Energy optimization of parallel programs in a heterogeneous system by combining processor core-shutdown and dynamic voltage scaling

Zhuowei Wang,[1] Hao Wang, [2,] * Wuqing Zhao, [3] Lianglun Cheng,[1]

[1] *School of Computers, Guangdong University of Technology, Guangzhou 510006, China*
[2]*Department of ICT and Natural Sciences, Norwegian University of Science and Technology, Norway*
[3]*Dingxin Information Technology Co.,Ltd, Guangzhou,510006, China*
**Corresponding author: hawa@ntnu.no*

Abstract: Reducing power consumption and improving efficiency are important aspects of the development of supercomputers into large-scale systems. As a result, heterogeneous systems have become an important development trend in high-performance computing. From the perspective of heterogeneous systems, this study establishes a model for energy optimization of parallel programs (EOPP) and puts forward a method of using it. By considering the energy overheads caused by re-synchronization, voltage switching, and operations in critical sections, the model effectively combines processor core-shutdown and dynamic voltage scaling technologies, which can be applied in a heterogeneous system to guide the optimization process. The results show that the proposed model can effectively reduce the energy consumption of parallel programs. Moreover, increasing the proportion of operations in the critical section enhances the optimal frequency of a processor while decreasing the probability of conflicts in the critical section. It can thus provide optimization space for reducing the frequency of a processor which ultimately reduces the energy overhead of the system.

Keywords: *Energy optimization; Parallel program; Heterogeneous system; Processor core-shutdown; Dynamic voltage scaling*

1. Introduction

Increasing growth in the demand for information is promoting a rapid development in high-performance computers. In order to enhance the performance of high-performance computer systems and solve large-scale computing problems, the development of heterogeneous systems which integrate general-purpose processors and accelerated processing units (APUs) has been one of the important trends in high-performance computing. Although heterogeneous systems show high peak computation speeds and efficiency, the problem of high power consumption still exists. High power consumption greatly challenges various aspects of large-scale heterogeneous and parallel computer systems (including reliability and heat dissipation) and causes significant consumption of energy. In this context, the power consumption problem is causing unprecedented concern [1].

Power optimization research encompasses a broad range of approaches ranging from hardware (bottom-level) to software (upper-level) [2–3]. Hardware approaches mainly include the use of low-power optimization technologies, e.g. using low-power logic devices [4-5], interconnected power optimization [6-7], control of leakage currents [8-9] and layout packaging [10-11]. Hardware optimization methods have reached a certain level of maturity and it is hard to further tweak them to satisfy the increasing demand for power optimization. Therefore, concern has shifted to power optimization at the software level [12–15].

Dynamic voltage scaling and processor core shutdown are two commonly used technologies to reduce the energy consumption. By reducing the voltage to reduce the energy consumption, Dynamic voltage scaling technology mainly uses the relationship between the dynamic power and the square of the voltage. The processor core shutdown technology is to reduce the energy consumption by closing the processor in the idle state. In recent

years, a large number of international research work focused on the low energy optimizations using the two technologies respectively. However, Combining the two technologies can get better energy optimization effect. At the same time, during the parallel program execution process, energy overheads caused by the re-synchronization, voltage switching, and operations in critical section will inevitably bring extra energy consumption.

Overall, the contributions made by this study can be summarized as follows:

(1) We establish an energy optimization of parallel programs (EOPP) model based on a heterogeneous system. During the modeling stage, all-round power modeling is conducted on the heterogeneous system from a program level perspective (fully considering the effect of energy overheads caused by re-synchronization, voltage switching, and operations in the critical section) to increase the accuracy of the power model.

(2) Using the EOPP model, we study a strategy that combines processor core-shutdown and DVS. Processor core-shutdown is used in the serial segment of the program — the other processor cores are shut down (i.e. all but the host processor core performing the serial program) in order to reduce energy consumption. On the other hand, the parallel segments of the program are processed using DVS technology. According to the loads on various processor cores, the voltage/frequency of each processor core is scaled to reach to an ideal voltage/frequency balance. By doing so, energy wasted by performing at maximum voltage/frequency can be avoided.

Section 2 presents a formalized description of the EOPP model in a heterogeneous system. Section 3 gives an analysis of the EOPP model in a heterogeneous system. On this basis, considering the influence of energy overheads caused by re-synchronization, voltage switching, and operations in the critical section, we propose a strategy for energy optimization that combines both processor core-shutdown and DVS technology. Section 4 presents the experimental results, and the last section summarizes the conclusions and future work.

2. Related work

A great deal of research has been carried out on technologies aimed at optimizing the power consumption of software running in a multi-core processor. Therein, processor core-shutdown [16–17] and dynamic voltage scaling (DVS) [18–19] are the two main technologies aimed at optimizing the power used by software.

From the perspective of complier optimization technology with multi-threading parallel and low power consumption, Zhao et al. [20] proposed a low-power optimization model integrating fine-grained multi-threading division and dynamic frequency adjustment based on two multi-thread system structures (Chip Multi-processing and Simultaneous Threading). Their model performs well in reducing the power consumption of a processor during operation as much as possible on the premise that instruction-level and thread-level parallelism is influenced little. Grochowski et al. [21] discussed the problem of trade-off between the throughput capacity of a microprocessor under speed and power constraints. These workers applied DVS, asymmetrical and sizeable processor cores, and inferential control technology to dynamically change the energy consumed during instruction execution under parallelization instruction of software. Their results indicate that the optimal choice is to comprehensively utilize DVS with asymmetrical processor cores.

Kadayis et al. [22] proposed a method capable of shutting down idle processor cores to reduce the energy consumption of nested loops. However, their energy optimization method does not utilize DVS technology. In subsequent research [23], they proposed employing DVS to decrease the voltages of processor cores with lightly-loaded threads. In this way, the degree of load equilibrium of the programs can be adjusted to further save power and reduce energy consumption. However, their research only made a comparison between DVS and processor core-shutdown and failed to favorably combine the two low-power technologies.

Li et al. [24–25] have advised that a two-dimensional space defined for parallel programs in a multi-core structure should be optimized (one dimension corresponding to change in number of active processors; the other to conducting DVS on each processor core). However, a parallel program tends to contain both parallel and serial segments and there are certain time and energy overheads associated with conducting DVS and processor core-shutdown. When a program progresses from a serial segment to a parallel segment, it is assumed that closing idle processor cores does not waste extra time and energy. However, when the operations in a serial segment complete and enter into the next parallel segment, it is necessary to re-activate the processor cores in the dormant state and then make the processor core recover to the normal work state. In this context, there is a re-synchronization overhead with respect to time and energy. Moreover, the voltage switching overhead in time and energy always exists during each DVS process. Thus, the associated power consumption cannot be ignored when frequently conducting DVS operations. Additionally, data must inevitably be shared during parallelization of programs. Therefore, in order to maintain data consistency, shared data need to be operated on within a critical section. Thus, the synchronization operation on the critical section has an effect on power optimization.

This study establishes a model for energy optimization of parallel programs. By considering the energy overheads caused by re-synchronization, voltage switching, and operations in critical sections, the model effectively combines dynamic voltage scaling and processor core-shutdown technologies, which can effectively reduce the energy consumption of parallel programs.

3. Formal description of the optimization problem

3.1 Architecture of a heterogeneous system

Before giving a description of the problem, the architecture of a heterogeneous system is first abstracted to establish a basis for the energy model for parallel programs. Fig. 1 shows the typical architecture of a heterogeneous parallel system. Such a system contains multiple computing resources: central processing units (CPUs), graphics processing units (GPUs), medium interface connectors (MICs), and field programmable gate arrays (FPGAs).

In general, those parts used to accelerate processing (the APUs) are only charged with performing specific computing tasks and are not equipped with accomplished task management and

scheduling mechanisms. Therefore, most of the APUs need to be executed under the control of a general-purpose microprocessor (host processor). The host processor is generally connected to the APU through an external bus and they have their own (off-chip) dynamic random-access memory (DRAM). In this way, the two processors can realize data communication in the form of direct memory access (DMA).

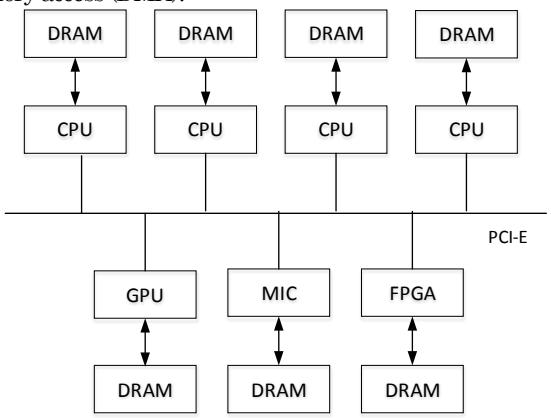

Fig.1. The typical architecture of a heterogeneous parallel system.

3.2 Problem description and conditional hypotheses

The problem can be described as follows. Consider a heterogeneous system consisting of a host processor and an APU and a parallel program containing N_C serial and N_G parallel segments. It is hypothesized that the architecture of the heterogeneous system contains P_C host processor cores and P_G APU cores, and that the voltage and frequency allocated to the mth processor at the jth stage (serial or parallel) are expressed in the form $(V_{j,m}, f_{j,m})$. We are required to find the optimal voltage/frequency of each processor core that reduces the total energy consumed by the whole parallel program to the largest extent without incurring performance loss.

In order to accurately define the energy optimization problem for parallel programs executing on a heterogeneous system, some further hypotheses are made about the program, system architecture, and circuit realization:

(1) The logistical behavior of the program does not change when the frequency changes.

(2) The frequency of a processor is continuous and adjustable.

(3) The serial and parallel parts of the parallel program are separately fulfilled by the host processor and APU. The time and energy overheads resulting from data transmission between the host processor and APU can be ignored.

(4) The program flows from parallel to serial segments and there are no extra time and energy overheads caused by closing idle processors.

(5) The program operates from serial to parallel segments taking into account the re-synchronization overheads in time and energy.

(6) The time and energy overheads caused by converting the frequencies of the processors are considered.

(7) The probability of a critical section occurring within a thread conforms to a uniform distribution while that in different thread levels is completely independent. The number of threads with competitive critical sections within unit time is in accordance with the binomial distribution.

It can be considered that the assumption 1 and 2 are true and reasonable. Processor core shut-down, as well as the data transmission between the master processor and the accelerated processor will impose additional time and energy consumption. However, it is important to point out that this article focuses on the energy consumption of re-synchronization, voltage switching, and operation in critical sections, the time and energy consumption overhead caused by processor core shutdown are not the main component of the execution time and total system energy consumption, so it can be considered that the assumptions 3, 4, 5, 6 are reasonable. In addition, the energy consumption analysis of operations in the critical sections is based on the literature [26], therefore the assumption 7 is also reasonable.

3.3 Parameters used in the EOPP model for the heterogeneous system

The following parameters are defined for the problem in the heterogeneous system:

N_C: the number of serial segments;

N_G: the number of parallel segments;

P_C: the number of host processor cores;

P_G: the number of APU cores;

tc_i: the execution time at the ith serial segment;

IC_i^C: the instruction cycles of the ith serial segment;

t_s: time of each synchronization overhead;

$f_{C_{max}}$: the maximum clock frequency of a host processor core in an active state;

$f_{G_{max}}$: the maximum clock frequency of an APU core in an active state;

$V_{C_{max}}$: the maximum voltage of a host processor core;

$V_{G_{max}}$: the maximum voltage of an APU core;

$P_{dyn}(f)$: the dynamic power consumption when the frequency of the processor is f;

P_{stat}: the static power consumption of a processor core in the shutdown state;

E_{syn}: the energy overhead of each re-synchronization step;

$tG_{j,m}$: the execution time of the mth processor at the jth serial segment;

tG_j: the longest execution time of P processors at the jth parallel segment;

$IC_{j,m}^G$:: (profiling data) the instruction cycles of the mth processor at the jth parallel segment;

E_C: the energy consumption of a serial segment;

E_G: the energy consumption of parallel segment;

E_{total}: the total energy consumption before optimization;

E_{opt}: the total energy consumption after optimization;

$t_{swi}(V_{j-1,m}, V_{j,m})$: the time overhead due to switching voltage from $V_{j-1,m}$ to $V_{j,m}$;

$f_{j,m}$: the optimal clock frequency of the mth processor at the jth parallel segment;

$V_{j,m}$: the optimal voltage of the mth processor at the jth parallel segment;

$E_{swi}(V_{j-1,m}, V_{j,m})$: the energy overhead when switching the voltage from $V_{j-1,m}$ to $V_{j,m}$.

4. Analysis of the EOPP model for a heterogeneous system

4.1 The total energy consumed by parallel programs that are not optimized

Complementary metal-oxide semiconductor (CMOS) transistors are the basic devices that make up a computer. The power consumption of a CMOS is mainly comprised of dynamic and static components and the dynamic component P_{dyn} arises due to the changes in state of the CMOS as it performs work. Thus, we can write:

$$P_{dyn} = \alpha C V^2 f, \tag{1}$$

where α, and refer to is the switching activity factor (in the range of 0–1), C the switching capacitance, V the supply voltage, and f the clock frequency. The static component arises due to leakage currents in the idle state and can be estimated according to a certain proportion.

According to basic physics, the energy consumed (E) can be found by integrating power consumption level with respect to time. A simple calculation can therefore be made by multiplying the average power consumption (P) by time (t):

$$E = Pt. \tag{2}$$

The parallel program is assumed to consist of N_C serial segments and N_G parallel segments. Therefore, the total energy consumed by running the parallel program can be expressed as the sum of the energies consumed by the serial and parallel segments:

$$E_{total} = E_C + E_C. \tag{3}$$

If DVS is not conducted, all the processor cores (including the host processor and APU cores) operate at maximum voltage/frequency. Therefore, according to the hypotheses in Section 2.2 (the serial and parallel parts of the parallel program are separately completed by the host processor and APU, and the time and energy overheads caused by data transmission between the host processor and APU are ignored), the total energy consumed by all the processor cores due to the N_C serial and N_G parallel segments is given by:

$$E_{total} = \sum_{i=1}^{N_C} (P_C \cdot tc_i \cdot \alpha C {V_{C_{max}}}^2 f_{C_{max}}) + \left(\sum_{j=1}^{N_G} (P_G \cdot tG_j \cdot \alpha C {V_{G_{max}}}^2 f_{G_{max}}\right). \tag{4}$$

4.2 Combining DVS and processor core shutdown

The parallel parts of the program are to be subjected to energy optimization. On the one hand, $P_C - 1$ host processors (which are not utilized in the execution of the serial segment) are shut down. On the other hand, it is hypothesized that the optimal voltage/frequency of the mth APU core during the execution of the jth parallel segment is $(V_{j,m}, f_{j,m})$. Therefore, failing to consider the time and energy overheads caused by state switching, the ideal total energy consumed by the parallel program can be expressed as:

$$E_{opt} = \sum_{i=1}^{N_C} \left(tc_i \cdot \alpha C {V_{C_{max}}}^2 f_{C_{max}} + (P_C - 1) \cdot tc_i \cdot P_{stat}\right) + \sum_{j=1}^{N_G} \sum_{m=1}^{P_G} \left(tG_j \cdot \alpha C {V_{j,m}}^2 f_{j,m}\right). \tag{5}$$

It is further hypothesized that the ratio of the power consumed by a processor core in the inactive state to that in the active state is ε. When $P_C - 1$ processor cores are shut down during the execution of the serial segment, the power consumed by the processor cores in the shutdown state is given by:

$$P_{stat} = \varepsilon \cdot \alpha C {V_{C_{max}}}^2 f_{C_{max}}. \tag{6}$$

For the sake of convenience, the frequency and voltage of the processor cores can be standardized. For a parallel segment of the program, it is hypothesized that the frequency of the APU core due to DVS is $f_{j,m}$ and the maximum allowable frequency is f_{max}. The standardized frequency is now given by

$$f_{j,m}{}' = \frac{f_{j,m}}{f_{max}}. \quad (m = 1, \cdots P_G, j = 1, \cdots, N_G). \tag{7}$$

In a similar way, it can be hypothesized that the voltage of a processor core due to DVS is $V_{j,m}$ and the maximum allowable voltage is V_{max} so that a standardized voltage can be defined as

$$V_{j,m}{}' = \frac{V_{j,m}}{V_{max}} \ (m = 1, \cdots P_G, j = 1, \cdots, N_G.). \tag{8}$$

According to the relationship between supply voltage and clock frequency, it can be further seen that:

$$V'' = k_1 + k_2 \cdot f'. \tag{9}$$

Eq. (7) conforms to industrial standards, according to the literature [27]. By analyzing the technical indices available, it can be speculated that k_1 and k_2 should be offset to make them 0.3 and 0.7, respectively.

For the parallel segments of the program, ignoring the synchronous waiting and intercommunication between parallel parts, the voltage/frequency of each processor core can be scaled to different levels according to the different execution times. The longest execution time tG_j of the processor core can be standardized to 1 so that those of the other $P_G - 1$ processor cores all satisfy $tG_{j,m} \le tG_j$. According to the various different execution times, $tG_{j,m}$, the frequency of the mth coprocessor core is therefore scaled by $f_{j,m}{}'$. By doing so, the following formula can be obtained. where:

$$f_{j,m}{}' = \frac{tG_{j,m}}{tG_j}. \tag{10}$$

The two parameters $tG_{j,m}$ and tG_j in Eq. (10) can be measured using a performance test tool (data profiling) and, based on Eq.(9), the following formula can be acquired:

$$V_{j,m}{}' = k_1 + k_2 \cdot f_{j,m}{}'. \tag{11}$$

4.3 Re-synchronization and voltage switching overheads

Each DVS operation can cause additional time and energy overheads due to voltage switching. Therefore, the total energy consumed by the parallel program needs to have the overheads caused by re-synchronization and voltage switching to be separately taken into account.

(1) Energy overhead caused by re-synchronization

During the execution of the parallel program, it is necessary to re-activate the processor cores in a dormant state to put them back into the working state. This occurs when the execution process in a serial segment has completed and we return to the next parallel segment. In this context, the whole system exhibits re-synchronization overheads in terms of time and energy. Time overheads induced by re-synchronization can be avoided, however, by using a pre-activation strategy [28]. Pre-activation

refers to when a processor core in a shutdown state is activated in advance before reutilization is required so that the processor core is in an active state when it is needed. This can obviously avoid any time overhead. The ideal pre-activation strategy is shown in Fig.2.

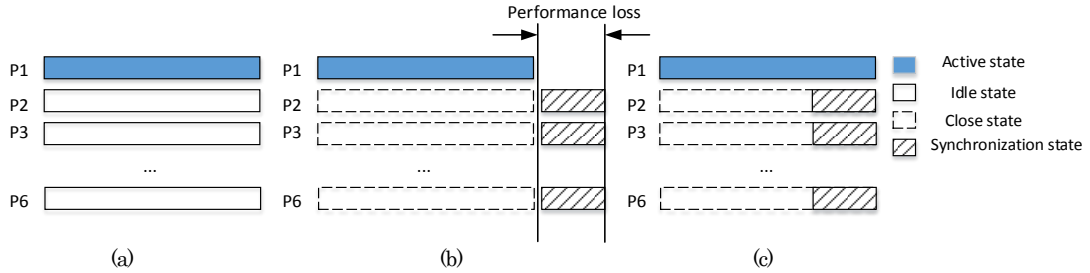

Fig.2. The ideal pre-activation strategy showing: (a) the initial state, (b) re-synchronization overhead caused by processor core-shutdown, and (c) pre-activation to eliminate the synchronization overhead

The pre-activation strategy can remove the time delay but fails to avoid an energy overhead caused by re-synchronization. One purpose of this study is to take re-synchronization overheads into account when calculating the total energy overhead of the parallel program. To that end, the total energy overhead is written in the form:

$$E_{opt}{}^{1} = \sum_{i=1}^{N_C}\left(tc_i \cdot \alpha C V_{C_{max}}{}^{2} f_{C_{max}} + (P_C - 1)((tc_i - t_s) \cdot P_{stat} + E_{syn}\right) + \sum_{j=1}^{N_G}\sum_{m=1}^{P_G}\left(tG_j \cdot \alpha C V_{j,m}{}^{2} f_{j,m}{}'\right) \quad (12)$$

(2) Energy overhead caused by voltage switching

During the execution of the parallel program, each DVS step can incur a voltage switching overhead in time and energy. In this case, the time overhead can once again be avoided by, this time, using a method of pre-switching. In the CPU–GPU heterogeneous system, the parallel parts of the parallel program are completed using APUs and, therefore, DVS is generally aimed at processes going on in the APU cores. According to the various loads on each APU core, the voltages/frequencies are rescaled to ensure the different parallel parts are completed in the same amount of time, thus eliminating any null cycles.

Fig.3 shows the strategy used to avoid time overheads using DVS. Fig.3(a) shows the load conditions when each processor core operates at its maximum voltage/frequency and DVS running consumes certain time. If the optimal voltage/frequency is calculated based only on the different loads, a time delay is accumulated once the paralleled intervals are increased which causes performance loss, as shown in Fig.3(b). In order to avoid a time delay, the time overhead in this part is subtracted when calculating the voltage/frequency of each processor core, as shown in Fig.3(c).

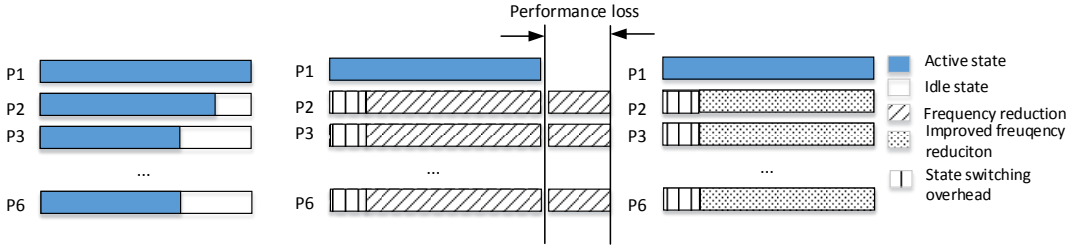

Fig.3. Diagram to illustrate DVS overheads: (a) without conducting DVS, (b) carrying out DVS after considering the voltage switching overhead, and (c) carrying out DVS to avoid performance loss.

As a result of rescaling, the frequency of the APU core can be expressed in the form:

$$f_{j,m}{}'' = \frac{tG_{j,m}}{tG_j - t_{swi}(V_{j-1,m} - V_{j,m})} \cdot f_{G_{max}} \quad (13)$$

It has been shown that the time and energy overheads caused by voltage switching can be expressed via the relationships [29]:

$$\boldsymbol{t_{swi}(V_{j-1,m}, V_{j,m}) = \frac{2C}{I_{max}}\left|V_{j-1,m} - V_{j,m}\right|,\ j = 2, \cdots, N_G} \quad (14)$$

$$E_{swi}(V_{j-1,m}, V_{j,m}) = (1-u) \cdot C \cdot \left|V_{j-1,m}{}^{2} - V_{j,m}{}^{2}\right|, \quad (15)$$

where C, and refer to is the capacitance, u the energy effectiveness factor of the voltage regulator, and I_{max} the maximum allowable current. Substituting Eq. (14) into Eq. (13) allows us to acquire the optimal frequencies to which each processor core needs to be scaled to using Eqs.(7)–(9): $f_{j,m}{}'' = \frac{tG_{j,m} \cdot f_{G_{max}}}{tG_j - \frac{2C \cdot k_2 \cdot V_{G_{max}}}{I_{max} \cdot f_{G_{max}}}\left|f_{j-1,m} - f_{j,m}\right|}$.

Considering the re-synchronization overhead part of the total energy overhead of the parallel program, the total energy overhead can be calculated using:

$$E_{opt}{}^{2} = \sum_{i=1}^{N_C}\left(tc_i \cdot \alpha C V_{C_{max}}{}^{2} f_{C_{max}} + (P_C - 1)((tc_i - t_s) \cdot P_{stat} + E_{syn}\right) + \sum_{j=1}^{N_G}\sum_{m=1}^{P_G}\left(tG_j \cdot \alpha C V_{j,m}{}^{2} f_{j,m}{}''\right) + \sum_{j=1}^{N_G-1}\sum_{m=1}^{P_G}(1-u) \cdot C \cdot \left|V_{j,m}{}^{2} - V_{j+1,m}{}^{2}\right|. \quad (16)$$

(3) Energy overheads due to operations in critical sections

In the process of parallelizing programs, the need to share data between parallel segments becomes inevitable. In order to guarantee the consistency of the data, operations involving shared data need to be performed in a critical section. The effect

of using critical sections on the power consumption of the program must therefore be accounted for.

It is hypothesized that the parallel parts need to undergo parallel processing using jth class processor in which the task load, operating proportion, and probability of conflict in the critical section are S_j, σ, and c, respectively. The number of the jth class processor r_j is recorded as N_j and $\sum_{r_j \in R} N_j$.

The work in this section is based on certain hypotheses appearing in the literature [30]. In particular, it is hypothesized that the probability that a critical section occurs within a thread conforms to a uniform distribution and that the probabilities for different threads are completely independent. The number of threads with competitive critical sections per unit time follows a binomial distribution. According to conflict models established in the literature [24], the time a processor takes to execute a critical section is composed of the time taken for execution of the local critical section and the waiting time in the critical section. The waiting time is proportional to the execution time, execution probability, and conflict probabilities of the critical sections of the other processors. Therefore, the execution time in the critical section of the jth class processor can be expressed as:

$$t_{j,m} = \frac{\sigma S_j}{f_{j,m} v_j N_j} + \left((N_j - 1) \frac{\sigma S_j}{f_{j,m} v_j N_j} + \sum_{r_k \in R - \{r_j\}} N_k \frac{\sigma S_k}{f_{k,m} v_k N_k} \right) \sigma c. \quad (17)$$

The second item in Eq.(17) represents the average waiting time of the jth class processor. Considering that the contribution made by $\sigma^2 S_j c / f_{j,m} v_j N_j$ to $tG_{j,m}$ is small, the execution time of the critical sections of the jth class processor can be approximated as:

$$t_{j,m} = \frac{\sigma S_j}{f_{j,m} v_j N_j} + \sum_{r_k \in R} \frac{\sigma^2 S_k c}{f_{k,m} v_k}. \quad (18)$$

When we include the overheads in the critical section, the execution time of the jth class processor is therefore given by the expression:

$$tG_{j,m} = \frac{(1-\sigma) S_j}{f_{j,m} v_j N_j} + \frac{\sigma S_j}{f_{j,m} v_j N_j} + \sum_{r_k \in R} \frac{\sigma^2 S_k c}{f_{k,m} v_k} = \frac{S_j}{f_{j,m} v_j N_j} + \sum_{r_k \in R} \frac{S_k}{f_{k,m} v_k} \sigma^2 c. \quad (19)$$

Now, $\forall r_j,\ r_k \in R,\ tG_{j,m} = tG_{k,m}$. Therefore, $S_j / S_k =$; Owing to $f_{j,m} v_j N_j / f_{k,m} v_k N_k$. As $\sum_{r_k \in R} S_j = S$, it can be seen that $S_j =$ Thus $f_{j,m} v_j N_j / \sum_{r_k \in R} f_{k,m} v_k N_k$, so that,

$$tG_{j,m} = \frac{S_j}{f_{j,m} v_j N_j} + \sum_{r_k \in R} \frac{S_k}{f_{k,m} v_k} \sigma^2 c = \frac{1 + \sigma^2 cn}{\sum_{r_k \in R} f_{k,m} v_k N_k} S. \quad (20)$$

Therefore, according to Eq. (10), it can be seen that:

$$f_{j,m}''' = f_{j,m}(1 + \sigma^2 cn). \quad (21)$$

Based on the foregoing analysis, the following conclusion can be drawn. For a given program, the operations in the critical section cause a fractional increase in the operating frequency of a processor of $\sigma^2 cn$ (compared to the situation in which critical sections are not used). This further increases the total energy overhead of the program.

5. Experiments and analysis

5.1 Experimental platform

A heterogeneous system consisting of an Intel Core i7 920 quad-core CPU and AMD 4870 GPU was used to form an experimental platform for testing purposes. In the system, the CPU and GPU have their own separate independent memory spaces and are connected via the PCI-E bus to realize data communication.

The theoretical analysis of the energy consumption model proposed in this study suffers from certain limitations. More precisely, it fails to accurately describe some of the uncertain behavior of the processors during the execution process and to simulate the energy consumed in each part. In practical CPU–GPU heterogeneous systems, the GPU does not provide perfect support for dynamic scaling of the voltage/frequency (i.e. a few frequency tuning ranges are used). This is disadvantageous when it comes to conducting theoretical research and verifying the behavior of the EOPP model under the combined effect of DVS and processor core-shutdown. Therefore, we employ a GPU power simulator, GPGPUSim, to facilitate experimental verification.

GPGPUSim, however, fails to operate in a GPU environment and therefore we made modifications based on GPGPUSim for this study. A simple simulator was established in the application layer (CPU end of CUDA program) to simulate the simultaneous execution of multi-GPUs by practice driving. In addition, GPGPUSim configures the CPU by reading configuration files during operation. Therefore, the multi-GPU environment of the heterogeneous system can be simulated by dynamically modifying the configuration files of the program.

The GPU power simulator used in this study was developed using the Wattch power model (based on GPGPUSim). To favorably realize scaleable dynamic voltages, the following modifications were made to the simulator:

(1) The frequency of each processor core is independently scaled by changing the latency of all the functional parts in the processor cores to scale the clock frequency of each core.

(2) The voltage is updated according to the scaled clock frequency.

(3) The power consumption of each functional part in the Wattch model is updated based on the new clock frequency and voltage.

(4) When the voltage state switches, the voltage switching overhead in time and energy are calculated according to Eqs. (14) and (15).

Table 1 Parameters of the processors used in the test platform

	Intel Core i7 920 CPU	AMD 4870 GPU-H/GPU-L

Processor frequency (GHz)	2.67, 2.4, 2.0, 1.6	0.75, 0.65, 0.55
Memory frequency (GHz)	1.33 (DDR3)	0.9/0.7/0.5 (GDDR5)
Cache	L1: I 32 KB, D 32 KB; L2: 256 KB; L3: 8 MB	–
Memory (GB)	8	1

5.2 Test cases

A great deal of attention has been paid in the scientific computing field to multiplying a vector by a sparse matrix using parallel computing. For this study, we chose four sparse matrices from the Harwell11 sparse matrix set [31-32] and used them as test cases to verify the results from the EOPP model. Table 2 displays details of the sparse matrices chosen.

Table 2. Details of the test cases used.

Application name	Scale	Description
Matrix 1	1030	21*21*5 irregular grid
Matrix 2	886	21*21*5 irregular grid
Matrix 3	1080	35*11*13 grid
Matrix 4	5005	16*23*3 grid

5.3 Compiler implementation

To unify the programming models of the CPU and GPU and simplify the migration of existed applications to CPU-GPU heterogeneous systems, we have extended four GPU oriented instruction commands based on the OpenMP language [33]. The commands are used to instruct the compiler to convert the OpenMP code executed on the original CPU to Brook+ code executed on the GPU. At the same time, a source-to-source compiler called MPtoStream is designed and implemented based on the GCC compiler. It can complete the aforementioned code conversion process. Fig.4 shows the MPtoStream compiler framework. Firstly, the expanded OpenMP program is analysed by the Parser module, and the intermediate syntax tree is generated. Then, the OpenMP program is converted through a series of syntax trees to generate the intermediate syntax tree structure for the Brook+ language. Finally, the reverse output process is called to generate the OpenMP codes which are executed on CPU processor and Brook+ codes which are executed (separately) on a GPU processor. Based on this MPtoStream framework, we expand the parallel loop scheduling module of sparse matrix vector multiplication (SpMV) for CPU-GPU heterogeneous system, as shown in the grey boxes in Fig. 4.

5.4 Code example

Fig. 5 shows that an example of parallel loop scheduling code. This program implements a sparse matrix vector multiplication algorithm (SPMV). The outermost layer loop index variable *i* represents the line number of the matrix, and the inlayer loop is calculated only for non-zero elements. *N(i)* indicates the number of non-zero elements in line *i*. It can be seen that the outermost layer loop is composed of TotalLine iterations. In this paper, the whole procedure is divided into one serial segment and two consecutive parallel program segments. The serial segment was mainly used to execute initialisation of the program (processed by the CPU). The parallel program segment was divided the sparse matrix into even two parts according to the row number, which were separately executed using two segments (processed by the GPU).

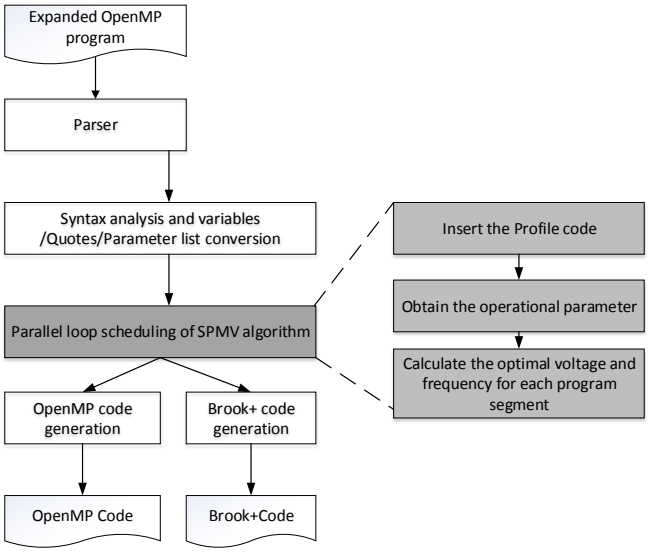

Fig. 4 MPtoStream compilation framework

The first loop iteration algebra on GPU is Ngpu. Therefore, the iterative subset (1,Ngpu), (Ngpu+1,TotalLine) is allocated to the GPU as the two continuous parallel program segments, and the rest of the programs are allocated to the GPU. As shown in Fig. 5, in the left upper dashed box, the expanded compile command identity maps the parallel segment to GPU for execution. The right-hand dashed box is the CPU code that has been converted. Firstly, the GPU data flow space is declared, and the data loading process is completed. Then the iteration space which mapped on the GPU is formulated (through the spmv_kernel program implementation). Finally, the computing process is executed on

the GPU. The kernel calculation process is completed on the GPU, and the output results are stored in the CPU storage space. This procedure accomplishes the parallel loop partitioning process of CPU-GPU heterogeneous parallel systems.

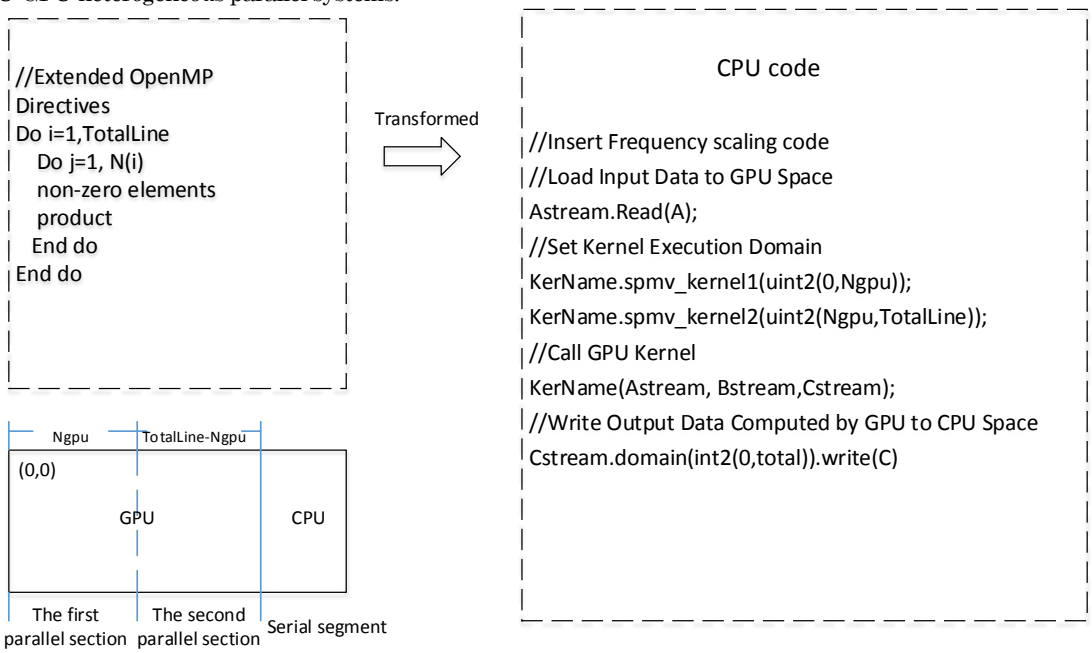

Fig. 5CPU-GPU SpMV parallel loop scheduling code example

5.5 Experimental analysis

(1) Analysis of the EOPP simulation results

The analysis process can be divided into three stages: profiling, voltage/frequency scaling, and optimization. In the profiling stage, the parameters needed to allow the subsequent calculations (of the optimal voltages/frequencies) to be executed are acquired for each program segment according to the formulae derived for the model. During the voltage/frequency scaling stage, compiler directive commands are inserted into the source code at the appropriate locations to realize voltage scaling. In the optimization stage, during transition from a parallel stage to the next parallel node, the same voltage is allocated to the same processor core, as far as possible, in order to avoid unnecessary overheads. For example, if the frequency of processor k is $f_{j,k}$ during the jth parallel segment, then the frequency of the processor also takes the value $f_{j,k}$ when we transit to the next adjacent parallel segment. In this context, the processor k is firstly selected.

The test cases were simulated using the GPGPUSim simulator to acquire the relevant parameters needed for the analysis and, finally, to obtain the resulting amounts of energy saved. In each of the four test cases, the parallel program was divided into a serial program segment and two continuous parallel segments. The serial segment was mainly used to execute initialization of the program (processed by the CPU). Each parallel program segment was composed of two parts, which were separately executed using two parallel segments. Table 3 shows the profile data measured for the serial and parallel segments.

Table 3. Profile data showing the number of execution cycles in each serial and parallel segment.

Test case	Execution cycles in each serial program	Execution cycles in the first/second parallel segment					
		P1	P2	P3	P4	P5	P6

1	75620	531381/ 521680	531743/ 531916	540119/ 540320	531736/ 531993	531295/ 531531	539931 /539572
2	90160	393208/ 393354	393557/ 393492	400844/ 400983	393620/ 393688	393480/ 393688	400383/ 401101
3	72590	580267/ 583660	588839/ 592460	580181/ 583581	588839/ 592078	580181/ 583140	588839/ 591924
4	72600	12502099/ 12546222	12543121/ 12547473	12544017/ 12508724	12544815/ 12548586	12547937/ 12547808	12511428/ 12549846

Table 4. The optimal scaled voltages and frequencies in each parallel segment.

Test case	Parallel segment	Optimal voltage/frequency (V/MHz).					
		P1	P2	P3	P4	P5	P6
1	1	2.0268/ 753.82	2.0277/ 754.49	2.0500/ 770	2.0277/ 754.48	2.0495/ 769.65	2.0270/ 753.66
	2	2.02271/ 754.02	2.0277/ 754.45	2.0500/ 770	2.0279/ 754.59	2.0267/ 753.74	2.0481/ 768.63
2	1	2.0227/ 750.95	2.0239/ 751.82	2.0500/ 770	2.0240/ 751.98	2.0236/ 751.63	2.0481/ 768.85
	2	2.0223/ 750.70	2.0229/ 751.08	2.0496/ 769.73	2.0235/ 751.35	2.0234/ 751.53	2.0501/ 770
3	1	2.0290/ 755.33	2.0498/ 769.89	2.0287/ 755.19	2.0498/ 769.89	2.0287/ 755.19	2.0498/ 769.88
	2	2.0288/ 755.24	2.0500/ 770	2.0286/ 755.11	2.0492/ 769.40	2.0276/ 754.40	2.0489/ 769.10
4	1	2.0466/ 769.26	2.0493/ 769.45	2.0492/ 769.78	2.0495/ 769.34	2.0499/ 769.90	2.0457/ 767.23
	2	2.0496/ 769.53	2.0567/ 769.65	2.0451/ 766.55	2.0495/ 769.82	2.0496/ 769.72	2.0497/ 769.86

The voltage/frequency scaling results were then obtained according to the method given in the analysis for setting the optimal voltage (Table 4). These values were then used together with the energy calculation formulae, Eqs. (4) and (16), to calculate the energy used before and after the optimization process. Thus, the resulting energy saving can be obtained taking into account the energy overheads caused by re-synchronization, voltage switching, and use of critical sections.

Fig. 6 shows the energies consumed in the serial and parallel segments, as well as the total energy consumed, in the four test cases (before and after optimization). The energy savings are also shown (as percentages). It can be seen from the results that the total energy saving was as large as 10.5%. Also, the energy saved in the serial segments was significantly larger than that saved in the parallel segments. The main reason for this is that the processor cores that were not being utilized were shut down when the programs entered the serial segment which greatly reduces energy consumption. Moreover, the results obtained can be related to the degree of parallelization and parallel programming loads in the test cases.

(2) Analysis of the EOPP real results

The theoretical analysis of the energy consumption model proposed in this study suffers from certain limitations. More precisely, it fails to describe some of the uncertain behaviours of the processors during the execution process and to simulate the energy consumed in each part. In this section, the energy optimization methods of parallel programs are implemented under the real heterogeneous system platform, so, we can obtain more realistic energy optimization results.

The test platform in this study is a system composed of an Intel Core I7 920 Quad-Core CPU and two AMD 4870 GPUs. To examine the efficiency of the algorithms proposed on this system, the frequency of the storage of one GPU kernel is adjusted from 900 MHz to 700 MHz so as to obtain two different GPU kernels with different performances. Thereinto, the kernel with the higher performance is recorded as GPU-H, while that with the lower performance is designated GPU-L. The specific parameters for this test platform are listed in Table 1.

The voltage/frequency information supported by the processor is obtained. The current mainstream CPU processor supports DVS technology, such as Intel's SpeedStep [34], AMD's PowerPlay [35] and other power management technologies. In

addition, the power management modules for the specific processor are integrated in the operating system (OS). The OS in this study is OpenSUSE v10.3. The operating frequency information supported by CPU can be obtained through the ACPI [36] interface provided by the system, and the operating frequency can be dynamically adjusted. The controlled power method of the GPU is also gradually improved. AMD provides an ADL library (AMD Display Library)[37], which can dynamically access and modify the operating voltage and frequency of GPU through the ADL_Overdrive5_ODPerformanceLevels_Get and ADL_Overdrive5_ODPerfromanceLevels_Set interfaces.

The power consumption of the system is measured by the external HIOKI 3344 power tester, and the power consumption is read through the RS-232 serial port mechanism. The measurement error of the instrument is within the ± 1 digit range of the measured value.

It is difficult to measure the energy consumption of the serial segment and parallel segment separately in real heterogeneous systems. Therefore, this article only compares the total energy consumption of the system before and after the optimization; and presents the statistics pertaining to the total energy consumption savings, as shown in Fig.7. During the execution of parallel programs, the voltage and frequency of the CPU and GPU are set to the maximum value (2.0 V/2.67 GHz; 2.0 V/750 MHz) in the initial case. After entering the parallel section, the voltage and frequency of the GPU are adjusted respectively (750 MHz, 650 MHz, 550 MHz). The experiment results show that using processor core shut-down and DVS technologies can reduce the energy consumption of parallel programs effectively, and the maximum energy saving can reach about 7.2%. The error rate in the energy saving and simulation energy saving is within ± 5%.

(3) The effect of critical sections on energy consumption

In order to suitably test the influence of critical sections on the frequency and consumption of energy using parallel programs, we carried out an investigation and analysis using typical parallel programming cases (Fig. 8).

As can be seen from the figure, the parallel segment of the parallel program consists of a completely-paralleled parallel segment and a critical section. The operating proportion of the critical section, σ of critical section is, was scaled without changing the computing amount in the parallel segment and the results used to estimate the effect of the critical section operations on the energy optimization of the parallel segments. Once again, the Intel Core i7 920 quad-core CPU was used as the experimental platform and the program compiled using gfortran (v4.2.1).

(a) Power overhead when the critical section is in a waiting state

In order to avoid overheads caused by other operations, the parallel segments are formed under the operation of a critical section ($\sigma = 1$) and therefore only one processor executes the effective calculation at any time. By modifying the environmental variable OMP_NUM_THREADS, the number of threads used in the concurrent execution can be adjusted.

Fig. 9 shows the change in the dynamic power overhead of the processor as the thread number changes. As can be seen from the figure, the total dynamic power consumed by the processor varies insignificantly. This implies that the processor does not cause a significant power overhead in the waiting state.

(b) Relationship between σ and optimal frequency

By adjusting the operating proportion of the critical section, σ, used in the test cases, we can detect the subsequent changes that occur in the frequency of the processor. The execution time when the processor operates in a non-critical state ($\sigma = 0$) using the lowest operating frequency is taken as the time constraint. In this work, we make a comparison of three different frequency values: (i) the 'theoretical' value (i.e. the frequency of the processor obtained using the analysis model), (ii) the 'physical' value (i.e. the actual frequency used by the processor in practice — corresponding to the lowest discrete operating frequency of the processor above the theoretical frequency), and (iii) the 'optimal' value (i.e. the ideal frequency according to the relationship between frequency and execution time under a discrete frequency value).

As shown in Fig. 10, the optimal frequency of the processor gradually rises with increasing σ value. When σ reaches 50%, the theoretical frequency exceeds the highest frequency allowed by the processor and therefore the physical value cannot be displayed.

(c) Reducing energy consumption by reducing conflict probability

The above analysis has shown that, compared to not using critical sections, the operating frequency of the processor is $(1 + \sigma^2 cn)$ times that of the original frequency (under the constraint that the execution time is the same). In this context, the effective computing time (eliminating the waiting time of the critical section) is $(1 + \sigma^2 cn)^{-1}$ that of the original time. Therefore, it can be seen that decreasing the conflict probability of the critical section (c) can effectively reduce the energy overhead, assuming σ is constant. For this study, we partitioned the critical section of the test program into two critical sections that can be executed concurrently using different processors. In this case, the conflict probability of the critical section can fall to about 50% which further saves energy by allowing the operating frequency of the processor to be decreased. Fig. 11 displays the energy optimization results obtained (all results are normalized values compared to the initial amount of energy consumed). The figure shows that reducing the conflict probability provides space for optimization to be made. This allows the frequency of the processor to be lowered which subsequently reduces the energy overhead of the system.

(3) Scalability of the solution

A new energy optimization of parallel programs (EOPP) model is proposed in this paper and its use can be extended to large-scale heterogeneous systems with multiple processing units, eg, data center [38-40]. At present, we have applied our theoretical results to the Tianhe-2A supercomputer. Based on the system load, energy consumption distribution, and hardware features, a power optimization management system is designed. The system architecture is composed of three layers (Fig.12), which are: the energy consumption decision layer, the perception control layer, and the hardware platform layer, respectively. The energy consumption decision layer is composed of six parts: a monitoring module, a job management module, a resource management module, a low-power compiling module, a peripheral device control module, and a power management interface module.

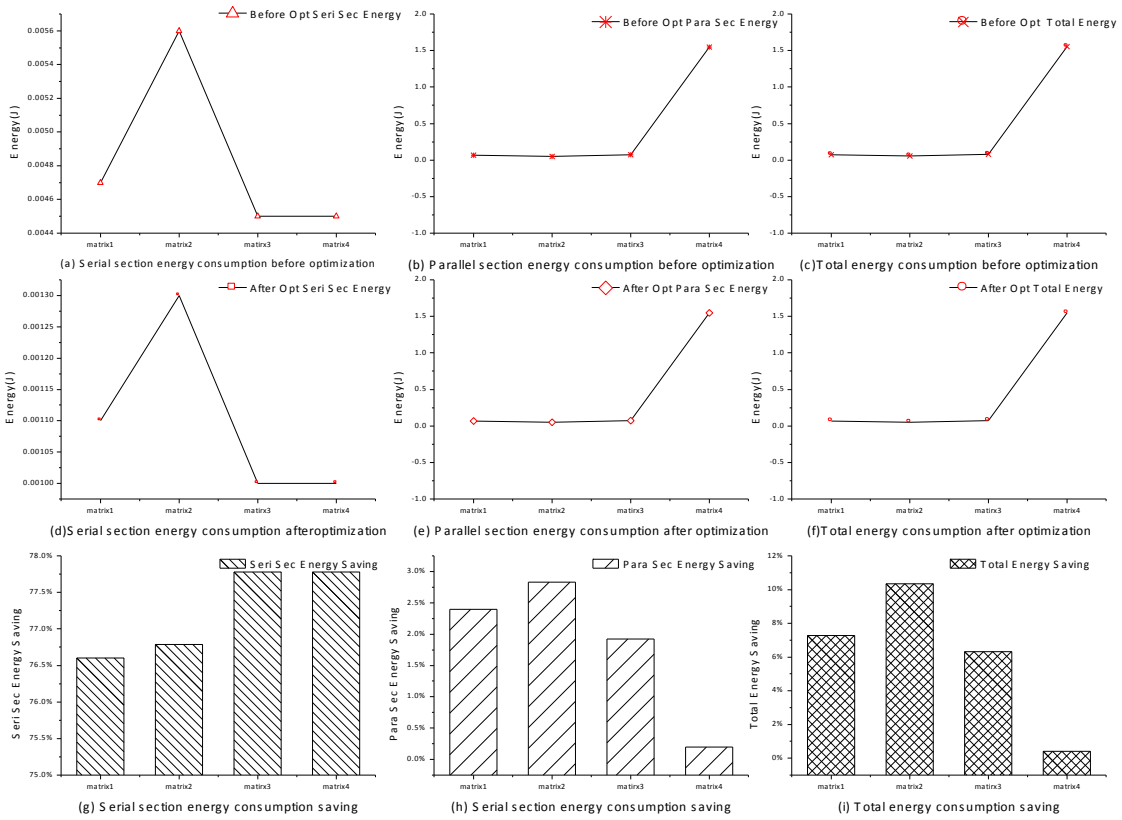

Fig. 6. The energy savings gained using the optimization process.

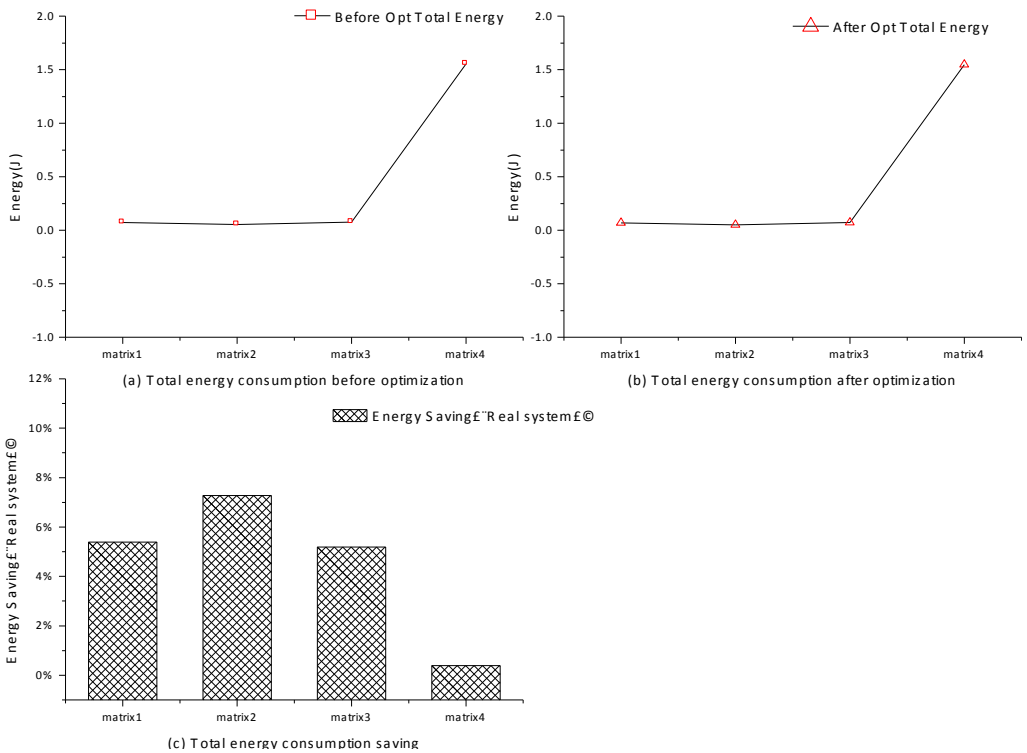

Fig. 7. Actual energy consumption optimization

```
Initial input array A
!$omp parallel
    do i = 1, n
        //full parallel code
        update A
        //critical section
        !$omp critical
            update A
        !$omp end critical
    enddo
!$omp end parallel
```

Fig. 8. Code illustrating the synchronization tests carried out on the critical sections.

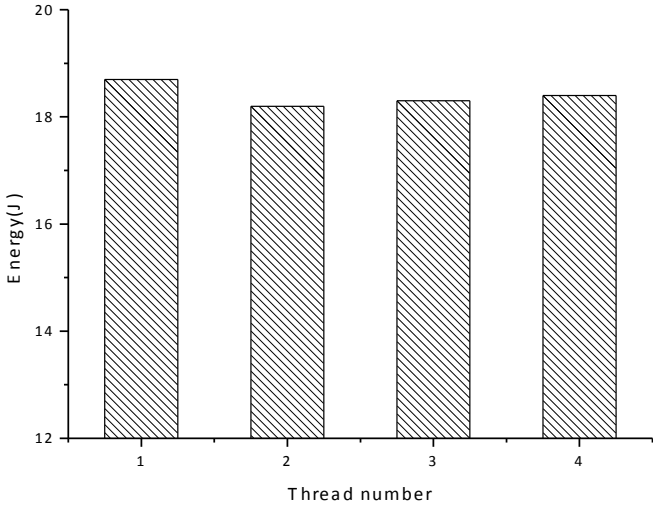

Fig. 9. The change in the power overhead with thread number.

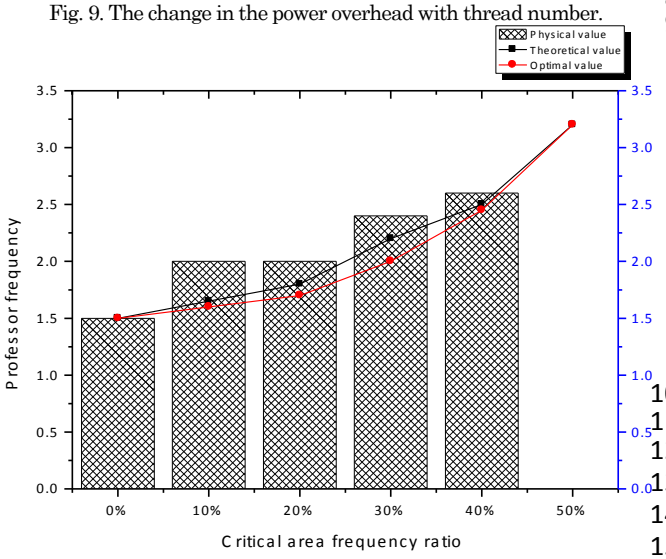

Fig. 10. Relationship between the operating frequency of the processor and the operating proportion of the critical section.

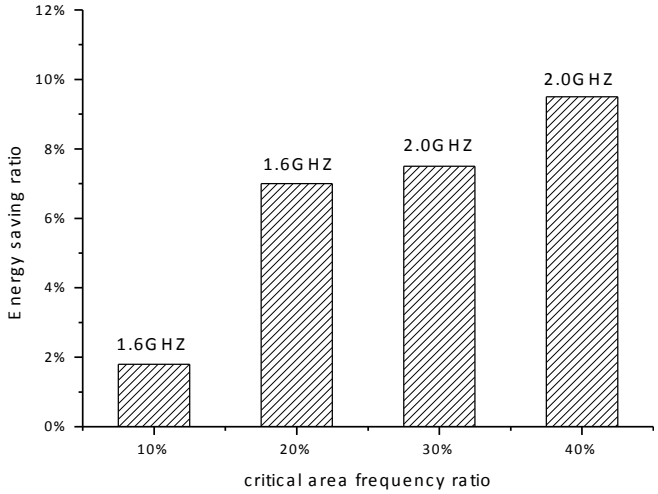

Fig. 11. Decreasing the energy consumption of the system by reducing the conflict probability of the critical section.

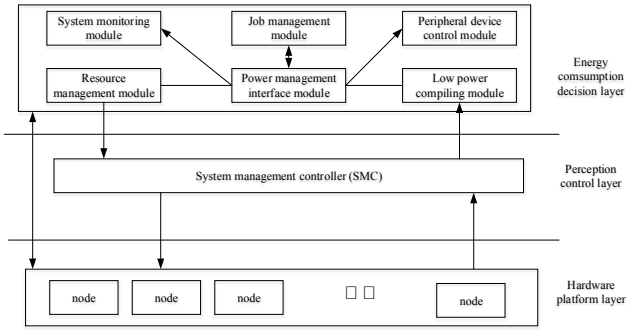

Fig. 12 Power management system architecture

The main function is to collect all data related to power consumption. The power optimization management system is integrated with monitoring, strategy making, power adjustment, and equipment control. It tries to reduce the whole system power while guaranteeing the performance, which is then combined with the methods of resource management, operation

management, low-power consumption compilation, and dynamic voltage adjustment.

6. Conclusions and future work

This study presents an energy optimization model (EOPP) for a heterogeneous system. By jointly utilizing processor core-shutdown and DVS, the model can instruct parallel programs to conduct energy optimization taking into account the energy overheads caused by re-synchronization, voltage switching, and the operation of critical sections.

Our experimental results show that the EOPP model is able to reduce the energy consumed by parallel programs, realizing a total energy saving of up to ~10.5%. Based on an experimental analysis of the effect of the operations in critical sections on energy consumption, it is speculated that increasing the proportion of the critical section causes a corresponding enhancement to be made to the optimal frequency of a processor. However, lowering the conflict probability of the critical section can provide optimization space which lowers the frequency of the processor and finally results in reduced energy overheads.

Due to the parallel program exists the load imbalance and data dependency during the actual execution process , the modelling of energy optimization for parallel program becomes very complicated. In the future work, the characteristic of the program parallelization will be further investigated, and the corresponding energy analysis and optimization will be carried out.

Acknowledgments

This work was sponsored by National Natural Science Foundation of China (grant number 61672168, 61300029, 61502110, 61672172, 61772143) , Guangzhou Major Science and Technology Projects (201604010096)

References and links

1. Geist A. Paving the Roadmap to Exascale. SciDACReview. 2010,16:52-59.
2. Holmbacka S, Keller J, Eitschberger P, Lilius J. Accurate energy modelling for many-core static schedules. Euromicro International Conference on Parallel Distributed & Network-Based Processing. Turku, 2015:525-532
3. Zhao Y, Li X, Ju L, Zong Z. Dependency-based energy-efficient scheduling for homogeneous muli-core clusters. IEEE International Conference on Trust, security and privacy in computing and communications. Melbourne, 2013:1299-1306.
4. Amilifard B., Fallah F., Pedram M. Low-power fanout optimization use in multiple threshold voltage inverters.Proceedings of the international symposium on low power electronics and design. New York, NY, USA, 2015:95-98.
5. Calhoun B H, Chandrakasan A. Characterizing and modeling minimum energy operation for subthreshold circuits. International Symposium on Low Power Electronics and Design. IEEE, 2004:90-95.
6. Donno M, Ivaldi A, Benini L, et al. Clock-tree power optimization based on RTL clock-gating. Design Automation Conference. ACM, 2003:622-627.
7. Wason V., Banerjee K. A probabilistic framework for power-optimal repeater insertion in global interconnects under parameter variations. Proceedings of the international symposium on Lowe power electronics and design. New York, NY, USA, 2005:131-136.
8. Kim N S, Blaauw D, Mudge T. Leakage Power Optimization Techniques for Ultra Deep Sub-Micron Multi-Level Caches. Ieee/acm International Conference on Computer-Aided Design. IEEE Computer Society, 2003:627.
9. Ananthan H, Kim C H, Roy K. Larger-than-Vdd Forward Body Bias in Sub-0.5V Nanoscale CMOS. International Symposium on Low Power Electronics and Design. IEEE, 2004:8-13.
10. Powell M D, Schuchman E, Vijaykumar T N. Balancing Resource Utilization to Mitigate Power Density in Processor Pipelines. Ieee/acm International Symposium on Microarchitecture, 2005. Micro-38. Proceedings. IEEE, 2005:294-304.
11. Jayaseelan R. Application-specific thermal management of computer systems. Singapore: National University of Singapore, 2009.
12. Wang Z, Xu X, Xiong N, et al. Energy cost evaluation of parallel algorithms for multiprocessor systems. Cluster Computing, 2013, 16(1):77-90.
13. Wang Z, Xiong N, Wang H, et al. Whole procedure heterogeneous multiprocessors low-power optimization at algorithm-level. Cluster Computing, 2018(1):1-17.
14. Wang Z, Zhao W, Wang H, et al. Three-level performance optimization for heterogeneous systems based on software prefetching under power constraints. Future Generation Computer Systems, 2018, 86:51-58.
15. Wang Z, Cheng L, Zhao W, et al. An architecture - level graphics processing unit energy model. Concurrency & Computation Practice & Experience, 2016, 28(10):2795-2810.
16. Kadayif I, Kandemir M, Karakoy M. An energy saving strategy based on adaptive loop parallelization.Design Automation Conference. ACM, 2002:195-200.
17. Kim H S, Vijaykrishnan N, Kandemir M, et al. Adapting instruction level parallelism for optimizing leakage in VLIW architectures. Acm Sigplan Notices, 2003, 38(7):275-283.
18. Hsu C H, Kremer U, Hsiao M. Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors. Low Power Electronics and Design, International Symposium on. IEEE, 2001:275-278.
19. Kim N S, Flautner K, Blaauw D, et al. Circuit and microarchitectural techniques for reducing cache leakage power. IEEE Transactions on Very Large Scale Integration Systems, 2004, 12(2):167-184.
20. Zhao R.C, Tang Z.M, Zhang Z.Q, et al. A multithreaded compiler optimization technology with low power. Journal of software, 2002, 13(1):1123-1129.
21. Grochowski E, Ronen R, Shen J, et al. Best of both latency and throughput. Computer Design: VLSI in

Computers and Processors, ICCD 2004. Proceedings. IEEE International Conference on. IEEE, 2004:236-243.
22. Kadayif I, Kandemir M, Sezer U. An integer linear programming based approach for parallelizing applications in On-chip multiprocessors. Design Automation Conference, 2002. Proceedings. IEEE, 2002:703-708.
23. Kadayif I, Kandemir M, Kolcu I. Exploiting Processor Workload Heterogeneity for Reducing Energy Consumption in Chip Multiprocessors. Conference on Design, Automation and Test in Europe. IEEE Computer Society, 2004:21158.
24. Li N J, Martinez J F. Power-Performance Implications of Thread-level Parallelism on Chip Multiprocessors. IEEE International Symposium on PERFORMANCE Analysis of Systems and Software. IEEE Computer Society, 2005:124-134.
25. Li J, Martinez J F. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In Proceedings of the International Symposium on High Performance Computer Architecture. 2006:77-87.
26. Eyerman S, Eeckhout L. Modeling critical sections in Amdahl's law and its implications for multicore design. Acm Sigarch Computer Architecture News, 2010, 38(3):362-370.
27. Nowka K, Carpenter G, Donald E M, et al. A 0.9 V to 1.95 V dynamic voltage-scalable and frequency-scalable 32 b PowerPC processor. Solid-State Circuits Conference, 2002. Digest of Technical Papers. ISSCC. IEEE International. 2002(1):340-341.
28. Delaluz V, Kandemir M, Vijaykrishnan N, et al. Energy-oriented compiler optimizations for partitioned memory architectures. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. ACM, 2000:138-147.
29. Burd T D. Design issues for dynamic voltage scaling. In Proceedings of 2000 International Symposium on Low Power Electronics and Design, 2000:9-14.
30. Stijn Eyerman, Lieven Eeckhout. Modeling critical sections in Amdahl's law and its implications for multicore design. International Symposium on Computer Architecture. ACM, 2010:362-370.
31. Du I S, Grimes R G, Lewis J G, et al. User's Guide for the Harwell-Boeing Sparse Matrix Collection. Tech. Report TR-PA-92-96. Toulouse Cedex, France, Oct 1992.
32. http://www.cise.ufl.edu/research/sparse/HBformat/HB/.
33. Yang X J, Tang T, Wang G B, et al. MPtostream: an OpenMP compiler for CPU-GPU heterogeneous parallel systems. Science China(Information Sciences), 2012, 55(9):1961-1971.
34. Enhanced Intel SpeedStep Technology for the Intel Pentium M Processor, White Paper. March, 2004.
35. Powerplay Technology. http://www.amd.com/us/products/technologies/ati-power-play/Pages/ati-power-play.aspx.
36. Advanced Configuration and Power Interface. http://www.acpi.info/
37. AMD Display Libray. http://developer.amd.com/GPU/ADLSDK/Pages/default.aspx.
38. Shuja J, Bilal K, Madani S A, et al. Survey of Techniques and Architectures for Designing Energy-Efficient Data Centers. IEEE Systems Journal, 2016, 10(2):507-519.
39. Shuja J, Gani A, Shamshirband S, et al. Sustainable Cloud Data Centers: A survey of enabling techniques and technologies. Renewable & Sustainable Energy Reviews, 2016, 62:195-214.
40. Shuja J, Bilal K, Madani S A, et al. Data center energy efficient resource scheduling. Cluster Computing, 2014, 17(4):1265-1277.