Energy Optimization by Software Prefetching for Task Granularity in GPU-based Embedded Systems

Zhuowei Wang, Lianglun Cheng, Hao Wang, *Member, IEEE*, Wuqing Zhao, and Xiaoyu Song, *Senior Member, IEEE*

***Abstract*— Energy saving and optimization play an increasingly important role in industrial electronic systems. A heterogeneous embedded system is composed of a general-purpose central processing unit (CPU) with an enhanced module of graphics processing units (GPU). This paper explores the effective strategies of task granularity and software prefetching for energy optimization. We propose a novel energy optimization model for GPU-based embedded systems by harnessing a communication-based pipeline spatial and temporal relation. We analyze the characteristics of a multiple thread execution of parallel GPUs. We present an effective algorithm for the dynamic power optimization with the adaptively adjusted distance of software prefetching. The experimental results show that the dynamic energy consumption can be saved by 22.1% and 21.8% respectively under two prefetching strategies (register and shared memory) without loss of performance. We demonstrate the effectiveness of the proposed methods for energy saving and consumption reduction of performance driven computing in industrial scenarios.**

***Index Terms*—Low power optimization, heterogeneous embedded systems, communication-computing pipeline spatio-temporal diagram, task partition, software prefetching**

I. INTRODUCTION

IN the field of industrial manufacturing, heterogeneous embedded system is needed as a supporting condition for product design and R&D processes such as aerospace, automobile, ship and so on. Although a heterogeneous embedded system shows a higher peak computing speed and peak computing efficiency, the problem of massive power consumption remains. Excessive power consumption poses a severe challenge of reliability and heat dissipation for large-scale heterogeneous embedded systems in safety-critical industrial applications. Therefore, power consumption has become a crux concern at an unprecedentedly high level [1, 2].

The difference of speeds between a processor and an off-chip memory leads to the presence of a memory wall, which has always been one of main problems hindering improvements in computational efficiency [3-6]. At present, on-chip multi- and many-core processors have been developed, and a parallel system places a more onerous burden on memory registers, which aggravates the severity of problems concerning memory allocation. Thus, reducing or hiding memory latency is important when specifying a system architecture. Prefetching is a method of hiding memory latency by utilizing overlapped memory access and computation [7, 8]. Prefetching optimization aims to decrease bottlenecks in memory access and improve execution performance by extracting data into cache in advance and overlaps execution of computing and memory access functions on a processor. Prefetching optimization can be divided into hardware and software prefetching. Hardware prefetching aims to identify and predict the memory access mode of programs controlled by the prefetch engine so as to prefetch data automatically. Hardware prefetching is characterized by having no software overhead while it has low flexibility and pertinence. Software prefetching is illustrated as follows: programmers or compliers insert prefetching instructions at an appropriate location in their code and extract data into a cache (or register) in advance, thus avoiding computation arising from aborting due to a delay when waiting for memory access. Software prefetching is characterized by flexibility, efficiency, and pertinence while it leads to software and power overheads. In this paper, only software prefetching is taken into account.

The contributions of this paper are as follows. By considering the energy consumed by prefetching instructions, GPU processors, and memory access, as well as the static energy consumption of a system, an optimization model for the energy consumption of a GPU-based embedded system is established. An algorithm for optimizing dynamic energy consumption of homogeneous multi-GPU processors based on

Manuscript received May 21, 2019; revised July 17, 2019; accepted September 16, 2019. This work was supported by the school of computers, Guangdong University of Technology. (Corresponding authors: Zhuowei Wang.)

Zhuowei Wang is with the school of computers, Guangdong University of Technology, Guangzhou 510006, China, and also is with the school of computer science, Wuhan Donghu University, Wuhan 430074, China. (e-mail: wangzhuowei0710@163.com)

Lianglun Cheng is with the School of Computers, Guangdong University of Technology, Guangzhou 510006, China (e-mail: llcheng@gdut.edu.cn)

Hao Wang is with department of Computer Science, Norwegian University of Science and Technology, Gjøvik, Norway (e-mail: hawa@ntnu.no)

Wuqing Zhao is with CSG Digital Power Grid Research Institute Co., Ltd. (DGRI), Guangzhou, China (e-mail: zhaowq@csg.cn)

Xiaoyu Song is with the Department of Electrical and Computer Engineering, Portland State University, Portland, OR 97207, USA (e-mail: songx@pdx.edu).

adaptive adjustment of the distance of software prefetching is proposed.

The paper is organized as follows. Section 2 presents a review of existing works. Section 3 provides architecture of a GPU-based embedded system. Section 4 analyses the opportunities of task partition and software prefetching for energy optimization. Section 5 establishes a model for energy optimization of a heterogeneous embedded system. Based on the model, Sections 6 propose an algorithm for optimizing the dynamic energy consumption based on adaptive adjustment of the distance of software prefetching. Section 7 evaluates and analyzes the experimental results. Section 8 concludes the paper.

II. Related Work

Software prefetching has been investigated for a long time. Mowry et al. [9] are one of teams investigating optimization algorithms based on software prefetching. They propose an algorithm for inserting prefetching instructions. The algorithm only prefetches data likely to be subjected to cache failure: this avoids extra overheads due to unnecessary prefetching. By exploring software prefetching from the perspectives of compiling.

Very recently, it has been demonstrated that software prefetching-based optimization can effectively hide memory latency and improve the performance of programs, while it inevitably leads to increased power consumption [10, 11]. The main reason is that prefetching instructions increase the number of codes and prefetching takes advantage of the spatial parallelism of memory and processor. As a result, the energy consumed by the whole processor per unit time increases. Aiming at the influence of software perfecting based optimization on power consumption, Agarwal et al. [12] propose an energy optimization strategy such that, the performance gain obtained by software prefetching is converted into a reduction in energy consumption by using dynamic voltage and frequency scaling (DVFS) technology [13]. In this way, about 38% of energy overhead can be eliminated without performance loss. Through analysis, it can be seen that the method ignores the influence of voltage (or frequency) scaling on prefetching-based optimization. Others [14, 15] propose that controlling the overhead of software prefetching depends on determination of the prefetch distance while the optimal prefetch distance is co-determined by execution time of memory latency and a single iteration. After the voltage (or frequency) of a processor is scaled, only is the execution time of iteration affected, but the absolute latency of memory access is unchanged. Therefore, after reducing the frequency (or voltage), it is necessary to decrease the prefetch distance, making it more reasonable.

Task scheduling and dynamic voltage scaling are two main methods used for optimizing the performance or energy consumption of a system. Keqin et al. [16] investigated a combined optimization of the two methods aiming at a homogeneous parallel system. Through theoretical analysis, they pointed out that the energy optimization under a performance constraint and performance optimization under an energy constraint both can be treated as a general power sum problem. In further investigation [17], the author analyzed the problem related to energy optimization of parallel tasks in a parallel system and it is necessary to consider simultaneously the influences of three factors (involving system partition, task scheduling, and frequency scaling) on the energy consumption of a system. Goraczko et al. [18] propose a task partition method for energy optimization for heterogeneous multi-core processors. The method can optimize the energy overhead of processors under the constraint of satisfying real-time application by mapping tasks into heterogeneous multi-core processors and combining the technology of frequency scaling of processors.

The difference between this paper and other power optimization work is that software prefetching is introduced into the energy consumption optimization model, and energy consumption optimization is carried out by voltage frequency regulation and task partition.

III. Architecture of a GPU-based Embedded System

The architecture of a typical heterogeneous embedded system with multiple GPUs is shown in Figure 1: this contains a central processing unit (CPU) (host) processor and multiple GPU co-processors. Each CPU processor and GPU processor have their own memories: when programs run, CPU processors can send DMA orders and transfer data between the host memory and GPU memory by using a specialized DMA. Owing to the host memory being shared by various GPUs, the host processor is only allowed to transfer data to a GPU at a given time while various GPU processors can run independently. In this architecture, the program is divided into serial program segment and parallel program segment during the executive process. Serial program segments are executed on CPU and parallel program segments are executed on multiple GPUs. Since there is only one CPU, there is no problem of how to allocate tasks and how to adjust the dynamic voltage frequency. Therefore, under this architecture, we focus on the power modelling and optimization in multi-GPU environment.

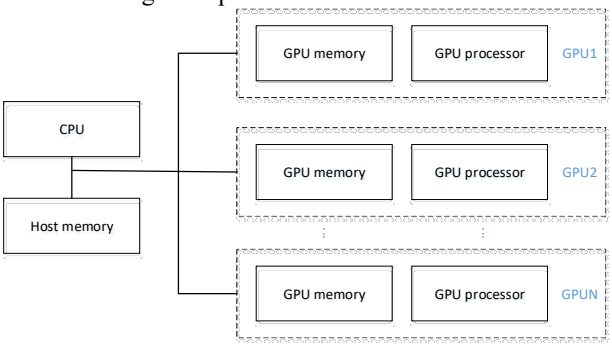


Fig. 1. Architecture of a typical GPU-based embedded system

IV. Task Partitioning and Software Prefetching for Energy Optimization

Three communication–computing pipeline spatio-temporal diagrams probably appear during execution of parallel programs in a homogeneous multi-GPU system.

We assume that the whole program contains n tasks m_i, m $= (m_1, m_2, \cdots, m_n)$ refers to the sequence of tasks arranged according to the program. For $1 \leq$ i $\leq$ n, Type(m_i) denotes the operation type of tasks m_i. The range of values is {C, P, T,

M} represent the kernel computing, data prefetching, data transferring, and memory latency, respectively. Time(Type(m_i)) is the time required for a task operation.

The perfect-overlap communication–computing pipeline spatio-temporal diagram is shown in Figure 2. The vertical axis refers to different GPUs and the middle part between two blue dashed lines denotes software prefetching and memory access conducted by the same GPU. The horizontal axis represents the execution time of programs where, $P(m_i)$ denotes prefetching, which is responsible for computing the address of data to be prefetched. For example, $P(m_1)$ refers to prefetching data required in the $C(m_{n+1})$ computing section and the prefetched data are accessed at point B. It can be seen from Figure 2(a) that, in the case of perfect overlap, the memory access brought about by the last prefetching task is completely overlapped with the computing operations in the processors. As this time there is,

$$\begin{aligned} Time\big(C(m_i)\big) + Time\big(P(m_i)\big) &= Time(T(m_i)) \times n \\ Time\big(C(m_i)\big) + Time\big(T(m_i)\big) + Time\big(P(m_i)\big) &= Time(M(m_i)) \times n \end{aligned} \quad (1)$$

When prefetching is executed early, data prefetched in a memory register are not immediately used as they access the memory. It is necessary to wait for some time; therefore, a period of idle memory appears. At this time there is,

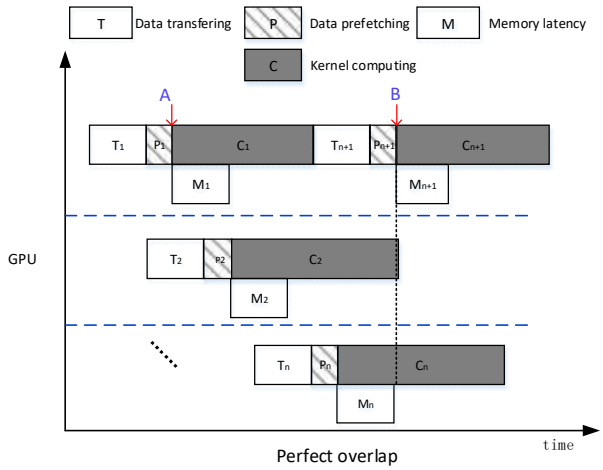
$$\begin{aligned} Time\big(C(m_i)\big) + Time\big(P(m_i)\big) &> Time(T(m_i)) \times n \\ Time\big(C(m_i)\big) &= Time(M(m_i)) \times n \end{aligned} \quad (2)$$

As depicted in Figure 2(b), the situation when the execution time of the processors plays a dominant role, due to prefetching operations being executed early, thus incurring an idle period of memory, is called *GPU-bound*. The occurrence of the idle period of the memory has an adverse influence on performance. In this case, from the perspective of energy, it is necessary to search for an appropriate task granularity to minimize the area of the rectangle enclosed by horizontal and vertical coordinates in the spatio-temporal diagram, thus decreasing static power consumption. Additionally, the frequency of the memory needs to be scaled to make the memory work at a low frequency. By doing so, on the condition of having no influence on the performance, this avoids accessing data in advance, therefore reducing dynamic energy consumption.

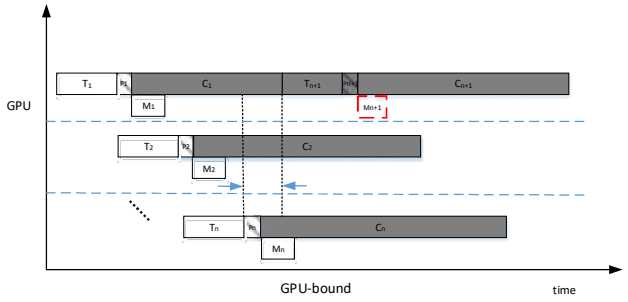
Under prefetching delay, the time of memory access is longer than the computing time of processors within the same stage. In this case, the idle period in computation occurs due to waiting to access data in memory cannot be completely removed. At this time there is,

$$\begin{aligned} Time\big(C(m_i)\big) + Time\big(P(m_i)\big) &< Time(T(m_i)) \times n \\ Time\big(C(m_i)\big) + Time\big(T(m_i)\big) + Time\big(P(m_i)\big) &> Time(M(m_i)) \times n \end{aligned} \quad (3)$$

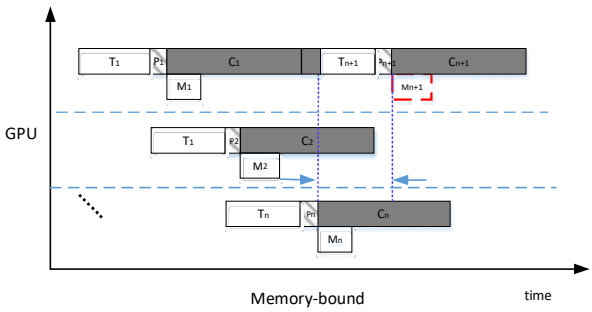
As depicted in Figure 2(c), the situation when the time of memory access plays a dominate role, due to prefetching operations being delayed, thus incurring an idle period on all processors, is called *memory-bound*. Under this circumstance, from the perspective of energy, it is feasible to determine an appropriate task granularity to minimize the area of the rectangle enclosed by horizontal and vertical coordinates in the spatio-temporal diagram, therefore lowering static power consumption. Additionally, the frequency of processors is adjusted to enable processors to work at a low frequency. In this way, while having no influence on the performance, the access of data in advance is avoided, thus decreasing dynamic energy consumption.



(a) Perfect-overlap communication-computing pipeline spatio-temporal diagram



(b) GPU-bound communication-computing pipeline spatio-temporal diagram



(c) Memory-bound communication-computing pipeline spatio-temporal diagram

Fig. 2. communication-computing pipeline spatio-temporal diagram

V. THE POWER MODEL OF HETEROGENEOUS SYSTEMS

In a CPU-GPU heterogeneous embedded system, k denotes a basic unit for tasks partition to be executed for a given kernel program. If the number of tasks assigned to a certain GPU is greater than the basic unit k, that is, when each Stream Multiple processors (SM) can be divided into more than one thread block, it can improve the computational intensive in SM to a certain extent, make more effective use of the performance of SIMD computing pipeline, and hide the delay caused by GPU memory access. When the number of thread blocks allocated to the GPU is not an integer multiple of the number of SMs, there will be load imbalance which makes some SMs idle. Therefore, in task

partitioning, we take the multiple $k \cdot r (r \geq 1)$ of the basic task partitioning unit k as the granularity. So $k \cdot r$ represents the granularity of task partitioning. A parallel program generally consists of multiple parallel loops. As data dependence between iterations does not exist in parallel loops, various loop iterations can be mapped into multiple processors for concurrent execution. In a heterogeneous embedded system with homogeneous multiple GPUs, the task $k \cdot r$ corresponds to a loop iteration in a parallel program and is assigned to a GPU processor for further execution by default.

When describing the problem related to energy optimization of a CPU-GPU heterogeneous embedded system under a performance constraint, the following parameters are involved:

$Data(k \cdot r)$ refers to the data size transferred to GPUs from a host processor (Host) corresponding to task k · r.

$T_t(Date(k \cdot r))$ represents the time taken transferring data size ($Date(k \cdot r)$) from the host processor (Host) to DRAM memory of GPUs.

$C_c(k \cdot r)$ refers to the clock cycle of processor computing required for completing task k · r by GPUs.

b denotes the number of cache blocks prefetched by a prefetching instruction.

N_b denotes the number of prefetching instructions in a loop iteration.

E_b denotes the energy overhead consumed when prefetching a cache block.

$\mathrm{C_p}$(Data(k·r)) denotes the clock cycle consumed during prefetching in each loop iteration.

$\mathrm{C_m}$(Data(k·r)) denotes the cycles of memory latency in an iteration caused by cache failure.

The optimization objective of energy and the performance constraint for the energy optimization problem of a multi-GPU system under a performance constraint are discussed below. The optimization objective is to minimize the total energy (E_t) consumed (including dynamic (E_d) and static (E_s) energy consumption), containing that in program execution in a prefetched loop section while ignoring energy consumed by the bus and clocks.

Dynamic energy consumption (E_d) mainly includes the energy (E_p) consumed when prefetching instructions calculate data addresses, energy (E_c) consumed during GPU computation, and energy (E_m) consumed by memory access.

It is supposed that the energy consumed in prefetching a cache block is E_b. The number of cache blocks prefetched by a prefetching instruction is b and the number of prefetching instructions in an iteration is N_b. In this case, the number of cache blocks to be prefetched in an iteration is expressed as $\mathrm{b} \cdot \mathrm{N}_b$. Therefore, the energy (E_p) consumed when prefetching instructions calculate data addresses within N_i iterations can be expressed as follows:

$$E_p = \mathrm{E}_b \cdot b \cdot \mathrm{N}_b \cdot N_i \tag{4}$$

For task $k \cdot r$, it is supposed that the power consumption of processors at the frequency f_c is $p_c(f_c)$ and the clock cycle consumed by computation of a processor within an iteration is $C_c(k \cdot r)$. Thus, the energy consumed by computation of the processor within N_i iterations is calculated as follows:

$$E_c = p_c(f_c) \cdot \frac{C_c(k \cdot r)}{f_c} \cdot N_i \tag{5}$$

For task $k \cdot r$, it is assumed that the power consumption of the memory at frequency $\boldsymbol{f_m}$ is $p_m(f_m)$ and the cycle of memory access within an iteration caused by cache failure is C_m(Data(k·r)). Under this circumstance, the energy consumed (E_m) by the memory within N_i iterations can be expressed as follows:

$$E_m = p_m(f_m) \cdot \frac{C_m(Date(\mathrm{k \cdot r}))}{f_m} \cdot N_i \tag{6}$$

Therefore, the dynamic energy consumption is:

$$E_d = \mathrm{E}_b \cdot b \cdot \mathrm{N}_b \cdot N_i + p_c(f_c) \cdot \frac{C_c(k \cdot r)}{f_c} \cdot N_i + p_m(f_m) \cdot \frac{C_m(Date(\mathrm{k \cdot r}))}{f_m} \cdot N_i \tag{7}$$

According to the reference [19], the dynamic power consumption P and the frequency f of electronic CMOS circuits satisfy $p \propto \alpha C f^3$(α is the switching activity factor, C is the switching capacitor) and therefore the power consumptions of the processors and memories can be separately expressed as follows:

$$p_c(f_c) = \alpha_1 \cdot C_1 \cdot f_c^3 \tag{8}$$

$$p_m(f_m) = \alpha_2 \cdot C_2 \cdot f_m^3 \tag{9}$$

By substituting the above two expressions into Formula (7), Formula (10) can be obtained:

$$E_{d=}\mathrm{E}_b \cdot b \cdot \mathrm{N}_b \cdot N_i + Q_1 f_c^2 C_c(k \cdot r) N_i + Q_2 f_m^2 C_m(Date(k \cdot r)) N_i \tag{10}$$

Where, $Q_1 = \alpha_1 \cdot C_1$ and $Q_2 = \alpha_2 \cdot C_2$.

As the main source of static power consumption, leakage current induced power consumption is generated when the circuit is stable, therefore, it can be assumed that the static power consumption of GPUs remains unchanged when programs run. The system contains M GPUs and the static power consumption is P_s. After being subjected to a certain task partition C, N (N ≤ M) GPUs take part in computing while the other GPUs are turned off or run at the lowest power consumption possible. In this case, the static power consumption can be ignored. The total execution time of programs is set to T and it is supposed that the static power consumption of GPUs remains unchanged during program execution. Thus, the total static power consumption of multiple GPUs can be expressed as follows:

$$E_s = N_G \cdot P_s \cdot T \tag{11}$$

Owing to P_s remaining unchanged during program execution, $E_s \propto N_G \cdot T$(N_G and T refer to the number of GPUs and total execution time of parallel programs, respectively) holds. This means that the static power consumption generated by multiple GPUs is positively proportional to the area of the rectangle enclosed by horizontal and vertical coordinates in the spatio-temporal diagram, therefore, the optimization objective of total energy of a CPU-GPU heterogeneous embedded system is as follows:

$$\min(\mathrm{E}_b \cdot b \cdot \mathrm{N}_b \cdot N_i + Q_1 f_c^2 C_c(k \cdot r) N_i + Q_2 f_m^2 C_m\big(Date(k \cdot r)\big) N_i + N_G \cdot P_s \cdot T \tag{12}$$

In terms of the energy optimization problem, performance is

the most important constraint condition. Performance refers to the execution time of parallel programs after being optimized based on an optimal task partition and software prefetching. The performance constraint should ensure that the total execution time of parallel programs does not rise to β. By analyzing the opportunities of task partition and software prefetching for energy optimization, it can be seen that the total execution time of parallel programs is related to the communication–computing spatio-temporal diagram. The diagram contains four basic operation types, involving communication for transferring data from the host processor to GPU DRAM, software prefetching for prefetching data from GPU DRAM to GPU cache, memory latency caused by software prefetching, and data calculation and processing. In the spatio-temporal diagram, it can be seen that, in a heterogeneous embedded system with homogeneous multiple GPUs, all memory latencies caused by software prefetching are hidden by the computing task(s) of GPUs, therefore, the ratio of the sum of the times consumed in GPU computations and software prefetching to the communication time (recorded as R) is a fundamental factor determining the distribution of the spatio-temporal diagram.

The total task load of parallel programs is expressed as F. For task $k \cdot r$ assigned to a certain GPU:

$$\mathrm{R}(k \cdot r) = \frac{C_c(k \cdot r)/f_c + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/f_c}{T_t(Date(k \cdot r))} \tag{13}$$

When considering the execution time of parallel programs, two conditions are shown, involving GPU-bound (Figure1 b) and memory-bound (Figure1 c) cases.

① If $\mathrm{R}(k \cdot r) \geq N_G$, various GPU processors are all in a full-load working state. Therefore, it can be considered that all data communication latencies are hidden by software prefetching and computing tasks in GPUs. Owing to the number of tasks assigned to each GPU being $\frac{F}{N_G \cdot k \cdot r}$, the total execution time of parallel programs is calculated as follows:

$$T_a(r) = N_G \cdot T_t\big(Date(k \cdot r)\big) + \frac{F}{N_G \cdot k \cdot r}\big(C_c(k \cdot r)/f_c + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/f_c\big) \tag{14}$$

② If $\mathrm{R}(k \cdot r) < N_G$, idling occurs in pipelines and data communication is taken as the key factor determining program execution times. It can be thought that all software prefetching and computing in GPUs are hidden by data communication, therefore, the total execution time of programs is expressed as follows:

$$T_b(r) = \frac{F}{k \cdot r} T_t\big(Date(k \cdot r)\big) + C_c(k \cdot r)/f_c + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/f_c \tag{15}$$

Thus, the execution time of parallel programs on the condition of having N_G GPUs satisfies the following piece-wise continuous functions:

$$\mathrm{T}(\mathrm{r}) = \begin{cases} T_a(r) & \mathrm{R}(k \cdot r) \geq N_G \\ T_b(r) & \mathrm{R}(k \cdot r) < N_G \end{cases} \tag{16}$$

When the degree of performance loss is allowed to be less than β, the following condition should be satisfied:

${f_c}^0$ represents the initial frequency of GPU processors and $\mathrm{T}^0(\mathrm{r})$ denotes the execution time of parallel programs when $f_c = {f_c}^0$. In this case, the following formula is acquired:

$$\mathrm{T}^0(\mathrm{r}) = \begin{cases} {T^0}_a(r) & \mathrm{R}(k \cdot r) \geq N_G \\ {T^0}_b(r) & \mathrm{R}(k \cdot r) < N_G \end{cases} \tag{17}$$

The goal of performance constraints is to require that the execution time of the optimized program should not exceed the original execution time (performance loss is expressed by parameter β) and minimize energy consumption. The optimization objective is the total energy consumption of the system (including dynamic and static energy consumption). Two constraints cond1 and cond2 ensure that the property of the program is not changed during the frequency regulation process (CPU-bound, Memory-bound). At the same time, the range of processor frequency and memory frequency is limited. f_c', f_c'', respectively, are the next and last period of processor frequency change. f_m', f_m'', respectively, are the next and last period of memory frequency change.

The energy optimization problem is described as follows.

$$\begin{gathered}
\min(\mathrm{E}_b \cdot b \cdot \mathrm{N}_b \cdot N_i + Q_1 {f_c}^2 C_c(k \cdot r) N_i + Q_2 {f_m}^2 C_m\big(Date(k \cdot r)\big) N_i + N_G \cdot P_s \cdot T(\mathrm{r})) \\
\mathrm{T}(\mathrm{r}) = \begin{cases} T_a(r) \leq (1+\beta){T^0}_a(r) & cond1 \\ T_b(r) \leq (1+\beta){T^0}_b(r) & \mathrm{cond2} \end{cases} \\
T_a(r) = N_G \cdot T_t\big(Date(k \cdot r)\big) + \frac{F}{N_G \cdot k \cdot r}\big(C_c(k \cdot r)/f_c + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/f_c\big) \\
T_b(r) = \frac{F}{k \cdot r} T_t\big(Date(k \cdot r)\big) + C_c(k \cdot r)/f_c + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/f_c \\
{T^0}_a(r) = N_G \cdot T_t\big(Date(k \cdot r)\big) + \frac{F}{N_G \cdot k \cdot r}\big(C_c(k \cdot r)/{f_c}^0 + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/{f_c}^0\big) \\
{T^0}_b(r) = \frac{F}{k \cdot r} T_t\big(Date(k \cdot r)\big) + C_c(k \cdot r)/{f_c}^0 + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/{f_c}^0 \\
cond1\text{: } \frac{C_c(k \cdot r)/f_c + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/f_c}{T_t(Date(k \cdot r))} \geq N_G \\
cond2\text{: } \frac{C_c(k \cdot r)/f_c + b \cdot \mathrm{N}_b \cdot \mathrm{C}_p(Date(k \cdot r))/f_c}{T_t(Date(k \cdot r))} < N_G \\
f_c' \leq f_c \leq f_c'' \\
f_m' \leq f_m \leq f_m''
\end{gathered} \tag{18}$$

VI. An Algorithm for Dynamic Energy Optimization Based on the Adaptively Adjusted Distance of Software Prefetching

The key to controlling the overhead of software prefetching is to determine the prefetch distance. For a loop structure, the prefetch distance denotes the number of loop iterations between prefetching instructions and true access. To hide the memory latency caused by prefetching, the time when prefetching instructions is completed must correspond to the moment of true access as far as possible during software prefetching. Therefore, the prefetch distance is co-determined by iterative delay and memory latency, so it can be expressed as follows:

$$\mathrm{PD} = \left\lceil \frac{AD}{RT} \right\rceil \tag{19}$$

Where, AD denotes the average memory latency and RT denotes the shortest possible execution time (containing time consumed by prefetching instructions) of each loop iteration. The purpose of rounding up is to guarantee that data have been prefetched before they are accessed. If the numerator and denominator of the fraction in Formula (19) are defined as wall-clock times, but not clock cycles, the formula can be written as follows:

$$\mathrm{PD} = \left\lceil \frac{AD}{C/f} \right\rceil \tag{20}$$

Algorithm: ECADP
Input: parallel programs being subjected to prefetching optimization
Output: PD′;
$f(\mathrm{i}) = f_0, f_m(\mathrm{i}) = f_{\mathrm{m0}}$, where, $1 \le \mathrm{i} \le \mathrm{N}$;
Execution is repeated at an interval of *2esE* threads until the program is completed {
if (parallel programs are memory-bound) then
    Executing *2esE* threads when prefetching is allowed;
    $\mathrm{T}_a(r)$ = execution time of parallel programs when executing *2esE* threads;
    Executing *2esE* threads when prefetching is not allowed;
    $\mathrm{T}_a(\mathrm{r})'$ = execution time of parallel programs when executing *2esE* threads;
    $\mathrm{gain} = \mathrm{T}_a(\mathrm{r})' - \mathrm{T}_a(\mathrm{r})/\mathrm{T}_a(\mathrm{r})'$;
    if $(\mathrm{gain} < 0)$ then
        //The time complexity of the first path is O(1)
        $f(\mathrm{i}) = f_0$;
        $f_m(\mathrm{i}) = f_{\mathrm{m0}}$;
        Prefetching is not allowed;
    else
        //The time complexity of the second path is O(n)
        for all i∈ [1, N] do
        $\mathrm{f(i)}' = f(\mathrm{i}) \cdot (\mathrm{C}_{cp}(\mathrm{i})/\mathrm{maxC}_{cnp}(\mathrm{i}))$;
        $V_c(i)' = V_c(i) \cdot (f(\mathrm{i})/\mathrm{f(i)}')$;
        $\alpha_C = \mathrm{f(i)}'/\mathrm{f}_0$;
        end for
        $\alpha = \alpha_c$;
    end
else (parallel programs are GPU-bound)
    Executing *2esE* threads when prefetching is allowed;
    $\mathrm{T}_b(r)$ = the execution time of parallel programs when executing *2esE* threads;
    Executing *2esE* threads when prefetching is not allowed;
    $\mathrm{T}_b(\mathrm{r})'$ = the execution time of parallel programs when executing *2esE* threads;
    $\mathrm{gain} = \mathrm{maxT}_b(\mathrm{r})' - \mathrm{maxT}_b(\mathrm{r})/\mathrm{maxT}_b(\mathrm{r})'$;
    if $(\mathrm{gain} < 0)$ then
        //The time complexity of the first path is O(1)
        $f(\mathrm{i}) = f_0$;
        $f_m(\mathrm{i}) = f_{\mathrm{m0}}$;
        Prefetching is not allowed;
    else
        //The time complexity of the second path is O(n)
        for all i∈ [1, N] do
        $\mathrm{f}_m(\mathrm{i})' = \mathrm{f}_m(\mathrm{i}) \cdot (\mathrm{C}_{mp}(\mathrm{i})/\mathrm{maxC}_{mnp}(\mathrm{i}))$;
        $V_m(i)' = V_m(i) \cdot (\mathrm{f}_m(\mathrm{i})/\mathrm{f}_m(\mathrm{i})')$;
        $\alpha_m = \mathrm{f}_m(\mathrm{i})'/\mathrm{f}_{m0}(i)$;
        end for
        $\alpha = \alpha_m$;
    end
end
C points are uniformly sampled as frequency scaling factors within the interval of [α, 1];
$minimise\ \mathrm{P}_X T_X \leftarrow select\ \alpha_X$;
if $(\mathrm{P}_X T_X < \mathrm{P}_0 T_0)$ then
    //The time complexity of the first path is O(1)
    $\alpha_{final} = \alpha_X$;
else
    //The time complexity of the second path is O(1)
    $f(\mathrm{i}) = f_0$;
    $f_m(\mathrm{i}) = f_{\mathrm{m0}}$;// it is not applicable to optimize the program based on software prefetching under a performance constraint
end
$\mathrm{PD}' = \alpha_{final} \cdot \mathrm{PD}$;
return PD′;

Fig. 3. An algorithm for optimizing dynamic energy consumption based on an adaptively adjusted distance of software prefetching

Where, C and f refer to the clock cycles within a single iteration and the working frequency of the processors, respectively. Generally, scaling the working frequency of processors cannot change the absolute latency of memory access. Thus, after reducing power, AD in Formula (20) remains unchanged while the number of clock cycles within a single iteration does not change with the working frequency, however, the delay in each clock cycle increases, thus, it can be seen that the prefetch distance shows an approximate, positively proportional, relationship with clock frequency:

$$\mathrm{PD}' = \alpha \cdot \mathrm{PD} \tag{21}$$

where, PD′ and α denote the prefetch distance after scaling and the frequency scaling factor, respectively.

For the architecture of GPUs, the burden on a register caused by prefetching may influence the degree of parallelism of programs, thus exerting a significant influence on performance. Therefore, reducing the prefetch distance generally means an increase in the degree of parallelism to improve performance. From this perspective, an algorithm for optimizing dynamic energy consumption based on adaptively adjusted distance of software prefetching (ECADP) is proposed and the pseudo-codes of the algorithm are as shown in Figure 3.

We assume that a heterogeneous embedded system contains E GPUs in which each GPU has e SMs. Additionally, the processors initially work at frequency f_0 while the initial memory frequency is f_{m0}. Simulation analysis is carried out on the original programs. It can be concluded that s thread blocks can synchronously work in SMs. To approach to true executive process, 2s thread blocks are assigned to each SM to make the SM always work in a full-load state during the simulation. According to the communication–computing pipeline spatio-temporal diagram in Section 3, $T_a(r)$ and $T_b(r)$ separately represent the execution time of parallel programs under GPU-bound and memory-bound conditions. For a given parallel program optimized by prefetching, the algorithm computes the data at the interval of 2esE threads and the execution of the 2esE threads is taken as a repetition period. During the execution of the 2esE threads, if the parallel program is in a memory-bound state, the initial 2esE threads are first executed on the condition of allowing prefetching. In this case, the execution time of parallel programs is $\mathrm{T}_a(r)$. Afterwards, the subsequent 2esE threads are executed without allowing prefetching. Under this circumstance, the execution time of parallel programs is $\mathrm{T}_a(\mathrm{r})'$. According to the execution time of 2esE threads when prefetching is, and is not, allowed, the performance gain can be calculated based on $\mathrm{gain} = \mathrm{T}_a(\mathrm{r})' - \mathrm{T}_a(\mathrm{r})/\mathrm{T}_a(\mathrm{r})'$. If the performance does not increase through prefetching, the programs are executed by applying the current voltage and frequency of the processors and prefetching is not allowed. When the performance is improved through prefetching, the performance gain can be converted into an energy saving by scaling the voltage and frequency of the processors. $\mathrm{C}_{cp}(\mathrm{i})$ refers to the number of execution cycles of the ith GPU when prefetching is allowed, and $\mathrm{maxC}_{cnp}(\mathrm{i})$ denotes the largest number of execution cycles of all GPUs when prefetching is not allowed. Through $\frac{\mathrm{C}_{cp}(\mathrm{i})}{\mathrm{maxC}_{cnp}(\mathrm{i})}$, the improvement ratio of performance is calculated. According to the relative increase in performance, the voltages and frequencies of each GPU processor core can be re-calculated to find the frequency scaling factor $(\alpha = \mathrm{f(i)}/\mathrm{f}_0)$ of each GPU

processor. In a similar way, if parallel programs are GPU-bound, the execution time of 2esE threads when prefetching is, and is not, allowed is expressed as $T_b(r)$ and $T_b(r)'$, respectively. When performance gain is generated through prefetching, $C_{mp}(i)$ is applied to represent the cycle of memory access of the ith GPU memory when allowing prefetching and $\max C_{mnp}(i)$ denotes the largest period cycle of memory latency of all GPU memories when prefetching is not allowed. Through $\frac{C_{mp}(i)}{\max C_{mnp}(i)}$, the performance improvement ratio is attained. Similarly, according to the performance improvement ration, the voltages and frequencies of each GPU memory can be re-calculated to find the frequency scaling factor ($\alpha = f_m(i)/f_{m0}(i)$) of each GPU memory. C points are uniformly sampled within the interval [α, 1] in steps 46-50 as frequency scaling factors. The frequency scaling factor α_X is selected to minimize the dynamic energy consumption ($P_X T_X$). If $P_X T_X < P_0 T_0$, α_X is taken as the final optimal frequency scaling factor (otherwise, the original frequency f_0/f_{m0} is used). In this case, it implies that it is inapplicable to optimize the program by using software prefetching under a performance constraint. After obtaining the optimal frequency scaling factor α_X, the access of prefetch distance is roughly determined at first according to the frequency scaling factor when determining the proper prefetch distance based on frequency during simulation in step 56 using Formula (20). Thereafter, slight scaling, with a small amplitude, is conducted to determine an appropriate prefetch distance, thus reducing dynamic energy consumption. We analyze the time complexity of the algorithm in terms of code nested layers. For the conditional judgment statements, the total time complexity is equal to the time complexity of the path with the greatest time complexity. In ECADP algorithm, there are three if-conditional judgment statements. In the first if-conditional judgment statement, the time complexity of the first path is $O(1)$ and that of the second path is $O(n)$, so the time complexity is $max\ (O(1), O(n))= O(n)$. Similarly, the time complexity of the second if-conditional judgment statement is $O(n)$. The time complexity of the third if-condition judgment statement is $O(1)$. So, the holistic time complexity of the whole algorithm is $max\ (O(n), O(n), O(1))$, that is $O(n)$.

VII. EXPERIMENTAL VERIFICATION

A. Experimental platform and test cases

We tested high performance computing platforms in industrial scenarios. Existing GPUs with few adjustable levels cannot completely support dynamic voltage (or frequency) scaling, which is not conducive to conducting theoretical research and verification of the low-power optimization of GPUs. Therefore, the simulator used for testing the power consumption of GPUs is used for experimental verification [20].

TABLE I PARAMETER SETTINGS OF THE SIMULATOR FOR GPU POWER CONSUMPTION

SM	8	Network	Crossbar
Warp size	32	SIMT width	32
Max. blocks per SM	8	Max threads per SM	1024
Core clock	325 MHz	L2 clock	650 MHz
Network clock	650 MHz	DRAM clock	800 MHz
Shared memory per SM	12 kB	Memory latency	450
L1 per SM		16 kB, 32bytes per block, 4-way	
L2 cache		2 MB, 32 bytes per block, 4-way	
Software		GPGPUSim 2.1.1b, NVCC 2.2, GCC 4.3	

The simulator for power consumption is realized by adding a Wattch model for power consumption into the GPGPUSim simulator to model the power consumption of various components (Shader Cores, L2 Cache, and the Memory Controller) in GPUs [21]. For an Interconnection Network, the modelling approach for power consumption used in PowerRed is applied [22]. For DRAM, the modelling is carried out by utilizing a published method [23]. For each component, the simulator counts the activities of each clock cycle in its clock domain and accumulates power consumption. Finally, the total power consumption of GPU is summed up. Because of the semiconductor technology adopted by the modern GPU is more mature and the characteristic coefficient is smaller than the set in Wattch model, it should be noted that the absolute power consumption given by the simulator is slightly higher than of the simulated target GPU (the general error is less than 10%). However, as a theoretical optimization method, this paper focuses on the relationship between power and performance changes of GPU after frequency reduction, rather than the absolute value of power consumption. Therefore, the absolute power error is acceptable. Parameters of the simulator for GPU power consumption are listed in Table 1.

TABLE II TEST CASE

Application	Data size	Thread block size	The number of thread block	Data/thread block	Data/r thread block
MM	2× 4MB	16×16	4096	128 kB	64 (1 + r) kB
DH	2 MB	256	1024	2 kB	2r kB
MV	8 MB	128	2048	64.5 kB	64r + 0.5 kB
LP	2 MB	16 × 16	2048	1.3 kB	1152r + 144 B
MT	4 MB	256	256	16 kB	16r kB
SQ	8×2MB	256	512	4 kB	4r kB
BS	2MB	256	256	8 kB	8r kB
FB	8MB	128	2048	64.5KB	64r + 0.5 kB

Eight typical applications (including BlackScholes (BS), dwtHaar1D (DH), fwtBatch1 (FB), MatrixMul (MM), Matrix-Vector (MV), Laplace (LP), MersenneTwister (MT), and SobolQRNG (SQ) from multiple cognate areas such as signal processing, finance, and scientific computation were used as the test cases. BS comes from the financial field. It implements the Black-Scholes model and calculates partial differential equations for financial prices. DH realizes the wavelet transform of signal. FB application comes from fast walsh transform. MT accomplishes Mersenne Twister pseudo-random number generation algorithm. SQ is the Sobol quasi-random number generation algorithm. MM, MV and LP come from the field of scientific computing. They are matrix multiplication, matrix vector multiplication and Laplace transformation. These applications are characterized by the fact that their loops are contained in a kernel function and the loop contains references to accessing global memory space. They satisfy the basic conditions for conducting software prefetching-based optimization. The specific data size of the test cases is listed in Table 2.

TABLE III FREQUENCY SCALING FACTORS AND OPTIMIZATION EFFECTS OF ENERGY CONSUMPTION

Application		Register				Shared memory			
		α		α_X	Energy consumption	α		α_X	Energy consumption
		α_c	α_m			α_c	α_m		
Memory-bound	MM	0.42		0.42	78.8%	0.46		0.46	86.2%
	DH	0.58		0.58	80.2%	-		-	100%
	MV	0.57		0.57	65.6%	0.43		0.43	66.3%
	LP	0.62		0.60	75.2%	-		-	100%
	MT	0.65		0.65	84.5%	-		-	100%
	SQ	-	-	-	100%	-	-	-	100%
GPU-bound	BS		0.35	0.35	84.5%		0.33	0.33	85%
	FB		0.23	0.2	76%		0.12	0.12	75%

B. Test results

Figure 4 shows the execution time and the ratio between power consumptions (after prefetching/before prefetching) of the applications on the condition of separately using register and shared memory as a prefetching buffer. When using shared memory as the buffer for software prefetching in LP and MT applications, the execution time of the applications increased compared with that before software prefetching, thus leading to poorer program performance. Under the two strategies, the performances after prefetching separately increase by 41% and 30% (on average) while the power consumption increased slightly due to software prefetching. The changes in simulated performances and energy consumptions of eight typical applications before, and after, optimization of dynamic energy consumption based on adaptively adjusted distance of software prefetching are analyzed on a GPU platform. Through analysis, it can be concluded that the adaptively adjusted distance of software prefetching is related to the frequency scaling factor. Therefore, the dynamic energy consumption can be optimized by scaling frequencies of processors and memories.

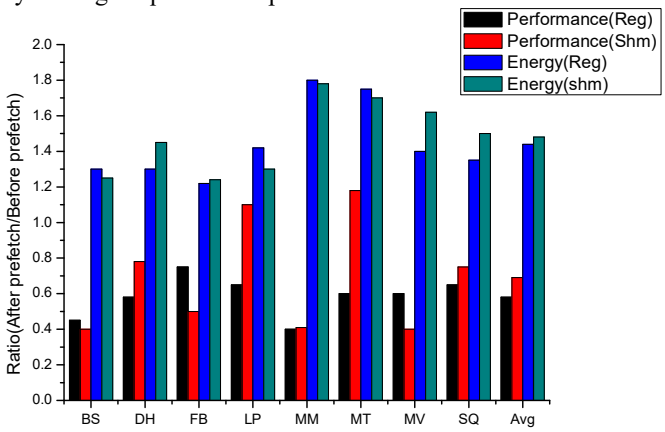


Fig. 4. Changes in simulated performance and energy consumption before, and after, prefetching

Figure 5 separately shows the performance and energy benefits when separately using the register and shared memory as a prefetching buffer. The prefetch distance is the optimal value calculated by using the algorithm for optimizing dynamic energy consumption based on the adaptively adjusted distance of software prefetching. As shown, when utilizing the register and shared memory as the prefetching buffer, the energy consumptions of the system can be separately reduced by 18% and 13% by applying the aforementioned algorithm under a performance constraint. It can be seen that the power consumptions of DH, LP, and MT, when using the shared memory as a prefetching buffer, are not optimized. The power consumptions of the application SQ under both strategies are not optimized. The main reason for this is that, when a program undergoing prefetching-based optimization is subjected to incremental frequency reduction, the moment at which the execution time increases to its original level is earlier than that when the energy consumption falls to its original level. This means that, when the execution time increases to the original time through frequency scaling, the power consumption is significantly greater than that seen in its original condition, thus failing to reduce energy consumption by decreasing frequency. Therefore, according to the optimization method, it is not applicable to utilize software prefetching for optimization and the application is still executed using the original programs at the original frequency.

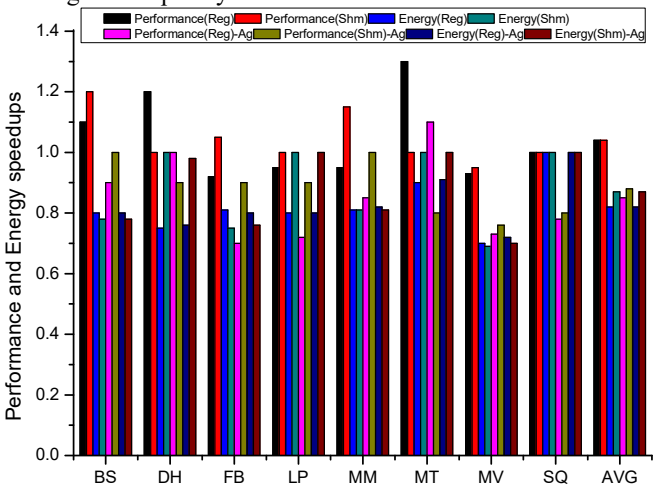


Fig. 5. Changes in simulated performance and energy consumptions before, and after, optimizing dynamic energy consumption based on the adaptively adjusted distance of software prefetching

Additionally, in a homogeneous multi-GPU system, there are three communication–computing pipeline spatio-temporal diagrams during software prefetching. According to different pipeline spatio-temporal diagrams, the performance bottlenecks of the application program can be further judged. By exploring the performance bottleneck, some parameters are calculated, including the frequency scaling factor α_X of processors (or memories) under optimal dynamic energy consumption and the frequency scaling factor α of processors (or memories) in performance-bound conditions under a time constraint. Based on the frequency scaling factor, an appropriate prefetch distance can be rapidly determined to further reduce dynamic energy consumption.

In the related work, we mentioned the work of Agarwal et al.

They proposed the energy optimization strategy of prefetching before DVFS. This method ignores the effect of DVFS on prefetching optimization. From the analysis of Section IV, we conclude that after DVFS, the prefetching distance should be reduced appropriately to make it more reasonable. It can also be seen form the experimental results of Fig.4, the register pressure brought by prefetching may affect the parallelism of the program and thus have a greater impact on the performance. Therefore, reducing prefetching distance often means increasing parallelism and improving performance.

In the energy consumption optimization method, Agarwal et al. adopted an on-line dynamic voltage regulation algorithm. The voltage was adjusted online according to the learning of instruction window. This method needs to ensure that the instruction window cannot be selected too small, in order to avoid the overhead of learning. For GPU program, the special structure of its program determines that it is not suitable to use learning method to dynamically adjust the voltage. First, the thread space of GPU is composed of a large number of thread blocks with isomorphic instructions, which also run on SM with isomorphic hardware. Because the number of thread blocks running at the same time on SM is limited, thread blocks occupy SM execution according to macro-batch and microscopic timesharing. From the time dimension of the whole program execution, SM execution is composed of a large number of similar computational processes, so there is no need to learn program behavior at the thread block level, only need to extract the behavior of single thread block by simulation. Secondly, the execution time of the single thread block itself is relatively short (the proportion of execution time of the total program is very small). If the instruction window is studied in the single thread block, the granularity is too fine. This will result in excessive learning overhead, and there is no practical significance. Therefore, 2esE thread blocks are simulated to extract the relationship between program performance, power consumption and software prefetching optimization.

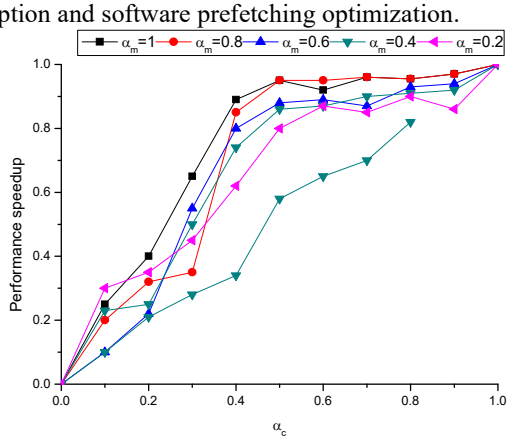


Fig. 6. (a) Performance improvements of application MM under different frequency scaling factors

In this paper, the proposed algorithm for dynamic energy optimization based on the adaptively adjusted distance of software prefetching compared with the energy optimization method (Abbreviated as Ag algorithm) propose by Agarwal et al. as shown in Figure 5. From the experimental results, we can see that the performance of our energy consumption optimization method is improved by 19% and 16% respectively, considering the appropriate reduction of prefetch distance compared with Ag algorithm, when the energy consumption is basically the same. So, the method proposed in this paper is more effective.

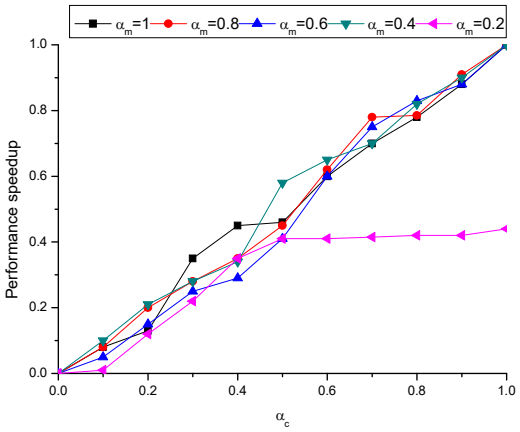


Fig. 6. (b) Performance improvements of application BS under different frequency scaling factors

Table 3 lists the optimal frequency scaling factors α_X, and frequency scaling factors α at the performance-bound state and optimization effects of energy consumption of eight applications. According to the communication–computing pipeline spatio-temporal diagram, the eight applications can be approximately classified into two types. The applications MM, DH, MV, LP, and MT are constrained by the memory of GPUs and therefore communication overhead is considered to be the performance bottleneck (memory-bound) of that type of application. Under this circumstance, the energy consumption can be reduced by decreasing the frequency α_c of the processors. Applications BS and FB are restricted by the processors in the GPUs and therefore the computing time of processors is regarded as the performance bottleneck (GPU-bound) of that type of application. In this case, the energy consumption can be reduced by reducing the frequency of the memories. It can be seen from Figure 4 that prefetching using shared memory is inapplicable for DH, LP, and MT applications. For an SQ program, it is not suitable to carry out prefetching under the two strategies, therefore, the power consumptions under these conditions are not optimized, with an energy saving of 0%. In the other applications, under most circumstances, the optimal frequency scaling factor α_X for energy consumption is consistent with the frequency scaling factor α (α_c/α_m) at the performance bound; however, for the three applications FB, LP, and MM, the condition $\alpha_X < \alpha$ arises. The main reason is that there is also non-negligible static energy consumption in the system, apart from its dynamic energy consumption. When improving the frequency to some extent in the vicinity of α, the reduction in static energy consumption is more significant than the increase in dynamic energy consumption. According to the frequency regulation factor α_X, the parameters of core clock and DRAM clock are adjusted in GPGPUSim power simulator. In GPGPUSim, each

power consumption data is recorded in a power_result_type structure, and the total power consumption data is output through double total_power. According to the ration of the total energy consumption after to before frequency regulation, the percentage of energy consumption optimization in Table 3 is obtained. Taking the average value, under the two prefetching strategies, energy consumptions can be separately reduced by 22.1% and 22.8% through frequency scaling.

To verify the effectiveness of the optimal frequency scaling factor in the present study, the applications MM and BS are selected and verified on the true Quadro FX 5600 platform. The two applications separately represent the two types of test cases, i.e., memory-bound and GPU-bound. Figure 6 shows the performance improvements in applications MM and BS obtained under different frequency scaling factors (α_c and α_m). It can be seen from Figure 6(a) that an inflection point occurs in the frequency scaling factor α_c plot between 0.4 and 0.45 for processors when the performance of MM rapidly declines. The reason for this is that MM is a computation-intensive application. When the processor works at a high frequency, the bottleneck during program execution frequency of the processor decreases to a certain extent, the bottleneck during the execution of application MM is found in the processors but not in memory. Under this circumstance, the performance of the application decreases if the frequency of the processor continues to be lowered. Therefore, the optimal frequency scaling factor of the MM application program is supposed to occur at the inflection point. As seen from Table 2, the optimal frequency scaling factors of GPU processors in MM application are theoretically calculated as 0.42 and 0.46 by using the algorithm for optimizing dynamic energy consumption based on the adaptively adjusted distance of software prefetching. The results are attained when the register and the shared memory are applied for prefetching-based optimization. The results show an insignificant discrepancy with the optimal frequency scaling factor $0.40 \leq \alpha_c \leq 0.45$ for processors tested on a true platform. In Figure 6(b), it can be seen that, for application BS, the performance exhibits a linear relationship with the frequency of processors for most α_m. It is because application BS is memory-intensive. In this case, the bottleneck of program operation is attributed to computation and therefore reducing the frequency of GPU processors certainly decreases the performance. Thus, under this circumstance, it is necessary to guarantee the performance and lower the energy consumption by decreasing the memory frequency. As shown in the figure, on the condition that α_m is low enough, that is, $\alpha_m = 0.2$, the bottleneck during program operation is transformed. In this context, reducing the frequency of GPU processors by a small amount cannot significantly influence program performance, therefore, it can be judged that the optimal frequency scaling factor for application BS is $\alpha_m = 0.2$, which differs by about 10%, from the theoretical optimal frequency scaling factors (0.35 and 0.33): this further validates the effectivenesses of the model for energy consumption and the algorithm for optimizing dynamic energy consumption.

VIII. Conclusion and Future Work

The opportunities of task partition and software prefetching for energy optimization of a CPU-GPU heterogeneous embedded system are analyzed through the communication–computing pipeline spatio-temporal diagram. Furthermore, a model for energy optimization of a homogeneous multi-GPU system is established. Based on the model, an algorithm for optimizing dynamic energy consumption of a homogeneous multi-GU architecture based on the adaptively adjusted distance of software prefetching is proposed. The algorithm is used for energy optimization of parallel programs. The test result showed that, under the two prefetching strategies (register and shared memory), the dynamic energy consumptions separately decreased by 22.1% and 21.8% (at most) on the premise of maintaining the program performance through frequency scaling of processors and memories.

Parallel programs exhibit various problems (such as load imbalance and data dependence) during actual executions, so it is difficult to establish models capable of exploring the energy optimization of parallel programs. For this reason, future work will focus on the investigation of the characteristics of program parallelism in order to conduct energy analysis and optimization thereof.

Acknowledgment

This work was sponsored in part by National Natural Science Foundation of China (grant number 61672168, 61672172, U1801263, U1701262, 61300029, 61772143, 61803093), National High-Resolution Earth Observation Major Project (83-Y40G33-9001-18/20), Guangdong Provincial Key Laboratory of Cyber-Physical System (2016B030301008)

References

[1] Y. L. Zhu, D. Pan, Z. W. Li, H. Liu, Y. Zhao, Z. Y. Lu, and Z. Y. Sun. "Employing multi-GPU power for molecular dynamics simulation: an extension of GALAMOST," *Molecular Physics*, vol.116, no.7-8, pp.1-13, 2018.

[2] L. Chao, Y. S. Fei, L. W. Zhang, A.A. Ding, L. Pei, S. Mukherjee, and D. Kaeli. "Power analysis attack of an AES GPU implementation," *Journal of Hardware & Systems Security*, vol.2, no.1, pp.69-82, 2018.

[3] Y. Yi, X. Ping, J. F. Kong, and H. Y, Zhou. "A GPGPU compiler for memory optimization and parallelism management," *ACM SIGPLAN Notices*, vol. 45, no.86-97, pp.86-97, 2010.

[4] H. M. Roudsari, A. Jalilian, S. Jamali. "Flexible fractional compensating mode for railway static power conditioner in V/v traction power supply system," *IEEE Transactions on Industrial Electronics*, vol. pp, no.99, pp.1-1, 2018.

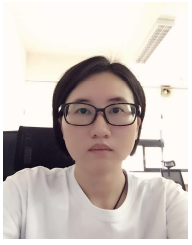
[5] Z. W. Wang, L. L. Cheng, W. Q. Zhao and N. X. Xiong. "An architecture-level graphics processing unit energy model," *Concurrency and Computation: Practice and Experience*, vol. 28, no.10, pp.2795-2810, 2016.

[6] C. Cen, K. Li, A. Ouyang, T. Zhuo, and K. Li. "GFlink: An in-memory computing architecture on heterogeneous CPU-GPU clusters for big data," *International Conference on Parallel Processing*, pp.1-1, 2016.

[7] Z. W. Wang, W. Q. Zhao, H. Wang, and L. L. Cheng. "Three-level performance optimization for heterogeneous embedded systems based on software prefetching under power constraints," *Future Generation Computer Systems*, vol.86, pp.51-58, 2018.

[8] S. M. Cheng, P. B. Gibbons, T. C. Mowry. "Improving index performance through prefetching," *Proceedings of ACM SIGMOD Conf*, vol.30, no.2, pp.235-246, 2001.

[9] T. Mowry, A. Gupta. "Tolerating latency through software-controlled prefetching in shared-memory multiprocessors," *Journal of Parallel and Distributed Computing*, vol.12, no.2, pp.87-106, 1991.
[10] M. Payami, E. Azarkhish, I. Loi, and L. Benini. "A hybrid instruction prefetching mechanism for ultra low-power multi-core clusters," *IEEE Embedded Systems Letters*. vol. PP, no. 99, pp.1-1, 2017.
[11] Y. Zhou, S. Taneja, C. W. Zhang, and X. Qin. "GreenDB: energy-efficient prefetching and caching in database Clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no.99, pp.1-1, 2018.
[12] D. N. Agarwal, S. N. Pamnani, Q. Gang, and D. Yeung. "Transferring performance gain from software prefetching to energy reduction," *IEEE International Symposium on Circuits & Systems*, pp.1-4, 2004.
[13] J. Chen, Y. Y. Ke. "A dynamic power management mechanism for embedded system with micro-kernel operating system," *Applied Mechanics and Materials*, vol.325-326, pp.916-921, 2013.
[14] A. C. Klaiber, H. M. Levy. "An architecture for software-controlled data prefetching," *ACM SIGARCH Computer Architecture News*, vol. 19, no. 3, pp. 43-53,1991.
[15] T. C. Mowry, M. S. Lam, A. Gupta. "Design and evaluation of a compiler algorithm for prefetching," *International Conference on Architectural Support for Programming languages and Operating Systems*, New York, NY, USA, pp.62-73,1992.
[16] K. Li. "Performance analysis of power-aware task scheduling algorithms on multiprocessor computers with dynamic voltage and speed," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1484-1497, 2008.
[17] K. Li. "Energy efficient scheduling of parallel tasks on multiprocessor computers," *Journal of Supercomputing*, vol. 60, no. 2, pp. 223-247, 2012.
[18] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, and F. Zhao. "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," *In Proceedings of the 45th annual design automation conference*. New York, NY, USAM, pp.191-196, 2008.
[19] T. D. Burd, R. W. Brodersen. "Energy efficient CMOS microprocessor design," *Twenty-Eighth Hawaii International Conference on System Sciences*, pp. 288-297,1995.
[20] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. "Analyzing CUDA workloads using a detailed GPU simulator," *IEEE Intl Symp Performance Analysis of Systems & Software*, pp.163-174, 2009.
[21] D. Brooks, V. Tiwari, M. Martonosi. "Wattch: A framework for architectural-Level power analysis and optimizations" *IEEE International Symposium on Computer Architecture*, pp. 83-94, 2000.
[22] K. Ramaniz, A. Ibrahim, S. Dan, "PowerRed: a flexible modeling framework for power efficiency exploration in GPUs," *Worskshop on Gpgpu*, pp.1-8, 2007.
[23] Š.Tajana, B. Luca, D. M. Govanni. "Cycle-accurate simulation of energy consumption in embedded systems". *In Proceedings of the 36th annual ACM/IEEE Design Automation Conference*. New York, NY, USA, pp. 867–872, 1999.



Zhuowei Wang received the Ph.D. degree in computer system architecture form Wuhan University, Wuhan, China, in 2012. She is now associate professor of the institute of computers at Guangdong University of Technology. Her research interests focus on high performance computing, low power optimization, distributed systems and etc.

Lianglun Cheng is now a professor of institute of computer in Guangdong University of Technology. He has a master degree in automation from Huazhong University of Science and Technology. He receives the Ph.D. degree in machinery manufacturing and automation, from Changchun Institute of Optical Precision Machinery and Physics, Chinese Academy of Sciences. His research interests focus on IOT, CPS and sensor networks etc.

Hao Wang (M'07) is an associate professor in the Department of Computer Science in Norwegian University of Science & Technology, Norway. He has a Ph.D. degree and a B.Eng. degree, both in computer science and engineering, from South China University of Technology. His research interests include big data analytics, industrial internet of things, high performance computing, safety-critical systems, and communication security. He has published 100+ papers in reputable international journals and conferences. He served as a TPC co-chair for IEEE DataCom 2015, IEEE CIT 2017, ES 2017, a senior TPC member for CIKM 2019, and reviewers for journals such as IEEE TKDE, TII, TBD, TETC, T-IFS, IoTJ, TCSS, and ACM TOMM, TIST. He is a member of IEEE IES Technical Committee on Industrial Informatics.

Wuqing Zhao received the Ph.D. degree in Wuhan University. He is now senior engineer in CSG Digital Power Grid Research Institute Co., Ltd. (DGRI). He is especially interested in data grid, cloud computing and etc.

Xiaoyu Song (M'99–SM'04) received the Ph.D. degree from the University of Pisa, Italy, in 1991. From 1992 to 1998, he was on the faculty at the University of Montreal, Canada. He joined the Department of Electrical and Computer Engineering at Portland State University in 1998, where he is now a professor. He was an editor of IEEE Transactions on VLSI Systems and IEEE Transactions on Circuits and Systems. He was awarded an Intel Faculty Fellowship from 2000 to 2005. His research interests include formal methods, design automation, embedded systems and emerging technologies.