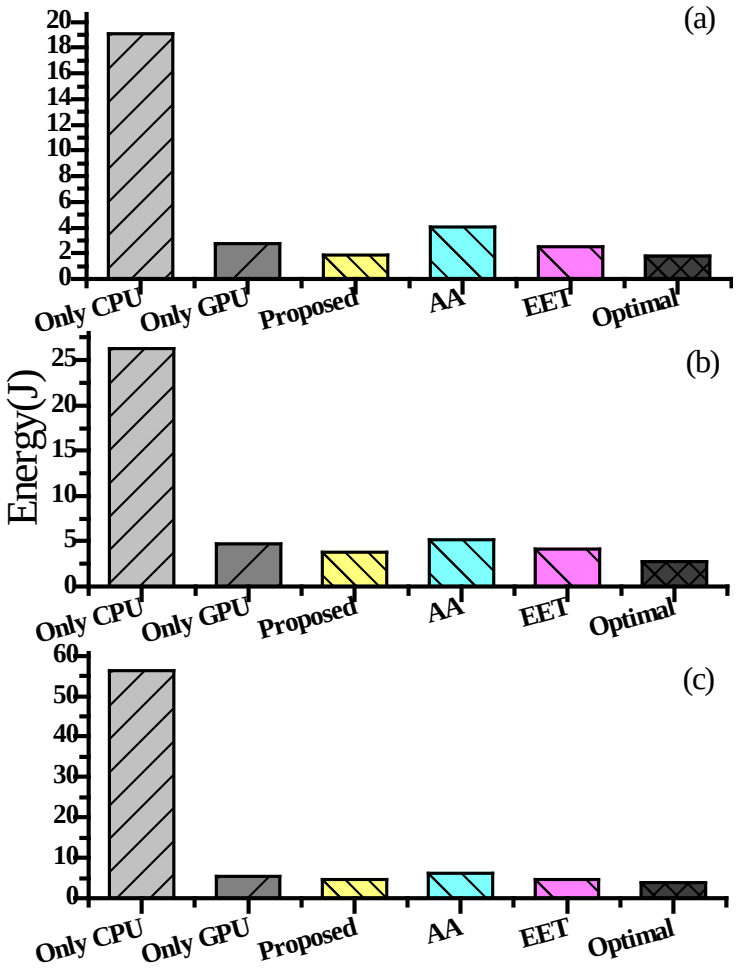


Fig. 6. Energy consumption of each scheme for different programs

Fig. 8 represents execution time ratio of CPU/GPU for PCGA and EET scheme; and **Fig. 9** demonstrates energy consumption ratio of CPU/GPU for PCGA and Opt scheme. The horizontal axis indicates the experimental cases using different scheme. For example, 7Diff_PCGA means that the experiment has 7 different programs using the PCGA scheme, 7Same_EET is the experiment has 7 same programs using EET policy and 6Same_Opt is the experiment has 6 same programs using Opt scheme. The vertical axis denotes the execution

time ratio for **Fig. 8** and energy consumption ratio for **Fig. 9**. The lower portion of the bar shows the execution time ratio / energy consumption ratio of CPU; and the upper portion of the bar denotes the execution time ratio / energy consumption ratio of GPU. As shown in the **Fig. 8**, the difference of time ratio between PCGA and EET is small for executing different programs, but the difference is obvious in executing the same program. This implies that the EET has the unbalanced work distribution between each PU. This phenomenon also appears in the **Fig. 9** on the energy consumption for Opt policy. These can also verify our approach has some advantage for EET and Opt scheduling.

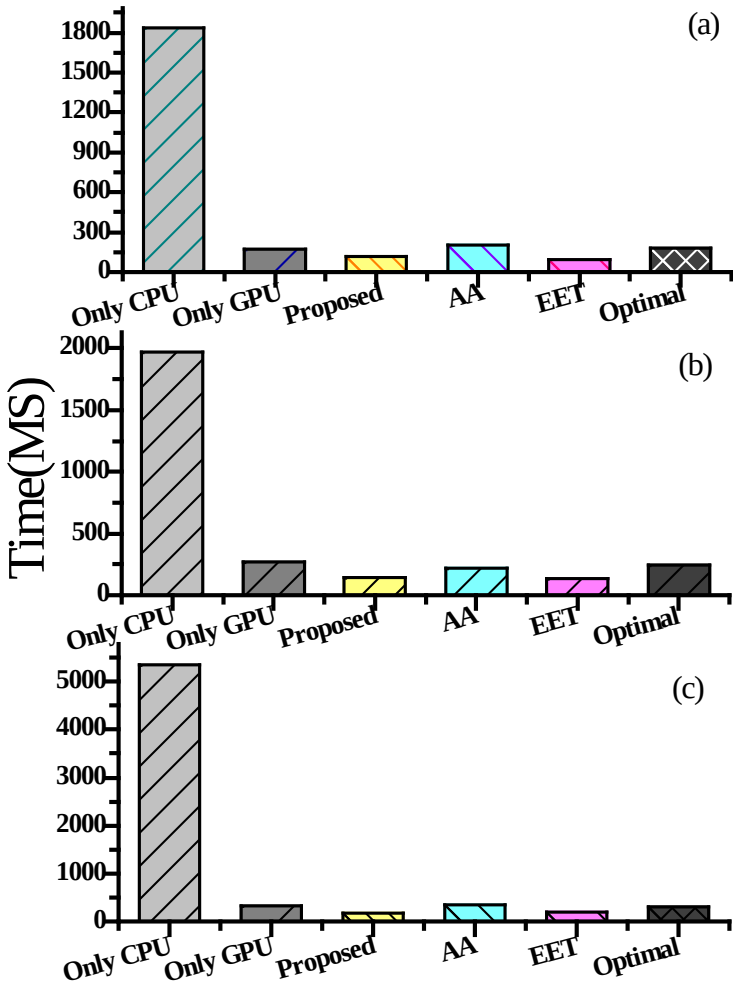


Fig. 7. Performance of each scheme for different programs

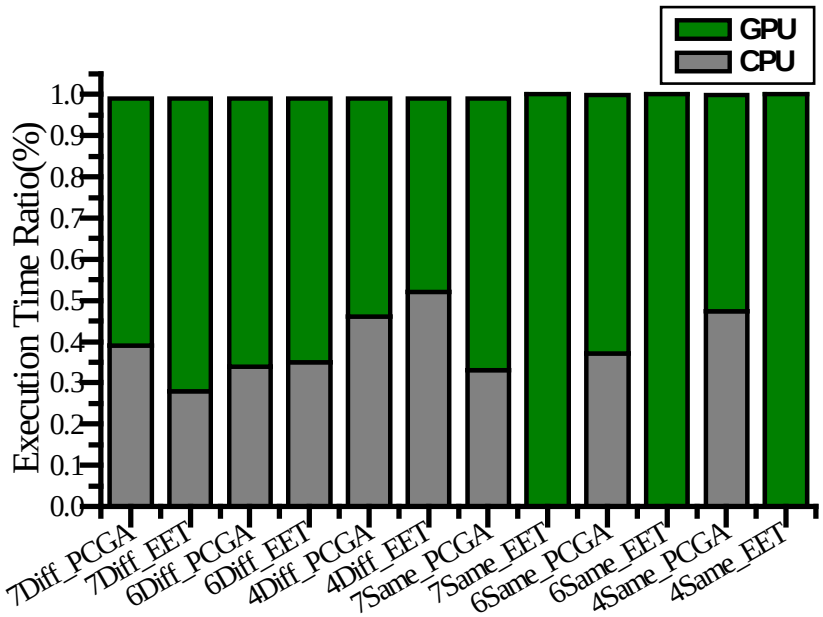


Fig. 8. Execution time ratio of CPU/GPU for the PCGA and EET scheme

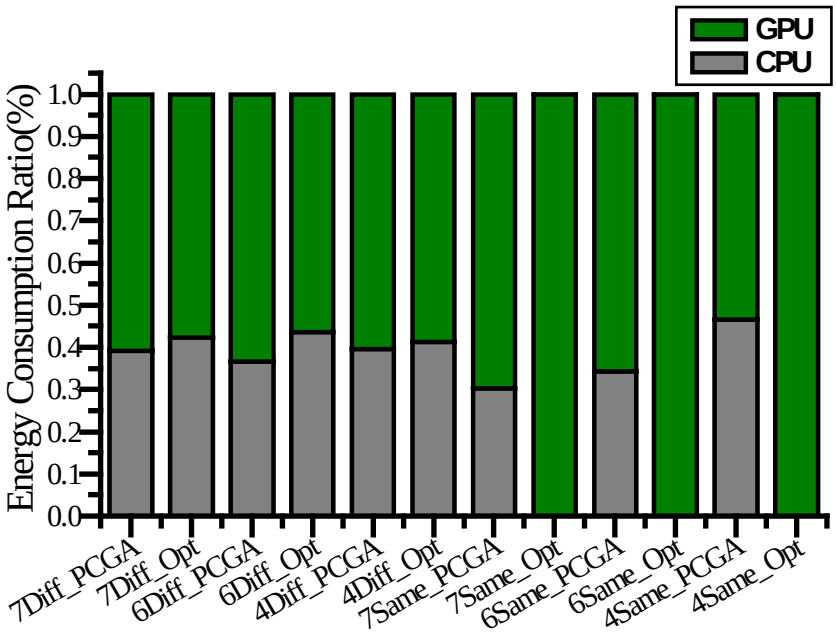


Fig. 9. Ratio of CPU/GPU energy consumption for the PCGA and Optiaml scheme

5. Conclusion

Energy saving will continue to be a key design goal for the heterogeneous computing system. In this paper, we focus the problem of saving the energy for heterogeneous with CPU and GPU by allocating the appropriate programs between them. In cope with the problem, we propose a framework to achieve this goal by considering the power, performance and the program scheduling. After getting power data, we formalize the energy saving problem as a

0-1 knapsack problem by considering the performance constraints. We present an experimental evaluation of our scheduling policy on typical platform by executing diverse set of benchmarks. The experimental results demonstrate that our approach is effective and soundness. On average, our proposed algorithm saves 14.97% energy compared with that of the time-oriented policy and yields 37.23% performance improvement than that of energy-oriented scheme.

References

[1] Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey, "Debunking the 100X GPU vs. CPU myth: An evaluation of throughput computing on CPU and GPU," *ACM SIGARCH Computer Architecture News*, 38(3), 2010. Article (CrossRef Link)

[2] ChrisGregg and Kim Hazelwood, "Where is the data?Why you cannot debate CPU vs.GPU performance without the answer," in *Proc. of the 2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'11)*, 134–144, 2011. Article (CrossRef Link)

[3] Sparsh Mittal and Jeffrey S. Vetter, "A survey of methods for analyzing and improving GPU energyefficiency," *ACM Computing Surveys*, 47(2), 2014. Article (CrossRef Link)

[4] Isaac Gelado, John E. Stone, Javier Cabezas, Sanjay Patel, Nacho Navarro, and Wen-mei W. Hwu, "An asymmetric distributed shared memory model for heterogeneous parallel systems," *ACM SIGARCH Computer Architecture News*, 38(1), 347–358, March 2010. Article (CrossRef Link)

[5] Qi Hu, Nail A. Gumerov, and Ramani Duraiswami, "Scalable fast multipole methods on distributed heterogeneous architectures," in *Proc. of the 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, New York, NY*, Article 36, pp. 1-12, 2011. Article (CrossRef Link)

[6] TOP500 List - November 2016, https://www.top500.org/list/2016/11/

[7] NOVEMBER 2016, https://www.top500.org/green500/lists/2016/11/

[8] Augonnet, Cédric, et al, "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, 23(2), 187-198, 2011. Article (CrossRef Link)

[9] Chi-Keung Luk, Sunpyo Hong, and Hyesoon Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *Proc. of the 42nd International Symposium on Microarchitecture (MICRO). ACM, New York, NY*, 45–55, 2009. Article (CrossRef Link)

[10] Belviranli, Mehmet E., Laxmi N. Bhuyan, and Rajiv Gupta, "A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(4), 57, 2013. Article (CrossRef Link)

[11] Grewe, Dominik, and Michael FP O'Boyle, "A static task partitioning approach for heterogeneous systems using OpenCL," in *Proc. of International Conference on Compiler Construction. Springer Berlin Heidelberg*, pp. 286-305, 2011. Article (CrossRef Link)

[12] Murilo Boratto, Pedro Alonso, Carla Ramiro, andMarcos Barreto, "Heterogeneous computational model for landform attributes representation on multicore and multi-GPU systems," *Procedia Computer Science*, 9, 47-56, 2012. Article (CrossRef Link)

[13] Bernabé, Gregorio, Javier Cuenca, and Domingo Giménez, "Optimization techniques for 3D-FWT on systems with manycore GPUs and multicore CPUs," *Procedia Computer Science*, 18, 319-328, 2013. Article (CrossRef Link)

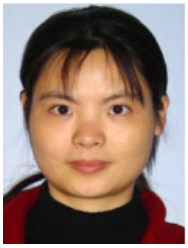
[14] Jiménez, Víctor J., et al, "Predictive runtime code scheduling for heterogeneous architectures," *International Conference on High-Performance Embedded Architectures and Compilers. Springer Berlin Heidelberg*, pp. 19-33, 2009. Article (CrossRef Link)

[15] Hong, Sunpyo, and Hyesoon Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," *ACM SIGARCH Computer Architecture News*, Vol. 37. No. 3, 2009. Article (CrossRef Link)
[16] Wang, Haifeng, and Yunpeng Cao, "Predicting power consumption of GPUs with fuzzy wavelet neural networks," *Parallel Computing*, 44, 18-36, 2015. Article (CrossRef Link)
[17] Junke Li, Bing Guo, et al, "A Modeling Approach for Energy Saving Based on GA-BP Neural Network," *Journal of Electrical Engineering and Technology*, 11(5), 1289-1298, 2016. Article (CrossRef Link)
[18] Paul, Indrani, et al, "Coordinated energy management in heterogeneous processors," *Scientific Programming*, 22(2), 93-108, 2014. Article (CrossRef Link)
[19] Guohui Wang, Yong Guan, Yi Wang, and Zili Shao, "Energy-Aware Assignment and Scheduling for Hybrid Main Memory in Embedded Systems," *Computing (Springer)*, Vol. 98, No. 3, pp. 279-301, 2016. Article (CrossRef Link)
[20] Yi Wang, Hui Liu, Duo Liu, Zhiwei Qin, Zili Shao, and Edwin H.-M. Sha, "Overhead-Aware Energy Optimization for Real-Time Streaming Applications on Multiprocessor System-on-Chip," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. 16, No 2, pp. 14:1-14:32, March 2011. Article (CrossRef Link)
[21] Wenjie Liu, et al, "A waterfall model to achieve energy efficient tasks mapping for large scale GPU clusters," in *Proc. of Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on. IEEE*, 2011. Article (CrossRef Link)
[22] Khemka, Bhavesh, et al, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *IEEE Transactions on Computers*, 64(8), 2394-2407, 2015. Article (CrossRef Link)
[23] Machovec, Dylan, et al, "Dynamic resource management for parallel tasks in an oversubscribed energy-constrained heterogeneous environment," in *Proc. of Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International. IEEE*, 2016. Article (CrossRef Link)
[24] Oxley, Mark A., et al, "Makespan and energy robust stochastic static resource allocation of a Bag-of-tasks to a heterogeneous computing system," *IEEE Transactions on Parallel and Distributed Systems*, 26(10), 2791-2805, 2015. Article (CrossRef Link)
[25] Qiang Liu, and Wayne Luk, "Heterogeneous systems for energy efficient scientific computing," in *Proc. of International Symposium on Applied Reconfigurable Computing. Springer Berlin Heidelberg*, 2012. Article (CrossRef Link)
[26] Kai Ma, et al, "Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures," in *Proc. of 2012 41st International Conference on Parallel Processing. IEEE*, 2012. Article (CrossRef Link)
[27] R. Kaleem, R. Barik, T. Shpeisman, B. Lewis, C. Hu, and K. Pingali, "Adaptive Heterogeneous Scheduling on Integrated GPUs," in *Proc. of the 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp.151-162, 2014. Article (CrossRef Link)
[28] Barik, Rajkishore, et al, "A black-box approach to energy-aware scheduling on integrated CPU-GPU systems," in *Proc. of the 2016 International Symposium on Code Generation and Optimization. ACM*, pp. 70-81, 2016. Article (CrossRef Link)
[29] Kai Ma, et al, "Energy conservation for GPU–CPU architectures with dynamic workload division and frequency scaling," *Sustainable Computing: Informatics and Systems*, 12, 21-33, 2016. Article (CrossRef Link)
[30] Totoni, Ehsan, Mert Dikmen, and María Jesús Garzarán, "Easy, fast, and energy-efficient object detection on heterogeneous on-chip architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4), 45, 2013. Article (CrossRef Link)
[31] Chandramohan, Kiran, and Michael FP O'Boyle, "Partitioning data-parallel programs for heterogeneous MPSoCs: time and energy design space exploration," *ACM SIGPLAN Notices*, Vol. 49. No. 5, 2014. Article (CrossRef Link)
[32] Guibin Wang, and Xiaoguang Ren, "Power-efficient work distribution method for CPU-GPU heterogeneous system," in *Proc. of International Symposium on Parallel and Distributed Processing with Applications. IEEE*, 2010. Article (CrossRef Link)

[33] Choi, Hong Jun, et al, "An efficient scheduling scheme using estimated execution time for heterogeneous computing systems," *The Journal of Supercomputing*, 65(2), 886-902, 2013. Article (CrossRef Link)
[34] Hamano, Tomoaki, Toshio Endo, and Satoshi Matsuoka, "Power-aware dynamic task scheduling for heterogeneous accelerated clusters," *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE*, 2009. Article (CrossRef Link)
[35] Mark Silberstein and Naoya Maruyama, "An exact algorithm for energy-efficient acceleration of task trees on CPU/GPU architectures," in *Proc. of the 4th Annual International Conference on Systems and Storage (SYSTOR'11). ACM, New York, NY*, Article 7, pp. 1-7, 2011. Article (CrossRef Link)
[36] Jang, Jae Young, et al, "Workload-aware optimal power allocation on single-chip heterogeneous processors," *IEEE Transactions on Parallel and Distributed Systems*, 27(6), 1838-1851, 2016. Article (CrossRef Link)

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 received his PHD degree in Computer Science from Sichuan University. He is currently an associate professor in the School of Computer and Information at Qiannan Normal University for Nationalities, China.

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 interest includes green computing.