AOA: Adaptive Overclocking Algorithm on CPU-GPU Heterogeneous Platforms

Zhixin Ou, Juan Chen(✉), Yuyang Sun, Tao Xu, Guodong Jiang, Zhengyuan Tan, and Xinxin Qi

National University of Defense Technology, Changsha, China
{ouzhixin16,juanchen,sunyuyang,xt.0320,jiangguodong,tanzhengyuan, qixinxin19}@nudt.edu.cn

Abstract. Although GPUs have been used to accelerate various convolutional neural network algorithms with good performance, the demand for performance improvement is still continuously increasing. CPU/GPU overclocking technology brings opportunities for further performance improvement in CPU-GPU heterogeneous platforms. However, CPU/GPU overclocking inevitably increases the power of the CPU/GPU, which is not conducive to energy conservation, energy efficiency optimization, or even system stability. How to effectively constrain the total energy to remain roughly unchanged during the CPU/GPU overclocking is a key issue in designing adaptive overclocking algorithms. There are two key factors during solving this key issue. Firstly, the dynamic power upper bound must be set to reflect the real-time behavior characteristics of the program so that algorithm can better meet the total energy unchanging constraints; secondly, instead of independently overclocking at both CPU and GPU sides, coordinately overclocking on CPU-GPU must be considered to adapt to real-time load balance for higher performance improvement and better energy constraints. This paper proposes an *Adaptive Overclocking Algorithm* (AOA) on CPU-GPU heterogeneous platforms to achieve the goal of performance improvement while the total energy remains roughly unchanged. AOA uses the function F_k to describe the variable power upper bound and introduces the load imbalance factor W to realize the CPU-GPU coordinated overclocking. Through the verification of several types convolutional neural network algorithms on two CPU-GPU heterogeneous platforms (Intel®Xeon E5-2660 & NVIDIA®Tesla K80; Intel®Core™i9-10920X & NIVIDIA®GeForce RTX 2080Ti), AOA achieves an average of 10.7% performance improvement and 4.4% energy savings. To verify the effectiveness of the AOA, we compare AOA with other methods including automatic boost, the highest overclocking and static optimal overclocking.

Keywords: Heterogeneous platform · Overclocking · Performance optimization · Energy consumption

1 Introduction

Almost all kinds of convolutional neural network algorithms and scientific computing programs generally use GPU for acceleration [4,8]. CPU-GPU heterogeneous platforms are becoming more and more widely used. Both CPU and GPU provide overclocking mechanisms, and moderate overclocking brings opportunities for further performance improvement. Although the CPU has the Intel Turbo Boost Technology [15], and the GPU has the Nvidia GPU Boost Technology [22], automatic overclocking through the system has certain limitations. It is meaningful to consider the goal that how to improve the performance by adaptively overclocking on CPU-GPU heterogeneous platforms while constraining the total energy consumption unchanged.

Existing researches on overclocking algorithms have targeted both homogeneous and heterogeneous platforms. One of the state-of-art algorithms for homogeneous platforms is to set a static frequency upper bound [38], which overclocks the frequency to the given upper bound in active state. Another existing method is overclocking under power constrain [14]. When the processor utilization is high and current power does not reach the power upper bound, the frequency keeps increasing. However, how to control the overall energy consumption of a certain task or program is a problem, especially for heterogeneous platforms with fewer overclocking algorithms [31]. A common way to improve CPU-GPU energy efficiency is to adjust CPU and GPU frequency on heterogeneous platforms. During the entire frequency adjustment process, the CPU and GPU are usually regarded as a whole for consistent frequency adjustment [36]. When it comes to CPU-GPU frequency adjustment on heterogeneous platforms, load balance between CPU and GPU is one of the key issues to consider. Frequency adjustment brings greater challenges to load balance.

For overclocking on CPU-GPU heterogeneous platforms, a commonly used method is overclocking under a given power upper bound, which achieves the purpose of improving performance while satisfying the total energy constraint. How to effectively keep the total energy unchanged during the CPU/GPU overclocking process is a key issue, which mainly includes two key factors.

– The static power upper bound is difficult to reflect the individual differences in performance and power of programs' behavior, which brings problems to improving program performance and the accuracy of power control. In addition to CPU/GPU frequency, the real-time usage of the program on the processor (such as instructions per cycle, processor core utilization, etc.) can also affect instantaneous processor power. Therefore, *power upper bound must be dynamic, not static.*
– Overclocking may cause or even aggravate the load imbalance of CPU-GPU heterogeneous platforms, thereby affecting performance improvement and total energy constrain. Independent overclocking of the CPU and GPU may cause the originally balanced load to become unbalanced, which is detrimental to our goals. When the CPU-GPU load imbalance increases, the performance has room for improvement, because the performance of heterogeneous

computing depends on the side with longer running time. Also, additional energy is consumed due to waiting. Therefore, *CPU-GPU overclocking must be coordinated, considering real-time load balance.*

Based on the two key factors in the design of overclocking algorithms, the solutions in this article are as follows.

- For the first key factor, turn the *static* power upper bound of overclocking into the adaptive *variable* power upper bound. The overclocking algorithm sets the initial power upper bound k_0 by calculating the expected power upper bound according to the characteristics of the usage of various programs on a given platform. In the dynamic overclocking process, the power upper bound k is adaptively adjusted to achieve precise real-time power control for individual program characteristics, according to the real-time usage of the current program on the CPU and GPU.
- For the second key factor, the load imbalance factor W is used to characterize whether the program is in the load balance state or load imbalance state, and to further adjust the overclocking range after normal overclocking for a further performance improvement.

In summary, this paper has the following three contributions.

1. Aiming at the challenge of constraining the total energy in overclocking, we find two key factors that affect adaptive overclocking: dynamic variable power upper bound k and coordinated overclocking (load imbalance factor W) at both sides of the CPU and GPU.
2. We propose an Adaptive Overclocking Algorithm on CPU-GPU Heterogeneous platforms (AOA), which first adjusts the dynamic power upper bound k of CPU and GPU separately according to the real-time operation of the current program on the CPU-GPU heterogeneous platforms; and introduce load imbalance factor W to coordinate overclocking range by improving load imbalance phenomenon for further performance improvement and better total energy constrain.
3. AOA is implemented and verified on two heterogeneous platforms (Intel®Xeon E5-2660 & NVIDIA®Tesla K80; Intel®Core™i9-10920X & NIVIDIA®GeForce RTX 2080Ti) using convolutional neural network algorithms. The experimental results show that the AOA improves the performance by 10.7% on average and 4.4% energy savings. To verify the effectiveness of the AOA, we compared AOA with methods including automatic boost, the highest overclocking and static optimal overclocking.

The structure of this article is as follows: Sect. 2 introduces the background and related work of overclocking methods on heterogeneous platforms. Section 3 illustrates the design ideas of overclocking algorithms and the description of AOA. Section 4 introduces the experimental platform and the experimental results of AoA on two heterogeneous platforms. Section 5 discusses the impact of two key factors on AOA effect. Section 6 is the concluding remarks.

2 Related Work

2.1 DNN in CPU-GPU Platforms

The heterogenous architecture becomes widely used in high performance computing [18,35], which is inseperable from the requirement for boost performance (high accuracy and fast training speed) of DNNs. While convolutional neural network (CNN) is one of the most successful deep neural network (DNN) models and have been applied in various fields such as computer vision and speech recognition [12,20], the cost of high power and energy attracts widespread attention [21,26,32].

Th inspires researches on energy efficiency of DNN models on CPU-GPU platforms. Sun et al. study the performance, power and energy characteristics of CNNs when running on CPU and GPU [29]. Rodrigues et al. [25] propose an evaluation framework that combines standard deep learning frameworks such as Caffe [17] and cuDNNv5 [8] to measure the performance and energy of DNNs. Mittal et al. [21] survey on methods for analyzing and improving GPU energy efficiency.

2.2 Energy Efficiency Optimization Methods

Existing energy efficiency optimization methods for heterogeneous platforms mainly have two types of ideas. One type of methods to improve energy efficiency is to reduce power or energy consumption while maintaining performance, which is realized by dynamic voltage frequency scaling (DVFS, usually frequency reduction). Yao [37] study the energy efficiency of CNN on high performance GPUs. Through a large number of comparative experiments on GPU architecture, DVFS settings and DNN configuration, they studied the impact of GPU DVFS on deep learning energy consumption and performance based on experience. Tang et al. [30] also research on the impact of GPU DVFS on the energy and performance of deep learning. However, reduced frequencies sometimes mean slower system configurations, which to some extent violate latency and throughput requirements of heterogeneous platforms in the context of high performance computing.

Another type of methods is to improve performance while controlling power and energy consumption, which is mainly achieved by overclocking. Existing works research on the effect of overclocking on performance [31]. Sasikala et al. [24] exploit the techniques of overclocking and throttling to enhance the performance while maintaining the system reliability. Wu et al. [33,34] improves energy efficiency by processor overclocking and memory frequency scaling. Yang et al. [36] treat the CPU and GPU as a whole for consistent frequency modulation. However, there are significant imbalance in both frequency and computing ability of CPUs and GPUs, which is ignored by consistent frequency modulation.

Researches show that load imbalance between CPU and GPU in some programs (such as BFS) also has important impact on performance [10]. Acun et al. [5] agree that performance inhomogeneity in power constrained environments drastically limits supercomputing performance. Chen et al. [7] investigate the optimization possibilities of the performance of parallel systems utilizing the time-dimension communication features. Chasapis et al. [6] research on the effect of manufacturing variability on computing ability. Gholkar et al. [11] show that a power capped job suffers from performance variation of otherwise identical processors leading to overall sub-optimal performance.

3 AOA Design

3.1 Key Issue and Key Factors in AOA

Key Issue - Energy Constraint. Overclocking brings performance improvement together with inevitable increase in instantaneous power, which makes the total energy likely to increase or decrease. If the total energy can be restrained from increasing during the overclocking process, it will not only save the energy consumption of the task ($E = P \cdot t$), but also improve the energy efficiency ($EDP = E \cdot t$). *How to effectively constrain the total energy unchanged in the overclocking process is a key issue in the overclocking algorithm design.*

Furthermore, dynamic overclocking is more beneficial to meet the constant total energy constraint than static overclocking, so we adopt dynamic overclocking instead of static overclocking.

The algorithm is designed with the total energy unchanged as the constraint. There are usually three control conditions: controlling the total energy unchanged, controlling frequency not exceed the max frequency, and controlling power not exceed the power upper bound.

There are some problems in designing dynamic overclocking by controlling the total energy unchanged. Since the total energy is the accumulation of power over a period of time, the total energy consumption of the program cannot be obtained before the program finishes running. Therefore, it is not possible to design dynamic overclocking by controlling the total energy unchanged in the algorithm. Setting the power upper bound for overclocking can avoid the above problem. The power can be obtained in real-time during the running of the program. Compared with the total energy, power upper bound can be directly used as a constraint that is easier to control dynamic overclocking.

There are also some problems in controlling the frequency not to exceed the upper bound. Although the performance improvement goal can be achieved by using frequency as the control condition, it is difficult to meet the constraint condition of constant total energy. The reason is that in addition to frequency, the real-time occupation factors of the processor by the program (such as instructions per cycle, processor core utilization, etc.) also affect instantaneous power. The formula (1) reflects the relationship between the processor clock frequency f and the instantaneous processor power P:

$$P_{total} = \alpha C V^2 f + P_{static} \tag{1}$$

where α is the CMOS circuit switching factor that reflects the busyness of the processor. As shown in Table 1, taking platform A as an example, when the CPU frequency f is at 2.6 GHz, the power values under 10% and 88% utilization are 60W and 140W respectively, with a difference of 133%. Compared with controlling frequency not exceeding max frequency, controlling instantaneous power not exceeding a given power upper bound is more helpful to achieve the goal of remaining the total energy unchanged.

Key Factors - Power Upper Bound and Load Balance

Key factor 1: The power upper bound must be dynamic, not static.

In the process of dynamic overclocking, if the upper bound of power is fixed to a value, that is not the most appropriate because it will bring difficulties to the control of total energy. We define a variable upper bound of dynamic power kP that is a variable parameter and different from the hardware power capping. Among them, P is the power value of the processor at the default frequency f_0. Considering that the value of P has a certain fluctuation relative to the same frequency f_0 under actual conditions, we take the typical power value of the processor as P. k is the Power Upper Bound Factor, which satisfies $1 \leq k \leq 1.2$. When $k = 1$, it means no overclocking; when $k = 1.2$, it means that the processor power after overclocking cannot be greater than 1.2 times of P. 1.2 is an empirical value obtained from historical experiments on the platform, and the value setting varies slightly on different platforms.

The design of the dynamic power upper bound is based on two considerations. One is the determination of the initial value of the overclocking power coefficient, and the other is the dynamic change process of the overclocking power coefficient.

Define k_0 as the initial value of variable power upper bound factor. By running various test programs on a given hardware platform (considering the balance of program feature distribution), the power and energy results of the processor under different occupancy conditions can be obtained, and then the power upper bound under the constraint of constant total energy can be obtained. The overclocking power coefficient k_0 is determined from this.

On the basis of k_0, the variable overclocking power coefficient k_t at time t is calculated by the function F_k.

$$k_t \leftarrow F_k(k_0, U_t) \tag{2}$$

where U_t represents the real-time processor occupancy of the program at time t, which is calculated according to processor utilization *utilization_percentage*, i.e. $U_t = utilization\_percentage \times f$.

Particularly, on the GPU side, the function of overclocking power coefficient is as follows.

$$k_t^{GPU} \leftarrow F_k^{GPU}(k_0, U_t^{GPU}) \tag{3}$$

On the CPU side, the function of overclocking power coefficient is as follows.

$$k_t^{CPU} \leftarrow F_k^{CPU}(k_0, U_t^{CPU}) \tag{4}$$

Key factor 2: Overclocking at both sides of the CPU and GPU must be coordinated, and real-time load balancing must be considered.

In order to judge whether the program is in a load balance state during the running process, the Load Imbalance Factor W and the threshold γ are introduced. Load imbalance factor at time t is denoted as W_t where $W_t = \frac{U^{GPU} - U^{CPU}}{U^{GPU} + U^{CPU}}$. As shown in Fig. 1, the program running on the CPU-GPU heterogeneous platform may be in the load balance state (slash fill area, $|W_t| \le \gamma$) or in the load imbalance state (color fill area, $|W_t| > \gamma$).

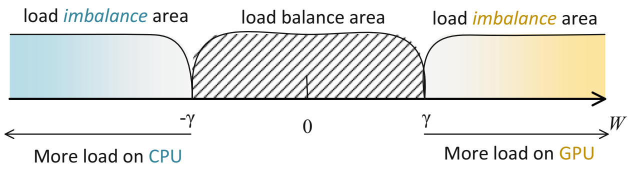

Fig. 1. Introduce load imbalance factor W to describe load balance area and load imbalance area (Color figure online)

When the program is in the load imbalance area, adjust the CPU and GPU frequencies according to the load balance adjustment function F_f to make the program in the load balance area again.

$$(f^{CPU}, f^{GPU}) \leftarrow F_f(f^{CPU}, f^{GPU}, W_t) \tag{5}$$

When $W > \gamma$, the workload on GPU side is heavier, the function F_f increases the GPU frequency f^{GPU} or decreases the CPU frequency f^{CPU}, where $f^{GPU}_{base} \le f^{GPU} \le f^{GPU}_{max}$, $f^{CPU}_{base} \le f^{CPU} \le f^{CPU}_{max}$, as follows.

$$\begin{cases} f^{GPU} \leftarrow f^{GPU} + \Delta f^{GPU}, & if\ (f^{GPU} + \Delta f^{GPU} \le f^{GPU}_{max}) \\ f^{CPU} \leftarrow f^{CPU} - \Delta f^{CPU}, & else\ if\ (f^{CPU} - \Delta f^{CPU} \ge f^{CPU}_{base}) \\ (f^{CPU}, f^{GPU})\ unchanged, & else \end{cases} \tag{6}$$

When $W < \gamma$, the workload on CPU side is heavier, the function F_f increases the CPU frequency f^{CPU} or decreases the GPU frequency f^{GPU}. The overclocking range is the same as before and the details are as follows.

$$\begin{cases} f^{CPU} \leftarrow f^{CPU} + \Delta f^{CPU}, & if\ (f^{CPU} + \Delta f^{CPU} \le f^{CPU}_{max}) \\ f^{GPU} \leftarrow f^{GPU} - \Delta f^{GPU}, & else\ if\ (f^{GPU} - \Delta f^{GPU} \ge f^{GPU}_{base}) \\ (f^{CPU}, f^{GPU})\ unchanged, & else \end{cases} \tag{7}$$

3.2 Description of AOA

The flowchart of the Adaptive Overclocking Algorithm (AOA) is shown in Fig. 2. Among them, the variable power upper bound factors on CPU and GPU are denoted as (k_{t+1}^{CPU}, k_{t+1}^{GPU}). Based on the t^{th} overclocking cycle CPU and GPU usage ratio U_t^{CPU} and U_t^{GPU}, k_{t+1}^{CPU} and k_{t+1}^{GPU} are updated on CPU and GPU, respectively. (corresponding to Factor 1, represented by blue and yellow colors in Fig. 2).

The gray box part is dynamic overclocking under variable power upper bound power control. On the basis of the normal CPU and GPU overclocking (represented by blue and yellow colors respectively), the frequency values at both CPU and GPU are further updated according to the load balance factor W_t (f_{t+1}^{CPU}, f_{t+1}^{GPU}), that is, CPU-GPU cooperative overclocking (corresponding to Factor 2, blue and yellow mixed colors).

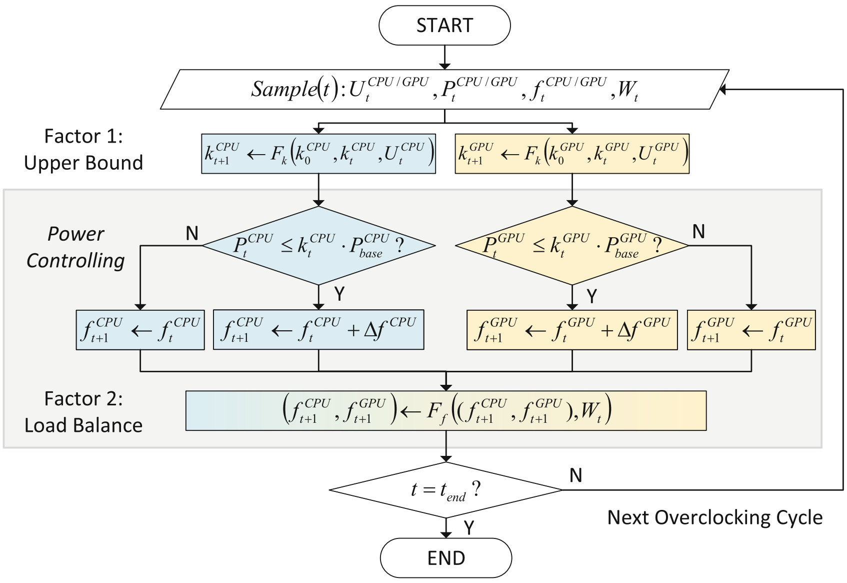
START
$Sample(t): U_t^{CPU/GPU}, P_t^{CPU/GPU}, f_t^{CPU/GPU}, W_t$
Factor 1: Upper Bound
$k_{t+1}^{CPU} \leftarrow F_k(k_0^{CPU}, k_t^{CPU}, U_t^{CPU})$
$k_{t+1}^{GPU} \leftarrow F_k(k_0^{GPU}, k_t^{GPU}, U_t^{GPU})$
Power Controlling
$P_t^{CPU} \le k_t^{CPU} \cdot P_{base}^{CPU}$?
$P_t^{GPU} \le k_t^{GPU} \cdot P_{base}^{GPU}$?
N
Y
Y
N
$f_{t+1}^{CPU} \leftarrow f_t^{CPU}$
$f_{t+1}^{CPU} \leftarrow f_t^{CPU} + \Delta f^{CPU}$
$f_{t+1}^{GPU} \leftarrow f_t^{GPU} + \Delta f^{GPU}$
$f_{t+1}^{GPU} \leftarrow f_t^{GPU}$
Factor 2: Load Balance
$(f_{t+1}^{CPU}, f_{t+1}^{GPU}) \leftarrow F_f((f_{t+1}^{CPU}, f_{t+1}^{GPU}), W_t)$
$t = t_{end}$?
N
Next Overclocking Cycle
Y
END

Fig. 2. Flow graph of the adaptive overclocking algorithm. (Color figure online)

The detailed description of the algorithm sees Algorithm 1.

Dynamic overclocking will cause additional time overhead. We evaluate the time overhead incurred by one overclocking operation or frequency scaling operation. Manually insert N times of overclocking or frequency scaling operations, then run the same program, and get the execution time of the program with the N times of overclocking, denoted as T. The default time of the program without overclocking is denoted as T_0. Comparing T and T_0 can obtain the additional time overhead due to the N times of overclocking or frequency scaling operations. Finally, the time overhead for a single overclocking or frequency scaling is calculated that is about 4 ms.

Algorithm 1. Adaptive Overclocking Algorithm

```
Input: program, $k_0$, $P_{base}^{CPU/GPU}$, $f_{base}^{CPU/GPU}$, $f_{max}^{CPU/GPU}$, $\Delta f^{CPU/GPU}$.
Output: $k_t$, $f_t^{CPU/GPU}$ $(t = 1, 2, 3..., t_{end})$.
 1: $t \leftarrow 1, k_t \leftarrow k_0$;
 2: for each overclocking cycle $t$ $(t = 1, 2, 3, ..., t_{end})$ do
 3:     Sample data: $U_t^{CPU/GPU}$, $P_t^{CPU/GPU}$, $f_t^{CPU/GPU}$, $W_t$;
 4:     Output $k_t$ and $f_t^{CPU/GPU}$;
 5:     Update Power Upper Bound Factor $k_{i+1}^{CPU/GPU} \leftarrow$
        $F_k(k_0^{CPU/GPU}, k_t^{CPU/GPU}, U_t^{CPU/GPU})$;
 6:     if $(P_t^{CPU/GPU} \leq k_t^{CPU/GPU} \times P_{base}^{CPU/GPU})$ then
 7:         $f_{t+1}^{CPU/GPU} \leftarrow f_t^{CPU/GPU} + \Delta f^{CPU/GPU}$;
 8:     else
 9:         $f_{t+1}^{CPU/GPU} \leftarrow f_t^{CPU/GPU}$;
10:     end if
11:     $(f_{t+1}^{CPU}, f_{t+1}^{GPU}) \leftarrow F_f((f_t^{CPU}, f_t^{GPU}), W_t)$;
12: end for
13: return $k_t$, $f_t^{CPU/GPU}$
```

4 Experimental Results and Analysis

4.1 Experiment Environment

Heterogeneous Platform Hardware Environment. The experiments are carried out on two CPU-GPU heterogeneous platforms, called platform A and platform B, respectively. The specific configuration is shown in Table 1.

Table 1. Configuration parameters of platforms A and B

Environment	Platform A		Platform B	
	CPU	GPU	CPU	GPU
Processor type	Intel® Xeon E5-2600 v3	NVIDIA® Tesla K80	Intel® Core™ i9-10920X	NVIDIA® GeForce RTX 2080Ti
Frequency	2.6 GHz	575 MHz	3.5 GHz	1455 MHz
Power	105 W	150 W	165 W	285 W
#cores	10	2496	12	4352
#CPU (#GPU)	2	4	1	2
Hyperthreading	Disable	–	Disable	–
Auto-boost	Supported	Supported	Supported	Not available

Software Environment

Dataset and Benchmark. As for data set, this article uses ImageNet [9], a very famous data set in the field of image vision (CV). ImageNet is a recognition system that simulates human beings and is currently the world's largest database in the field of image recognition. After being proposed in 2009, ImageNet triggered the ILSVRC (ImageNet Large-Scale Visual Recognition Challenge), the most influential competition in the field of deep learning in the following years.

As for benchmark, this article has selected several classic CNN models [27] that have won the previous ILSVRC competitions, including alexnet [19], resnet [13] and vgg [28]. The specific program code is officially provided by pytorch on GitHub [23].

Monitoring Tools. On heterogeneous platforms, the power and performance indicators on CPU and those on GPU need to be monitored separately.

The power of the CPU is monitored by the `dstat` [1] tool. `dstat` allows users to view all your system resources in real time, and provides detailed and selective information, such as power/energy-pkg. The performance monitoring counters (PMC) of the CPU is monitored by `perf` [2]. `perf` is an analyzer tool that can abstract CPU hardware differences in Linux performance measurement and provides a simple command line interface based on the perf_events interface exported by the latest version of the Linux kernel.

The performance and power of the GPU are monitored through the NVIDIA System Management Interface (`nvidia-smi` [3]). `nvidia-smi` is a command line utility, based on the NVIDIA Management Library (NVML), designed to help manage and monitor NVIDIA GPU devices.

Parameter Settings. According to the AOA in Fig. 2, the parameters involved in the paper mainly include the variable power upper bound factor k, the load imbalance threshold γ and the frequency gear Δf. In the experiment, $k_0 = 1.1$, and $\gamma = 0.2$.

Table 2. Frequency gear settings of platforms A and B

Gears	Platform A (MHz)				Platform B (MHz)			
	CPU	Δf^{CPU}	GPU	Δf^{GPU}	CPU	Δf^{CPU}	GPU	Δf^{GPU}
0	2100	–	575	–	2700	–	1350	–
1	2200	100	614	39	2800	100	1365	15
2	2300	100	653	39	3000	200	1380	15
3	2400	100	692	39	3200	200	1395	15
4	2500	100	732	40	3300	100	1410	15
5	2600	100	771	39	3500	200	1425	15
6	–	–	810	39	–	–	–	–
7	-	–	849	39	–	–	–	–

The setting of frequency gear depends on the support of CPU and GPU hardware platform (see Sect. 4.1). For CPU frequency modulation, Intel provides the official overclocking tool XTU [16] for Intel Turbo Boost Technology.

However, the manual mentions that XTU only supports consumer-level products, not enterprise-level servers. The CPU frequency is set by the 'cpupower frequency-set' command.

For CPU frequency setting, Platform A uses 1350 MHz as the default CPU frequency, and sets up five CPU frequencies at 100 MHz intervals as overclocking options; Platform B uses 2.7 GHz as the default CPU frequency, and sets it upwards [2.8 GHz, 3.0 GHz, 3.2 GHz, 3.3 GHz, 3.5 GHz] as CPU overclocking options.

For GPU frequency setting, `nvidia-smi` provides a list of core frequencies supported by the GPU. Platform A uses 575 MHz as the default GPU frequency, and sets up seven GPU frequencies at an interval of about 39 MHz as overclocking options; Platform B uses 1350 MHz as the default GPU frequency, and sets up seven frequencies at 15 MHz intervals as overclocking options.

In summary, we set the CPU/GPU frequency modulation gear Δf in the experiment as shown in Table 2.

Comparisons. Other methods compared to the AOA method are shown below.

default: The default frequency is fixedly set to gear 0 in Table 2.
static$_{max}$: The static$_{max}$ frequency is fixedly set to the max gear in Table 2.
static$_{best}$: The static$_{best}$ frequency is fixedly set by running the program at each frequency combination once, and selecting the configuration corresponding to the result with the best energy efficiency as the static$_{best}$ frequency.
auto: The automatic frequency is set by official overclocking tools, XTU [16] for Intel Turbo Boost Technology and `nvidia-smi` for GPU overclocking.

4.2 AOA Overall Result

This section compares the results of AOA and default, and gives the AOA on utilization $U^{CPU/GPU}$, power $P^{CPU/GPU}$, and frequency $f^{CPU/GPU}$ result of platform A. In this chapter, the default CPU and GPU frequency is set to gear 0 in Table 2.

On platform A, the result of alexnet is shown in Fig. 3, the result of resnet is shown in Fig. 4, and the result of vgg is shown in Fig. 5. The two semi-axes of the vertical axis in the figure are both positive axes, the upper half represents the result on the GPU side, and the lower half represents the result on the CPU side.

Taking resnet as an example in Fig. 3. From the perspective of change trend, due to the design of epoch in the CNN algorithm, its utilization and power reflect periodicity in the time dimension. Comparing the results of AOA (solid black line) and default (dashed gray), we can find that the utilization value of a single epoch in alexnet has little change (in Fig. 3-a); but in terms of running time, AOA is overall shorter than default. From Fig. 3-b, it can be further seen that the change trend of power and utilization is consistent, and the power curve of AOA is slightly higher than the default. This increase in power is caused by the increase in frequency.

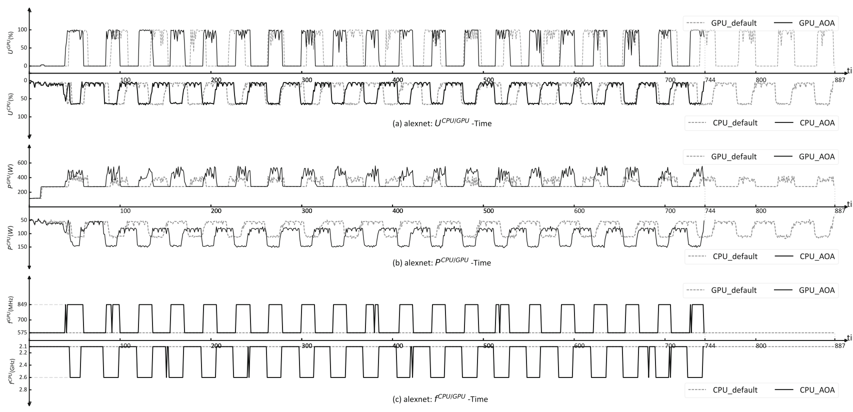

Fig. 3. $U^{CPU/GPU}$, $P^{CPU/GPU}$ and $f^{CPU/GPU}$ for alex in default frequency and AOA on platform A.

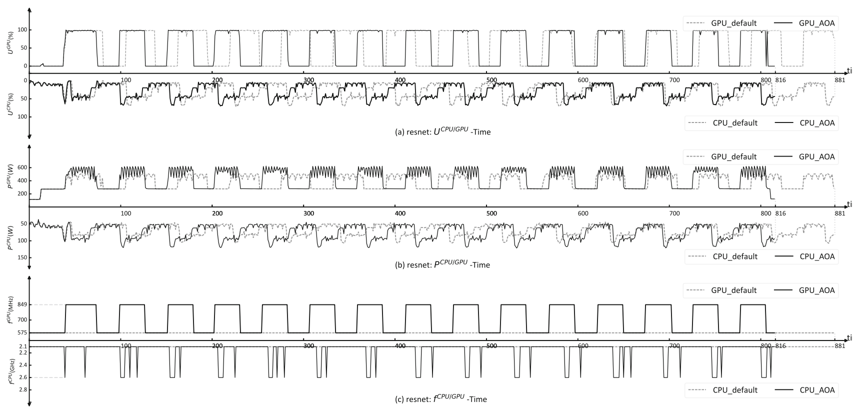

Fig. 4. $U^{CPU/GPU}$, $P^{CPU/GPU}$ and $f^{CPU/GPU}$ for resnet in default frequency and AOA on platform A.

Figure 3-c shows the adaptive dynamic adjustment effect of AOA on frequency. Corresponding to the utilization rate and power diagram, it can be found that when the utilization rate and power increase, AOA automatically increases the frequency according to the algorithm flow (Fig. 2). When the utilization rate and power decrease, the AOA automatically increases the frequency. Adapt to lower frequency.

On platform A, comparing AOA and default, the performance of alexnet is increased by 7.8%, and the energy is reduced by about 11%; the performance of resnet is increased by 8.3%, and the energy is increased by about 0.2%; the performance of vgg is increased by 16.1%, and the energy is reduced by 2.5%. The experimental results have reached the expected goal of the algorithm design, which is to improve the performance under the constraint of constant total

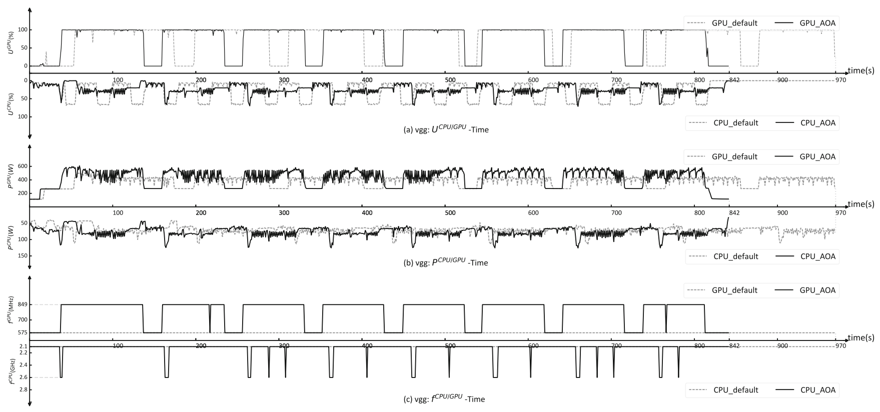

Fig. 5. $U^{CPU/GPU}$, $P^{CPU/GPU}$ and $f^{CPU/GPU}$ for vgg in default frequency and AOA on platform A.

energy. It is worth noting that the reduced energy can actually be used to further improve performance, and the overclocking frequency supported by the platform in the experiment has reached the upper bound.

Due to space limitations, the results on platform B are shown in Table 4. On platform B, it is difficult to increase energy efficiency because the power of NVIDIA GeForce RTX 2080Ti increases rapidly when the frequency overclocks slightly. Comparing AOA with default, even though the average performance is increased by 1.5%, and the average energy is increased by about 3.5%, AOA performs better in constraining total energy compared to max frequency.

4.3 Comparison with Other Methods

Platform a Comparison Results. This section compares the energy and performance between AOA and other methods, including default (frequency with gear 0), static max frequency (static$_{max}$), static best frequency (static$_{best}$) and automatic overclocking (auto).

The detailed comparison result on platform A is shown in Table 3. The calculation of Energy 'savings' and Time 'improvement' in Table 3 is as follow:$savings = \left(1 - \frac{E_{method}}{E_{default}}\right) \times 100\%$; $improvement = \left(1 - \frac{T_{method}}{T_{default}}\right) \times 100\%$.

As for comparison with the maximum frequency, the method for determining the maximum frequency is to set the maximum frequency on the CPU side and GPU side respectively.

As for comparison with the static optimal frequency, the method for determining the static optimal frequency is: according to setting multiple sets of static CPU-GPU frequencies, a set of results with the best performance under the constraint of not increasing the total energy is obtained. According to experimental results, in static frequency modulation, when the frequencies on both CPU and

GPU are set to the maximum value, the performance improvement goal can be best met under the energy constraint.

Moreover, we compares the energy efficiency of AOA and automatic overclocking technology of CPU-GPU heterogeneous platforms. The method of automatic overclocking is: turn on Intel Turbo Boost Technology [16] on CPU, and turn on Nvidia GPU Boost Technology [22] on GPU.

Table 3. Comparison of AOA and other methods on platform A

AOA vs Others		alexnet	resnet	vgg	AVE
Energy (J) [savings]	Default	366068.37 [0.00%]	403126.65 [0.00%]	438716.4 [0.00%]	402637.14 [0.00%]
	static$_{max}$	339484.11 [7.26%]	430984.6 [−6.91%]	448662.53 [−2.27%]	406377.08 [−0.64%]
	static$_{best}$	339484.11 [7.26%]	403126.65 [0.00%]	431870.19 [1.56%]	391493.65 [2.94%]
	auto	358028.78 [2.20%]	419412.02 [−4.04%]	460568.18 [−4.98%]	412669.66 [−2.27%]
	AOA	326077.16 [10.92%]	403948.64 [−0.20%]	428037.8 [2.43%]	386021.20 [4.38%]
Time (s) [improvement]	Default	888 [0.00%]	908 [0.00%]	1002 [0.00%]	932.67 [0.00%]
	static$_{max}$	728 [18.02%]	822 [9.47%]	779 [22.26%]	776.33 [16.58%]
	static$_{best}$	728 [18.02%]	908 [0.00%]	832 [16.97%]	822.67 [11.66%]
	auto	723 [18.58%]	755 [16.85%]	779 [22.26%]	752.33 [19.23%]
	AOA	819 [7.77%]	833 [8.26%]	841 [16.07%]	831.00 [10.70%]
Freq (MHz) [f^{CPU},f^{GPU}]	Default	[2100,575]	[2100,575]	[2100,575]	–
	static$_{max}$	[2600,849]	[2600,849]	[2600,849]	–
	static$_{best}$	[2600,849]	[2100,575]	[2400,771]	–
	Auto	[Turbo Boost, GPU Boost]			
	AOA	Dynamic adjust			

Platform B Comparison Results. The detailed comparison result on platform B is shown in Table 4.

Table 4. Comparison of AOA and other methods on platform B

AOA vs Others		alexnet	resnet	vgg	AVE
Energy (J) [savings]	Default	140507.8 [0.00%]	172697.6 [0.00%]	329068 [0.00%]	214091.13 [0.00%]
	static$_{max}$	141491.1 [−0.70%]	186065.9 [−7.74%]	338351 [−2.82%]	221969.33 [−3.75%]
	auto	Not available			
	AOA	143051.8 [−1.81%]	179545.8 [−3.97%]	342651 [−4.13%]	221749.53 [−3.30%]
Time (s) [improvement]	default	606 [0.00%]	532 [0.00%]	1106 [0.00%]	748.00 [0.00%]
	static$_{max}$	588 [2.97%]	522 [1.88%]	1069 [3.35%]	726.33 [2.73%]
	auto	Not available			
	AOA	600 [0.99%]	533 [−0.19%]	1077 [2.62%]	736.66667 [1.14%]
Freq (MHz) [f^{CPU}, f^{GPU}]	Default	[2700,1350]			
	static$_{max}$	[3500,1455]			
	Auto	Not available			
	AOA	Dynamic adjust			

Similar to the experimental setup of platform A, we also compared the effects of default, auto, static$_{max}$ and AOA of CNNs on platform B. We got the best results when using vgg as a neural network model. Compared with the benchmark setting, AOA has a performance improvement of 2.62% but caused a 4.13% increase in energy consumption; on the alexnet network, AOA reduces energy by 1.81%, but it brings a 1% performance loss; AOA does not work well on resnet and does not cause performance improvement, but it increases energy consumption by 3.97%.

As a result, the energy efficiency optimization effect of the AOA algorithm on platform A is better than that on platform B. Besides CPU itself, there are many factors that determine the performance of the CPU, such as the speed of the hard disk, the size of the memory, and the data throughput of the interconnection network, are all important factors that restrict the performance of the processor. As a server platform, platform A has higher reliability and performance of matching facilities than platform B, which is a computer host. Therefore, the performance improvement from overbanding is better than that of platform B. And the CPU and GPU power consumption of platform A is lower than that of platform B, as shown in the Table 1, the impact of overclocking on power consumption is lower than that of platform B. Therefore, the energy efficiency optimization effect of platform B is not as good as that of platform A.

5 Discussion for Key Factors

This section considers the evaluation of the key factors involved in AOA (see Sect. 3.1). Consider the energy constraint and performance improvement effect of the AOA algorithm when the dynamic power upper limit factor k and the load imbalance factor W change.

5.1 Impact of Factor 1: The Power Upper Bound Must Be Dynamic, Not Static

This section shows that the dynamic variable power upper limit k is necessary to meet the constraint of no increase in total energy. The variable power upper limit factor defined by AOA is based on the initial value k_0, and changes within a certain range according to the variable power upper bound function F_k according to the U_t obtained during the sampling period. The result is shown in Table 5.

Table 5. Evaluation on Parameter k

Parameter k		alexnet	resnet	vgg
Energy(J) [savings]	Default	366068.4 [0.00%]	403126.6 [0.00%]	438716.4 [0.00%]
	k = 1.1	382594.0 [−4.51%]	437027.9 [−8.41%]	457669.4 [−4.32%]
	k = 1.15	362765.6 [0.90%]	436204.3 [−8.20%]	458734.5 [−4.56%]
	aoa	326077.2 [11.92%]	403948.6 [−0.20%]	428037.8 [2.44%]
Time(s) [improvement]	Default	888 [0.00%]	908 [0.00%]	1002 [0.00%]
	k = 1.1	862 [2.70%]	861 [4.19%]	825 [16.67%]
	k = 1.15	819 [2.93%]	833 [5.18%]	841 [17.66%]
	aoa	837 [7.77%]	835 [8.26%]	816 [16.07%]

5.2 Impact of Factor 2: Overclocking at both Ends of the CPU and GPU Must Be Coordinated, and Real-Time Load Balancing Must be Considered

The load imbalance factor W defined by AOA mainly affects the result of AOA frequency modulation through the function F_f. According to the formula 5 of the function F_f, when the maximum/small frequency changes, it will affect the frequency modulation direction of AOA for load imbalance. In the experiment, the default frequency is always used as $f_{min}^{CPU/GPU}$. When $f_{max}^{CPU/GPU}$ is set to a small value, $f_t^{CPU/GPU}$ is easier to reach the upper limit judgment condition, and then the direction of further frequency modulation is changed from increasing the local frequency value to decreasing Frequency value at the other end. The influence of the maximum frequency change is shown in Table 6.

Table 6. Evaluation on parameter f_{max}

Parameter $[f_{max}^{CPU}, f_{max}^{GPU}]$		alexnet	resnet	vgg
Energy(J) [savings]	Default	366068.4 [0.00%]	403126.6 [0.00%]	438716.4 [0.00%]
	AOA[2400,771]	395969.9 [−8.16%]	432592.5 [−7.31%]	454511.72 [−3.60%]
	AOA[2600,849]	326077.2 [11.02%]	403948.6 [−0.20]	428037.8 [2.44%]
Time(s) [improved]	Default	888 [0.00%]	908 [0.00%]	1002 [0.00%]
	AOA[2400,771]	898 [−1.13%]	867 [4.51%]	824 [17.76%]
	AOA[2600,849]	819 [7.77%]	833 [8.25%]	841 [16.07%]

6 Conclusion

Although GPUs have been used to accelerate various convolutional neural network algorithms with good performance, the demand for performance improvement is still continuously increasing. CPU/GPU overclocking brings opportunities for further performance improvement in the CPU-GPU heterogeneous platform. How to effectively constrain the total energy to remain roughly unchanged during the CPU/GPU overclocking is a key issue in designing adaptive overclocking algorithms.

This paper proposes an *Adaptive Overclocking Algorithm (AOA)* on the CPU-GPU heterogeneous platform to achieve the goal of performance improvement while the total energy remains roughly unchanged. AOA uses the function F_k to describe the variable power upper bound, which embeds real-time CPU and GPU usage information of the program, and introduces a load imbalance factor W to realize the CPU-GPU coordinated overclocking. Through the verification of a convolutional neural network program on two CPU-GPU heterogeneous platforms, AOA achieved an average of 10.7% performance improvement, while 4.4% energy saved on platform A. Also, we compared AOA with other three methods, including programs with static max frequency, programs with static best frequency and automatic overclocking. The comparison results show that AOA performs the best with regard to the goal of improving performance while constraining total energy.

References

1. OL. http://dag.wiee.rs/home-made/dstat/. Accessed Dec 2021
2. Linux kernel profiling with perf. OL. https://perf.wiki.kernel.org/index.php/Tutorial. Accessed Dec 2021
3. Nvidia system management interface. OL. https://developer.nvidia.com/nvidia-system-management-interface. Accessed Dec 2021
4. Abadi, M., et al.: Tensorflow: large-scale machine learning on heterogeneous distributed systems. arXiv abs/1603.04467 (2016)
5. Acun, B., Miller, P., Kale, L.V.: Variation among processors under turbo boost in HPC systems. In: Proceedings of the 2016 International Conference on Supercomputing. ICS 2016. Association for Computing Machinery, New York (2016). https://doi.org/10.1145/2925426.2926289, https://doi-org-s.nudtproxy.yitlink.com/10.1145/2925426.2926289
6. Chasapis, D., Moretó, M., Schulz, M., Rountree, B., Valero, M., Casas, M.: Power efficient job scheduling by predicting the impact of processor manufacturing variability. In: Proceedings of the ACM International Conference on Supercomputing. ICS 2019, pp. 296–307. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3330345.3330372, https://doi-org-s.nudtproxy.yitlink.com/10.1145/3330345.3330372
7. Chen, J., et al.: Analyzing time-dimension communication characterizations for representative scientific applications on supercomputer systems. Front. Comp. Sci. **13**(6), 1228–1242 (2019)
8. Chetlur, S., et al.: CUDNN: efficient primitives for deep learning. arXiv abs/1410.0759 (2014)
9. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255 (2009). https://doi.org/10.1109/CVPR.2009.5206848
10. Gad, E.A.: A work-stealing for dynamic workload balancing an CPU-GPU heterogeneous computing platforms. Thesis (2017). http://www.pqdtcn.com.nudtproxy.yitlink.com:80/thesisDetails/46952B07E4A7CC0D8C9AB6B408B99235
11. Gholkar, N., Mueller, F., Rountree, B.: Power tuning HPC jobs on power-constrained systems. In: 2016 International Conference on Parallel Architecture and Compilation Techniques (PACT), pp. 179–190 (2016). https://doi.org/10.1145/2967938.2967961
12. guassic: Text classification with CNN and RNN. https://github.com/gaussic/text-classification-cnn-rnn
13. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90

14. Inadomi, Y., et al.: Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC 2015. Association for Computing Machinery, New York (2015). https://doi.org/10.1145/2807591.2807638, https://doi-org-s.nudtproxy.yitlink.com/10.1145/2807591.2807638
15. Intel®: Overclocking: Maximizing your performance. OL. https://www.intel.com/content/www/us/en/gaming/overclocking-intel-processors.html. Accessed Dec 2021
16. Intel®: Release notes (xtu-7.5.3.3-releasenotes.pdf). OLhttps://downloadmirror.intel.com/29183/XTU-7.5.3.3-ReleaseNotes.pdf. Accessed Dec 2021
17. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. arXiv abs/1408.5093 (2014)
18. Kodama, Y., Odajima, T., Arima, E., Sato, M.: Evaluation of power management control on the supercomputer Fugaku. In: 2020 IEEE International Conference on Cluster Computing (CLUSTER), pp. 484–493 (2020). https://doi.org/10.1109/CLUSTER49012.2020.00069
19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017). https://doi.org/10.1145/3065386
20. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems, pp. 253–256 (2010). https://doi.org/10.1109/ISCAS.2010.5537907
21. Mittal, S., Vetter, J.S.: A survey of methods for analyzing and improving GPU energy efficiency. ACM Comput. Surv. **47**(2) (2014). https://doi.org/10.1145/2636342, https://doi.org/10.1145/2636342
22. NVIDIA®: GPU boost. OL. https://www.nvidia.com/en-gb/geforce/technologies/gpu-boost/. Accessed Dec 2021
23. PyTorch: Imagenet training in PyTorch. OL. https://github.com/pytorch/examples/tree/master/imagenet. Accessed Dec 2021
24. Ravichandran, D.S.M.R.M.E.C.S.: Processor Performance Enhancement Using Self-adaptive Clock Frequency, vol. 3, July 2010
25. Rodrigues, C.F., Riley, G., Luján, M.: Fine-grained energy profiling for deep convolutional neural networks on the Jetson tx1. In: 2017 IEEE International Symposium on Workload Characterization (IISWC), pp. 114–115 (2017)
26. Rouhani, B.D., Mirhoseini, A., Koushanfar, F.: Delight: adding energy dimension to deep neural networks. In: International Symposium on Low Power Electronics and Design (2016)
27. Russakovsky, O., et al.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vision **115**(3), 211–252 (2015). https://doi.org/10.1007/s11263-015-0816-y
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. Comput. Sci. (2014)
29. Sun, Y., et al.: Evaluating performance, power and energy of deep neural networks on CPUs and GPUs. In: Cai, Z., Li, J., Zhang, J. (eds.) NCTCS 2021. CCIS, vol. 1494, pp. 196–221. Springer, Singapore (2021). https://doi.org/10.1007/978-981-16-7443-3_12

30. Tang, Z., Wang, Y., Wang, Q., Chu, X.: The impact of GPU DVFs on the energy and performance of deep learning: an empirical study. In: Proceedings of the Tenth ACM International Conference on Future Energy Systems. e-Energy 2019, pp. 315–325. Association for Computing Machinery, New York (2019). https://doi.org/10.1145/3307772.3328315
31. Thomas, D., Shanmugasundaram, M.: A survey on different overclocking methods. In: 2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA), pp. 1588–1592 (2018). https://doi.org/10.1109/ICECA.2018.8474921
32. Wang, Y., et al.: E2-train: training state-of-the-art CNNs with over 80% energy savings. In: NeurIPS (2019)
33. Wu, F., Chen, J., Dong, Y., Zheng, W., Pan, X., Sun, Y.: Improve energy efficiency by processor overclocking and memory frequency scaling. In: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 960–967 (2018)
34. Wu, F., et al.: A holistic energy-efficient approach for a processor-memory system. Tsinghua Sci. Technol. **24**(4), 468–483 (2019). https://doi.org/10.26599/TST.2018.9020104
35. Yang, C., et al.: Adaptive optimization for petascale heterogeneous CPU/GPU computing. In: 2010 IEEE International Conference on Cluster Computing, pp. 19–28 (2010). https://doi.org/10.1109/CLUSTER.2010.12
36. Yang, F., Xu, Y., Meng, X., Gao, W., Mai, Q., Yang, C.: Nvidia tx2-based CPU, GPU coordinated frequency modulation energy-saving optimization method. Patent (2019). Patent Application Number: 201910360182.6. Publication Patent Number: CN 110308784 A
37. Yao, C., et al.: Evaluating and analyzing the energy efficiency of CNN inference on high-performance GPU. Concurrency and Computation: Practice and Experience (2020)
38. Zamani, H., Tripathy, D., Bhuyan, L., Chen, Z.: SAOU: safe adaptive overclocking and undervolting for energy-efficient GPU computing. In: Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design. ISLPED 2020, pp. 205–210. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3370748.3406553, https://doi.org/10.1145/3370748.3406553