Three-level performance optimization for heterogeneous systems based on software prefetching under power constraints

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

[1] *School of Computer, Guangdong University of Technology, Guangzhou 510006, China*
[2]*Dingxin Information Technology Co.,Ltd, Guangzhou,510006, China*
[3]*Department of ICT and Natural Sciences, Norwegian University of Science and Technology, Norway*

**Corresponding author: hawa@ntnu.no*

Abstract: High power consumption has become one of the critical problems restricting the development of high-performance computers. Recently, there are numerous studies on optimizing the execution performance while satisfying the power constraint in recent years. However, these methods mainly focus on homogeneous systems without considering the power or speed difference of heterogeneous processors, so it is difficult to apply these methods in the heterogeneous systems with an accelerator. In this paper, by abstracting the current execution model of a heterogeneous system, we propose a new framework for managing the system power consumption with a three-level power control mechanism. The three levels from top to bottom are: system-level power controller (SPC), group-level power controller (GPC) and unit-level power controller (UPC). The study establishes a power management method for software prefetch in UPC to scale frequency and voltage of programs, select the optimal prefetch distance and guide optimization process to satisfy the constraint boundary according to power constraints. The strategy for dividing power based on key threads is put forward in GPC to preferentially allocate power to threads in key paths. In SPC, a method for evaluating the performance of heterogeneous processing engines is designed for dividing power in order to improve the overall execution performance of the system while sustaining the fairness between concurrent applications. Finally, the proposed framework is verified on a central processing unit (CPU)-graphics processing unit (GPU) heterogeneous system.

Keywords: *High-performance Computing Systems; Heterogeneous system; Performance optimization; Software prefetch; Energy constraints*

1. Introduction

Nowadays, green computing has become one of the hottest topics in high performance computing fields. It is an essential way of lowering the system power consumption and improving system energy efficiency to scale up supercomputers [1-4]. Heterogeneous parallel systems have become one important trend for high performance computing systems (HPCS).

As heterogeneous parallel system integrates several different types of processors, the power technologies are different from those for homogeneous systems. The research on low power optimization of heterogeneous system is still in its infancy. Due to the great differences in architecture, general microprocessors and accelerators have different execution characteristics under different system configurations and problem scales, it is the research focus of heterogeneous system power optimization to effectively combine the efficiency advantages of different computing resources and fully employ the heterogeneous parallel processing capability [5-6].

The research on the energy-consumption optimization problem can be categorized in two approaches: the first one is to focus on the average energy consumed by the system, that is, system designers try to minimize the cost of the average amount of energy consumed while satisfying the constraints placed for application performance. The second approach is to focus on the upper limit of the energy consumption, that is, system designers try to maximize the system execution performance without exceeding the given energy consumption limit. Energy-consumption optimization traditionally follows the first approach. As the scale of these HPCS has continued to grow, energy consumption has gradually become one of the most important constraints affecting their design, operation, and management [7-8].

Recently, more studies follow the second approach, i.e., optimizing the system performance within a given power budget, called *peak power management*. However, most existing solutions are designed for homogeneous system without considering the differences in power consumption and processing speed between heterogeneous processors, therefore they could not be adapted for accelerator-based heterogeneous parallel system effectively.

In order to address these challenges, we made the following contributions in this paper:

(1) A hierarchical power management strategy is proposed. By abstracting current execution model of a heterogeneous system, we propose a management framework for system power combining a three-level power control mechanism.

(2) In unit-level power control (UPC) for HPCSs, a power control method for software prefetch is proposed. First, with the source program analyzed, the method makes the prefetch according to the optimal prefetch distance to optimize the performance. Then the frequency and voltage of optimized programs are scaled based on the constraint condition of the optimization objective while the prefetch distance is adjusted accordingly. The process is repeated until the optimization objective reaches to the boundary of the constraint condition. Under the constraint condition, simulation sampling is conducted to retrieve the performances and power consumption of all sampling sites. Based on this information, the change curves of performances and power are fitted to obtain the prefetch distance and the coefficients of frequency and voltage scaling under optimal performance.

(3) In group-level power control (GPC) for HPCSs, a strategy for allocating power based on key threads is proposed to preferentially apportion power to threads in key paths.

(4) In system-level power control (SPC), a method for evaluating the performance of a heterogeneous processing group (HPG) is proposed. Applying it as the basis for power distribution, we attempt to improve the overall execution performance of a system while sustaining the fairness of concurrent applications.

Section 2 presents a review of existing works; Section 3 introduces the hierarchical power management framework for HPCSs; Sections 4, 5, and 6 respectively present the methods for unit-, group- and system-level power controls for software prefetch in HPCSs; Section 7 describes the experiments and analyzes the results; Section 8 concludes the paper.

2. Related work

Constantly scaling-up HPCSs are consuming dramatically more power, which presents great challenges to the power supply and cooling systems. Therefore, power consumption is not only an important goal of system optimization but also has become one of the critical constraints on system design [9-10].

In recent years, there have been numerous studies focusing on how to optimize the execution performance of a system while satisfying the maximum power constraint. Wang and Chen [11] proposed a cluster-level feedback power control method which distributes the total power according to the utilization ratios of processors in different servers with a power consumption upper limit. Raghaverdra et al. [12] designed a power management framework for data centers combining multi-level power management technologies. Additionally, the maximum power management method at microprocessor level was also investigated. A MaxBIPS (billions of instructions per second) strategy based on prediction was proposed in [13]. By searching the frequency levels supported by different processors, the whole throughput is maximized while satisfying the power constraints of chips. Meng et al. [14] established a hierarchical management method for maximum power. The power is preferentially allocated to the processor cores with a large ratio of performance gain to power cost by using the optimal promotion strategy. The aforementioned two methods assume that the application programs runing in different processor cores are mutually independent, and therefore they are not applicable to multi-thread parallel application environments. Ma et al. [15] proposed a hierarchical management method based on control theory which focuses on the influence of load imbalance on different threads within an application on power distribution apart from considering the power distributions between concurrent applications. These exiting methods for managing power mostly focus on homogeneous parallel systems without considering the speed or power difference between different computation units. Consequently, it is difficult to apply these methods to heterogeneous systems.

The aforementioned studies on performance optimization under energy constraints mainly consider energy consumptions of processors [16-19] but not the energy problem of off-chip memories. It has been indicated that it is necessary to take off-chip memories into consideration in energy optimization [20]. There is a large performance difference between the processors and off-chip memories. With constant growth of dominant frequency of a processor, the difference is still significant in spite that the memory access latency is also steadily reducing. The “memory wall” is always a crucial factor restricting the performance improvement of programs [21]. To alleviate the memory wall problem, numerous optimization methods for reducing or hiding the memory access latency are proposed [22-23]. *Software prefetch* is an effective method for hiding memory access latency [23]. The software prefetch optimization overlaps operations of the processor computation and memory access and partly hides the memory access latency by fetching data to Cache in advance. This method significantly improves performances, it however rapidly increases the power consumption [24]. On one hand, adding prefetched instructions increases the amount of codes. On the other hand, the great overlap between processor computation and memory access significantly improves the utilization rate of functional units within unit time so that the energy consumption within unit time greatly increases. In this context, it is of importance to improve the software prefetch performance effectively without increasing power.

To address the overall system performance problem under the energy consumption constraints, this paper proposes a management framework for system power combining a three-level power control mechanism by abstracting current execution model of a heterogeneous system. Respectively at the unit-level, group-level, and system-level power control, a power control method for software prefetch, a strategy for allocating power based on key threads, and a method for evaluating the performance of a heterogeneous processing group are proposed. In this paper, we aim to explore the advantages of the heterogeneous parallel processing within the constrained power budget, and optimize the system performance.

3. The hierarchical power management framework

A heterogeneous parallel system includes multiple computation resources involving central processing units (CPUs), graphics processing units (GPUs), field programmable gate arrays (FPGAs) and monolithic integrated circuits (MICs). Generally, accelerators are only used to execute specific computing tasks, so they are not equipped with complete task management and scheduling mechanism. Thus, the accelerators work under the control of a general microprocessor. The computation module consisting of a host processor and its corresponding accelerator is called a *heterogeneous processing unit* (HPU). In general, all threads of multi-thread parallel programs are mapped to the host processor which loads the specific computation processes within threads to the accelerator for execution. Therefore, different threads within the multi-thread parallel programs are mapped to multiple HPUs. A *heterogeneous processing group* (HPG) consists of multiple HPUs running the same application program, multiple of which make up a *heterogeneous processing system* (HPS). The architecture of a typical HPS is shown in Figure 1.

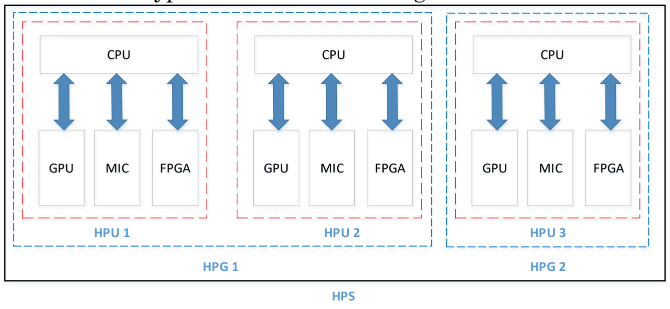

Fig.1. The architecture of a typical HPS

Power consumption of a processor is mainly composed of dynamic and static power consumptions. Dynamic power consumption is related to the operating frequency of a processor, which can be reduced by using dynamic voltage and frequency scaling (DVFS) technology. The static power of a processor is associated with the operating voltage and temperature of chips and always accompanies the operation of a processor. Therefore, the distributable power range is the difference between power constraints and static power of a system.

For SPC, according to contributions of different application programs to the whole performance of a system, the power distributed to the *ith* SPC is denoted as P_{SPC_i} considering the power of a horizontal division system of concurrent applications. SPC does not consider the specific execution process of an application program but executes the power control process in a fixed period using the transparent application control mode.

A GPC is in charge of distributing the power P_{SPC_i} to multiple parallel processing groups (P_{GPC_i}). Different from a SPC, a GPC can select a more effective control strategy according to the different characteristics of programs. In general, OpenMP-like parallel programs mainly consist of multiple serial or parallel computing segments, so the entry of computing segments is taken as the calling point of the GPC. The key to power control of a HPG is to observe the loads of different HPUs or differences of execution time during parallel execution, so as to effectively distribute power.

UPC takes charge of distributing the group power P_{GPC_i} to heterogeneous computing units P_{CPC_i} in a group. Because the heterogeneous processors in a HPU exhibits different execution rates and power consumptions, how to optimize the overall performance of a HPU while meeting the power constraint is the key problem. This paper proposes an optimization method based on software prefetch to reduce the execution time of programs while satisfying power constraints. The overall idea is as follows: the performance improves by using optimization method based on software prefetch while the dynamic power increases. In this context, the frequency of processor reduces according to the constraint conditions to achieve the optimal performance under the power constraint.

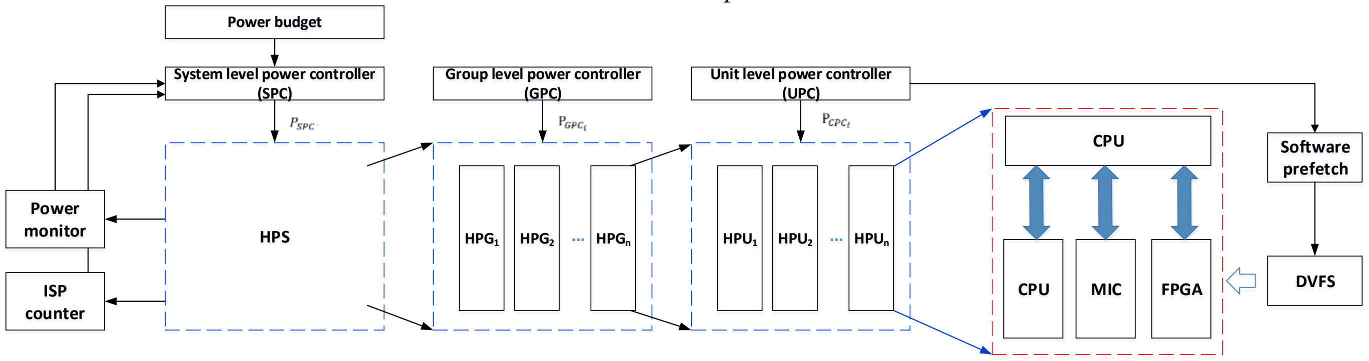

Fig.2. Hierarchical power management framework for heterogeneous systems

4. The UPC of a SPC based on software prefetch

Software prefetch refers to that programmers or compliers insert prefetched instructions in proper locations of codes to fetch data into cache or register in advance to avoid the pause of computation due to waiting for memory access. Software prefetch is characterized by flexibility, high performance and pertinency while it also causes software overhead and power consumption. Controlling consumptions of software prefetch mainly depends on the proper determination of the prefetch distance [25-26], (namely, to determine the distance from the prefetched instructions to true access), which is generally called prefetch scheduling. For the cyclic structure of parallel programs, prefetch distance refers to the cyclic iterations between prefetched instructions and true accesses.

To perfectly hide the memory access latency of prefetching, the prefetch scheduling should ensure that the time completing prefetched instructions is just at the moment of true access as far as possible. Therefore, the prefetch distance PD is determined by iteration and memory access latencies:

$$\mathrm{PD} = \left\lceil \frac{AL}{S} \right\rceil \quad (1)$$

Where *AL* refers to the average latency of memory access while *S* represents the possible shortest operating time of each cyclic iteration. The purpose of rounding up is to guarantee to complete prefetch operation before conducting data access.

A circuit cannot normally work unless its voltage and frequency are synchronously adjusted. Dynamic power consumption P_d is directly proportional to the cube of the frequency f of a processor, namely, $P_d \propto f^3$, while the execution time *T* of a program is approximately inversely proportional to the frequency *f*, namely, $\mathrm{T} \propto f^{-1}$. Therefore, the dynamic power consumption satisfies $E_d = P_d T \propto f^2$.

By simulating a single thread block, the relationship of the performances and power of programs with optimization based on software prefetch is explored here. To simulate true execution environment, the simulation on the single thread block does not mean that only a thread block operates on a Stream multiprocessor (SM). Multiple thread blocks synchronously operate on the same SM so that multiple thread blocks compete for resources on a SM. The quantity of synchronously operating thread blocks on a SM is denoted as M. Additionally, the thread blocks synchronously operating on diverse SMs also compete for global resource including Internet-on-a-chip and global cache. The quantity of SMs is denoted as N. Therefore, it is necessary to guarantee that there are MN thread blocks at least in a system while simulating the operation of a single thread block. In order to avoid that certain SMs fulfill works in advance and are free due to the load imbalance of SMs in test process, 2MN thread blocks are employed in simulation.

The method for optimizing performances of programs based on software prefetch under power constraints mainly consists of the following five steps.

Step 1: the source program is simulated to obtain the quantity of thread blocks simultaneously operating on the SM, expressed as M.

Step 2: the thread space of the source program is modified to assign 2MN thread blocks, execute and record the dynamic power consumption P_{CPC_0} and the average execution time T_0 of the first thread block on various SMs.

Step 3: according to current frequency, the proper prefetch distance is chosen. Formula (1) is for calculating the prefetch distance. If the upper and lower dimensions of the fraction are defined as wall-clock time but not clock cycles, Formula (1) can be rewritten as follows:

$$\mathrm{PD} = \left\lceil \frac{AL}{\omega/f} \right\rceil \quad (2)$$

Where, ω refers to the clock cycles of a single iteration. Generally, adjusting the operating frequency of a processor does not change the absolute latency of memory access. Namely, AL is unchanged after decreasing the frequency while the clock cycles of a single iteration do not change with the frequency, but the latency of each clock cycle increases. Thus, there is an approximate proportional relationship between the prefetch distance and the clock frequency, which can be expressed as $PD' = \alpha \cdot PD$. Where, PD' and α represent the adjusted prefetch distance and the adjustment factor of frequencies. During simulation, the range of prefetch distance is roughly determined according to the adjustment factor of frequencies first while determining proper prefetch distances according to the frequency. Afterwards, the proper prefetch distance is rapidly found by conducting fine adjustment. After selecting a proper prefetch distance, the source program is optimized based on software prefetch and the dynamic power consumption P_{CPC_1} is simulated and recorded by using the same thread space setting as Step 2.

Step 4: The frequency of a processor is adjusted by using the factor $\alpha = (\frac{\mathrm{P}_{CPC_0}}{\mathrm{P}_{CPC_1}})^{-1/3}$. Step 3 is repeated until $\mathrm{P}_{CPC_1} \approx \mathrm{P}_{CPC_0}$. In this context, the average execution time of the first thread block is denoted as T1 and the operating frequency of the processor is denoted as f_1, and $\alpha' = f_1/f_0$.

Step 5: If $T_1 < T_0$, the performance is optimized through software prefetch combining the frequency adjustment with a factor of α'. Otherwise, the source program and frequency f_0 are employed, which indicates that it is not applicable to carry out the optimization based on software prefetch for the program under power constraints.

5. The GPC of a SPC

In OpenMP-like programs, a computing segment is mainly composed of serial segments and parallel segments guiding statement identifications. Only a single thread operates in the serial segment and therefore all of the power P_{HPG_i} should be allocated to GPCs operated in the serial segment, namely, $\mathrm{P}_{CPC_i} = \mathrm{P}_{GPC_i}$. Where, P_{GPC_i} represents the power allocated to the *ith* GPC in a group. During the operation in a parallel segment, the local computing tasks are completed by multiple GPCs in parallel. Generally, there is a barrier synchronous statement at the exit of the parallel segment to guarantee that the subsequent operations are executed after all threads complete current parallel tasks. Therefore, the execution time of parallel programs depends on the execution unit with longest execution time, i.e., the execution unit in key paths. The power needs to be differentially distributed according to the execution time of different computing units under power constraints to optimize the performance of parallel executions.

The threads of key paths can be determined by inserting timestamps in compiling process to record the starting time of different threads (T_{start}) and time of entering and leaving synchronous statements (T_{enter}, T_{leave}) [27-28]. $T_{enter} - T_{start}$ and $T_{leave} - T_{enter}$ refer to effective computing time and waiting time of threads, respectively. In terms of a parallel execution process, the effective computing rate of a processor can be defined as the ratio of effective computing time to the total execution time, namely,

$$S = \frac{T_{enter} - T_{start}}{T_{leave} - T_{enter}} \qquad (3)$$

It can be clearly seen that the effective computing rate of key threads is 1 while the execution frequency of other processors can be declined to 100S% of the previous ones on the premise of not influencing the parallel execution time. On this basis, the influence of power constraint on performances can be minimized on condition that the effective computing rate ratio of various processors does not change under power constraints. Multiple HPUs exhibit isomorphic relations within a HPG. Therefore, the power in a group is divided by taking the effective computing rate of processors as the criterion: the power distributed to the ith UPC is

$$P_{CPC_i} = \frac{S_i}{\sum_{k=0}^{nCPC_i-1} S_k} P_{GPC_i} \qquad (4)$$

Where, $nCPC_i$ refers to the quantity of UPCs in the ith GPC. It can be seen from Formula (4) that when the loads on various UPCs are balanced, the power P_{GPG_i} should be uniformly allocated to all GPCs in the group, namely, $P_{CPC_i} = P_{GPG_i}/nCPC_i$.

6. The SPC

The SPC aims to improve the overall execution performance of a system and maintain the fairness between concurrent applications as much as possible while satisfying the power constraint. The concurrently executed multiple programs show different program characteristics and execution traces and therefore the execution rate of GPCs is described by using billions of instructions per second (BIPS). Due to different types of processors have different instruction set architectures, it is unfair to directly compare the BIPS of different types of processors. Therefore, the operating speed of accelerated processing units (APUs) is normalized as that of the main processor to measure the execution performance of GPCs at the same standard. The BIPS of each CPU in a GPC is denoted as BIP_0, which can be obtained by employing a hardware counter. The execution speed of a processor is described by using the iteration times per unit time and therefore those of the main processor and APUs under current operating frequency can be obtained, expressed as v_0 and v_k, respectively. It can be acquired that equivalent BIPS of the APU is $BIP_k = BIP_0 \times \frac{v_k}{v_0}$ and the execution speed of a single UPC can be calculated with the following formula:

$$v_{CPC} = BIP_0 \times \sum_{k=0}^{N_{CPC}} \frac{v_k}{v_0} \qquad (5)$$

Where, N_{CPC} refers to the quantity of processor units contained in a GPC. Due to the characteristics of applications, different application programs exhibit diverse BIPSs. Therefore, directly distributing power of a system according to absolute BIPS can artificially prolong the execution time of some application programs with low BIPSs, thus threatening the fairness of concurrent tasks. By using the evaluation method in [29], the executive characteristics of programs are described by employing the relative execution speed, which refers to the ratio of BIPS at current frequency to BIPS at the highest frequency, namely,

$$B = \frac{BIPS}{BIPS'} \qquad (6)$$

Where, $BIPS'$ represents the BIPS value of a GPC when all processors operate at the highest frequency. The power of the UPC at the frequency is denoted as P_{CPC}, and then the performance is defined as the ratio of the relative execution speed to power consumption, namely,

$$e_{CPC} = \frac{B}{P_{CPC}} \qquad (7)$$

Because GPC consists of multiple UPCs, the weighted sum of performances of various UPCs is taken as the performance of a GPC, namely,

$$e_{GPC_i} = \sum_{j=0}^{nCPC_i-1} e_{CPC_j} \times \omega_j \qquad (8)$$

Where, e_{CPC_j} and $nCPC_i$ refer to the execution performance of the jth UPC in a GPC and the quantity of UPCs contained in the ith GPC, respectively. $\omega_j (0 \le \omega_j \le 1)$ denotes the task load allocated to the jth UPC that satisfies $\sum_{j=0}^{nCPC_i-1} \omega_j = 1$. After acquiring the performances of various GPCs, the total power P_{SPC} is allocated to multiple GPCs according to the performance ratios of different programs. The total power allocated to the ith GPC is shown as follows:

$$P_{GPC_i} = \frac{e_{GPC_i}}{\sum_{k=0}^{nGPC-1} e_{GPC_k}} P_{SPC} \qquad (9)$$

Where, $nGPC$ refers to the quantity of concurrently executed programs in a system. According to the aforementioned power distribution strategy, the SPC executes the power control process in a fixed period T. Therefore, according to the operating performances e_{GPC_i} of various application programs in the prior control period, the power constraints of various programs in the next control period are formulated considering the current power constraint P of the system and transferred to the power controllers at the subordinated level.

7. Experimental evaluation and analysis

7.1 Experimental platform

The heterogeneous system consisting of an Intel Core I7 920 Quad-Core CPU and an AMD 4870 GPU was taken as the experimental platform. As shown in Table 1, in the heterogeneous system, the CPU and the GPU have their individual memory spaces and data communications are achieved through connection of PCI-E bus.

Table 1 Parameters of the test platform

Processor	Intel Core I7 920 CPU	AMD 4870 GPU-H/GPU-L
Frequency of processor (GHZ)	2.67,2.4,2.0.1.6	0.75,0.65,0.55
Frequency of memory (GHZ)	1.33 (DDR3)	0.9/0.7/0.5(GDDR5)
Cache	L1 I32KB, D32KB, L2 256KB, L3 8MB	-
Memory	8GB	1GB

Table 2 Test cases

Applications	Description	Problem scale	Kernel program
HopSpot	Thermal Simulation Tool	2048*2048 data points	Hotspot
Kmeans	Clustering Algorithm	819200 points 34 features	Cluster
MGRID	Poisson Equation Solver	256*256*256 data points	RESID, PSINV, RPRJ3, INTERP
SWIM	Shallow Water Modeling Solver	2048*2048 data points	CALC1, CALC2, CALC3

7.2 Test cases

Four applications are chosen as test cases. As shown in Table 2, MGRID and SWIM are both taken from the SPECOMP2001 benchmark test set and used as Poisson Equation Solver and shallow water modeling solver (SWMS), respectively. HotSpot and Kmeans are both collected from the Rodinia program set which is mainly applied for evaluating HPSs. HotSpot application is employed for simulating the chip temperature model while Kmeans is applied as a clustering algorithm frequently used in data mining field. These applications exhibit the following characteristics: having cyclic processes in Kernel function, containing the applications accessing global memory space in the cyclic processes and satisfying the basic conditions for conducting optimization based on software prefetch.

7.3 Experimental evaluation

7.3.1 Evaluation on the UPC based on software prefetch in a HPS

The UPC based on software prefetch is first evaluated. In the experiment, only a single application program operates each time and OpenMP parallel loop is subjected to parallel execution after being partitioned in the multi-core processor and the GPU acceleration unit by using manual modifying method. A specific processor core is in charge of controlling the execution of the GPU while the other three processor cores are responsible for the computation of the rest cyclic iterations. Figure 3 displays the actual power consumption of the system under different power constraints, where X- and Y-coordinates denote the given power constraint and actual power consumption, respectively. For the sake of simplicity, the figure merely shows the execution condition of a kernel in each application program. It can be seen from the figure that the power consumption of the system can accurately approach to but not exceed the given power constraint by using the proposed power control method based on software prefetch. Only the actual power consumption of CALC1 program has a large difference with the power constraint in the aforementioned four Kernel programs. This is mainly because the physical processor only supports limited dispersed operating frequencies and shows lower and upper bounds. The given power constraint cannot be used when the power constraints are 80% and 90%, thus resulting in power waste.

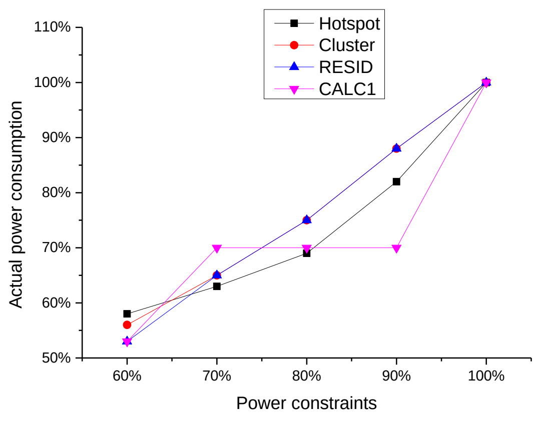

Fig. 3. Power consumption control precisions of single tasks

Figure 4 reveals the performance speedups of programs under power constraints. Overall performances increase by 17% and 11% respectively by using the register and the shared memory as the software prefetch buffers in the optimization. It can be seen from the figure that performances of Cluster and RESID do not improve while using shared memory as prefetch buffer while CALC3 does not acquire performance speedup with the two strategies. These programs which do not obtain performance speedups show the common characteristic: the frequency adjustment factor at power boundary is larger than that in optimal performance under power constraint. It means that the programs after prefetch optimization reach to the boundary of performance constraint at first while reducing frequencies. In this context, the power is still higher than the original level, so it is necessary continuously decrease the frequencies to reduce the power to the initial level while the execution time is higher than the initial value. Thus, it is improper to optimize these programs based on software prefetch from the perspectives of either time or power constraints.

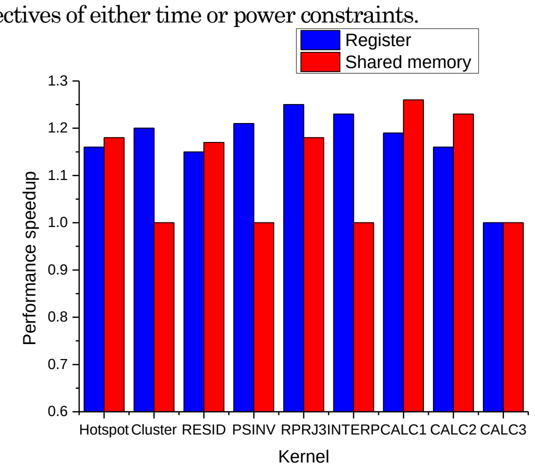

Fig. 4. Performance speedups after being optimized under power constraints

7.3.2 Evaluation on the SPC

The test platform contains two GPUs. HPGs are formed individually by using a CPU core and a GPU. Two individual programs are separately mapped on the two HPGs to successively evaluate the control precision of power consumption and average performance losses. Under the power constraint, the smaller the difference between the actual and the scheduled power consumptions is, the higher the control precision. Figure 5 displays the actual power consumption of various application combinations when the power constraint decreases to 80% of the maximum power consumption. The symbol x-y in the legend refers to the combination of programs x and y. It can be seen from the figure that the power consumption of the system favorable approach to the given constraint for different program combinations, showing an average difference of 4.8% with the constraint. Because the system power is distributed merely according to the latest execution condition of the system and not historical information, SPCs exhibit different power division to concurrent applications in different control periods. Consequently, the total power consumption of the system fluctuates.

Figure 6 illustrates performance changes of various program combinations when the power constraint of the system reduces to 80% of previously maximum power. The legend First and Second denote respectively the first and second program in a program combination while Total refers to the program combination. As shown in the figure, concurrent applications in most application combinations exhibit similar performance losses, showing an average performance loss of 5.4%. The applications in combinations H-M and K-M have great differences in performance loss. It is found through analysis that HotSpot and Kmeans both belong to high computationally-intensive applications and therefore their execution performances are significantly larger than that of MGRID. Thus, the two applications share larger power and exhibit higher execution speed while combining with MGRID.

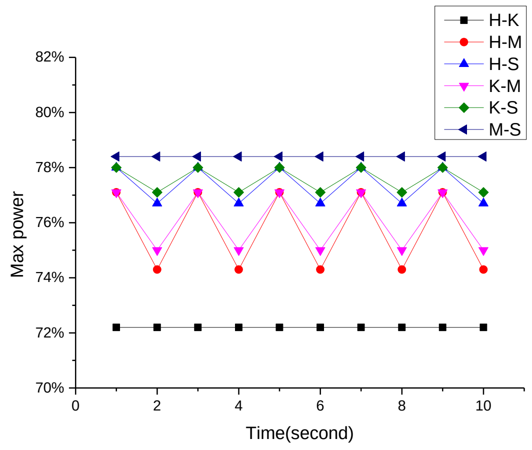

Fig. 5. Power control under the concurrent execution of multiple programs

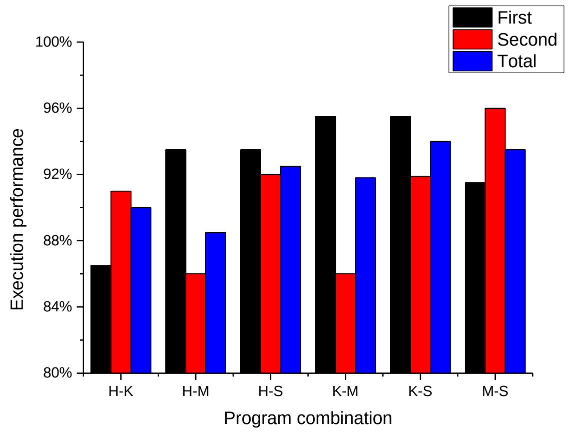

Fig. 6. Performance changes of various program combinations when the power constraint of the system reduces to the 80% of the maximum power consumption

8. Conclusions and future work

Aiming at multiple programs executed in a heterogeneous parallel system, this study established a management framework for system power combining a three-level power control technology including the UPC, GPC and SPC. We proposed a method for controlling power based on software prefetch in the UPC: The performance of source programs is optimized by conducting prefetch according to the optimal prefetch distance. Afterwards, the frequency and voltage reduction is carried out on the optimized program according to the power constraint conditions, followed by adjustment of the prefetch distance to guide the optimization objective to satisfy the boundary of constraint conditions. Through application test of typical scientific computation, the power consumption of the system can accurately approach to but not exceed the given power constraint by using the proposed power control method based on software prefetch. Furthermore, performances improve by 17% and 11% by applying register and shared memory as software prefetch buffers in the optimization, respectively. In HPG power control, GPCs takes charge of partitioning the power allocated to specific application programs to multiple heterogeneous processing engines under concurrent executions. Power is uniformly allocated to multiple engines under load balance while it is preferentially allocated to those in key paths under load imbalance to improve the overall concurrent execution performance. In SPC, the system power is apportioned to diverse concurrently executed application programs. The ratio of relative execution speed to power consumption is considered as the execution performance to distribute system power for the sake of guaranteeing the fairness of concurrent applications. The experimental results indicate that the SPC can successfully approach to the given power constraint, with an average difference of 4.8% with the power constraint. On the other hand, the fairness is favorably guaranteed: the average performance loss between concurrent applications is only 5.4%.

In this paper, our research mainly focuses on dynamic energy consumption, so it is assumed that the static energy consumption remains unchanged during in the system running process. However, as the microprocessors design technology advances to the deep nanometer era, the proportion of static energy consumption is increasing, the focus of energy consumption optimization will gradually shift from dynamic energy to dynamic energy and static energy regarded both as equally important. In future work, we will consider both the static energy and dynamic energy and comprehensively utilize a variety of optimized technologies to improve the execution performance of large-scale heterogeneous parallel system.

Acknowledgments

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

References and links

1. Gu, C., Fan, L., Wu, W., Huang, H., & Jia, X. Greening cloud data centers in an economical way by energy trading with power grid. Future Generation Computer Systems. 2018,78(1):89-101
2. Baker T, Al-Dawsari B, Tawfik H, et al. GreeDi: An energy efficient routing algorithm for big data on cloud. Ad Hoc Networks, 2015, 35:83-96.
3. Baker T, Asim M, Tawfik H, et al. An energy-aware service composition algorithm for multiple cloud-based IoT applications. Journal of Network & Computer Applications, 2017, 89(C):96-108.
4. Makaratzis A T, Giannoutakis K M, Tzovaras D. Energy modeling in cloud simulation frameworks. Future Generation Computer Systems, 2018,79(2):715-725.
5. Wang B, Yang Q, Deng X. Energy management for cost minimization in green heterogeneous networks. Future Generation Computer Systems, 2017
6. Hu Y, Liu C, Li K, Chen X, Li K. Slack allocation algorithm for energy minimization in cluster systems. Future Generation Computer Systems, 2017(74):119-131.
7. Horvath T, Skadron K. Multi-mode energy management for multi-tier server clusters. International Conference on Parallel Architectures and Compilation Techniques. IEEE, 2017:270-279.
8. Baker T, Ngoko Y, Tolosanacalasanz R, et al. Energy Efficient Cloud Computing Environment via Autonomic Meta-director Framework. Sixth International Conference on Developments in Esystems Engineering. IEEE, 2015:198-203.
9. Liu Y, Yuen C, Yu R, et al. Queuing-Based Energy Consumption Management for Heterogeneous Residential Demands in Smart Grid. IEEE Transactions on Smart Grid, 2017, 7(3):1650-1659.
10. Aldawsari B, Baker T , England D. Trusted Energy-Efficient Cloud-Based Services Brokerage Platform . International Journal of Intelligent Computing Research, 2015, 6(4):630-639.
11. Wang X, Chen M. Cluster-level feedback power control for performance optimization. In 17th International Conference on High-Performance Computer Architecture. 2008:101-110
12. Raghaverdra R, Ranganathan P, Talwar V, et al. No "power" struggles: coordinated multi-level power management for data center. In Proceedings of the 13th international conference on architectural support for programming languages and operating systems. New York, NY, USA, 2008:48-59.
13. Isci C, Buyuktosunoglu A, Cher C-Y, et al. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. In Proceeding of the 39th Annual IEEE/ACM International Symposium on Microarchitecture Washington, DC, USA, 2006:347-358.
14. Meng K, Joseph R, Dich R P, et al. Multi-Optimization Power Management for Chip Multiprocessors. In PACT'08: Proceedings of the 15th international conference on parallel architectures and compilation techniques. New York, NY, USA, 2008:177-186.
15. Ma K, Li X, Chen M, et al. Scalable power control for many-core architectures running multi-threaded applications. In Proceeding of the 38th annual international symposium on Computer architecture. New York, NY, USA, 2011:449-460.
16. Yang H , Gao G R. Power-aware Compilation Techniques for High Performance Processors. Doctor dissertation. University of Delaware. Winter 2004.
17. Hsu C H, Kremer U. Single Region vs. Multiple Regions: A Comparison of Different Compiler-Directed Dynamic Voltage Scheduling Approaches. Lecture Notes in Computer Science, 2002, 2325:43-48.
18. Saputra H, Kandemir M, Vijaykrishnan N, et al. Energy-conscious compilation based on voltage scaling. Acm Sigplan Notices, 2002, 37(7):2-11.
19. Xie F, Martonosi M, Malik S. Compile-time dynamic voltage scaling settings: opportunities and limits. ACM SIGPLAN Notices, 2003, 38(5):49-62.
20. Fan X, Ellis C S, Lebeck A R. The synergy between power-aware memory systems and processor voltage

scaling. International Conference on Power - Aware Computer Systems. Springer-Verlag, 2003:164-179.

21. Wulf W A, McKee S A. Hitting the Memory Wall: Implications of the Obvious. Computer Architecture News. 1995,23(1):20-24.
22. Mowry T C. Tolerating latency through software-controlled data prefetching. Ph.D. thesis. Stanford University, 1995.
23. Chen S, Gibbons P B, Mowry T C. Improving index performance through prefetching[. Acm Sigmod Record, 2000, 30(2):235-246.
24. Juan C. Study of Low-Power Software Optimization Technology. Ph.D. thesis. National University of Defense, Technology.2007.
25. Klaiber A C , Levy H M. An architecture for software-controlled data prefetching. In Proceedings of the 18th annual international symposium on Computer architecture. New York, NY, USA, 1991:43-53.
26. Mowry T C, Lam M S, Gupta A. Design and evaluation of a compiler algorithm for prefetching. In Proceedings of the fifth international conference on Architectural support for programming languages and operating systems. New York, NY, USA, 1992:62-73.
27. Cai Q, Rakvic R, Magklis G, et al. Meeting points:using thread criticality to adapt multicore hardware to parallel regions// International Conference on Parallel Architectures and Compilation Techniques. IEEE, 2017:240-249.
28. Bhattacharjee A, Martonosi M. Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors// International Symposium on Computer Architecture. ACM, 2009:290-301.
29. Ma K, Li X, Chen M, et al. Scalable power control for many-core architectures running multi-threaded applications // International Symposium on Computer Architecture. ACM, 2011:449-460.