2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)

Exploration on Task Scheduling Strategy for CPU-GPU Heterogeneous Computing System

Juan Fang[1], Jiaxing Zhang[1], Shuaibing Lu[1], Hui Zhao[2]
[1]Faculty of Information Technology, Beijing University of Technology
fangjuan@bjut.edu.cn, S201761381@emails.bjut.edu.cn, shuaibinglu@bjut.edu.cn
[2]Department of Computer Science and Engineering, University of North Texas, hui.zhao@unt.edu

Abstract— **For a CPU-GPU heterogeneous computing system, different types of processors have load balancing problems in the calculation process. What's more, multitasking cannot be matched to the appropriate processor core is also an urgent problem to be solved. In this paper, we propose a task scheduling strategy for high-performance CPU-GPU heterogeneous computing platform to solve these problems. For the single task model, a task scheduling strategy based on load-aware for CPU-GPU heterogeneous computing platform is proposed. This strategy detects the computing power of the CPU and GPU to process specified tasks, and allocates computing tasks to the CPU and GPU according to the perception ratio. The tasks are stored in a bidirectional queue to reduce the additional overhead brought by scheduling. For the multi-task model, a task scheduling strategy based on the genetic algorithm for CPU-GPU heterogeneous computing platform is proposed. The strategy aims at improving the overall operating efficiency of the system, and accurately binds the execution relationship between different types of tasks and heterogeneous processing cores. Our experimental results show that the scheduling strategy can improve the efficiency of parallel computing as well as system performance.**

Keywords- Heterogeneous Computing System; Load balancing; Scheduling；Genetic Algorithm；

I. INTRODUCTION

With the advance of manufacturing processes, computer architectures have undergone tremendous changes. The single-core processor structure is limited by physical design limits and energy consumption, which will inevitably lead to the shift of the focus of Moore's Law from the number of simple transistors to the number of cores that can be integrated on the chip [1,2]. Therefore, the development direction of processors has changed from increasing the computing power of single-core processors, such as increasing the main frequency, to increase the number of processors that can be integrated on a single chip. The processor architecture has evolved from single-core architectures to multi-core architectures. However, as we enter the information era, the computation needs to be processed have also grown exponentially. In the face of massive data storage and computation, even if there are dozens of CPUs integrated on the same chip, such a system still cannot meet the needs of scientific computing. The isomorphic multi-core processor cannot support such a huge amount of calculations [3]. Therefore, combining the CPU and GPU into a heterogeneous computing system has emerged as an ideal solution. At present, heterogeneous computing systems have been proposed to include various computing units such as CPUs, DSPs, GPUs, ASICs, and FPGAs, using different types of instruction sets and "integration" of different architectures, allowing various cores to effectively collaborate and cope with multiple computing scenarios[4]. Due to the excellent performance/cost efficiency ratio of GPUs, the GPU-CPU heterogeneous architectures become the most commonly used heterogeneous computing system in high-performance computing clusters [5]. In such systems, the CPUs and GPUs belong to different types of computing resources and can provide powerful computing capabilities, however, in most current systems, the CPUs are only responsible for assigning parallel computing tasks to the GPUs. When the GPUs execute parallel tasks, the CPUs only waiting for the GPUs to return the results, without participating in the execution of the parallel tasks. As a result, the computing resources of the heterogeneous system cannot be fully utilized. Most research is also focused on accelerating GPU calculations and the powerful computing power of the CPU is not fully utilized [6-9]. The performance resources of the system have not been fully tapped due to unreasonable allocation, and few solutions can fully utilize the collaborative computing capabilities of CPU and GPU [10]. It is one of the problems to be solved urgently in the field of CPU-GPU collaborative computing by reasonably allocating the computing resources of the heterogeneous computing system to fully utilize the computing power of the CPU-GPU heterogeneous computing system. Therefore, designing an effective task scheduling algorithm for heterogeneous architectures has become an important idea to solve this problem.

A. Contributions

In this paper, we investigate the characteristics of different benchmarks and propose to improve the performance of heterogeneous computing systems through task scheduling algorithms. We developed a task scheduling strategy based on load-aware and a task scheduling strategy based on the genetic algorithm. Our evaluation results show

2159-3477/20/$31.00 ©2020 IEEE
DOI 10.1109/ISVLSI49217.2020.00063

Authorized licensed use limited to: University of North Texas. Downloaded on August 08,2022 at 15:46:44 UTC from IEEE Xplore. Restrictions apply.

that, the scheduling strategies proposed in this paper can significantly tap the performance limit of the system.

Our contributions can be summarized as follows:

•We propose a task scheduling strategy based on load-aware for CPU-GPU heterogeneous computing platform which uses the CPUs and GPUs to work together to complete parallel computing tasks under the single task model. This scheduling strategy significantly reduces the uneven load and only brings a small amount of additional scheduling overhead, which greatly reduces the idle time after the CPUs and GPUs finish their respective computing tasks.

•We propose a task scheduling strategy based on the genetic algorithm for CPU-GPU heterogeneous computing platform under the multi-task model. The strategy aims at improving the overall operating efficiency of the system, and accurately binds the execution relationship between different types of tasks and heterogeneous processing cores.

•We performed extensive evaluations of the scheduling strategies proposed in this paper. We show the advantage of the proposed load-aware scheduling algorithms, compared with traditional scheduling algorithms, static and dynamic scheduling algorithms, and hybrid scheduling algorithms. Compared with the conventional genetic algorithm and particle swarm algorithm, our optimized genetic algorithm can quickly reach convergence and can achieve higher system performance.

B. *Organization of the paper*

The rest of this paper is organized as follows. Section II introduces the research trends of task scheduling in heterogeneous computing systems. Section III introduces the characteristics of the benchmark applications and the design of our two parallel task model scheduling strategies. Section IV presents the results of our experimental evaluation. Section V concludes the paper and discusses our future work.

II. BACKGROUND

With the advancement of technology, the computing resources within the system have changed from single-core to homogeneous multi-core, and then developed into heterogeneous multi-core. Task scheduling algorithms have an increasing impact on improving the efficiency of heterogeneous systems and mining the performance of heterogeneous systems. On the one hand, loop processing in scientific computing is the most common parallel task. These loops, as the most computationally intensive part of the program, are one of the hot topics in parallel computing research. This model is a single task parallel loop [10]. On the other hand, with the complexity of computing scenarios, there are many single tasks in the system that can be processed simultaneously, and these single tasks can be combined into a multi-task model for parallel processing.

A. *Single task model*

Programmers divide parallel tasks into different parts, and the CPUs and GPUs process different parts of parallel tasks at the same time [11-14]. This type of coordination can avoid idle time in the computing resources and shorten the task execution time, and improve the system performance. Although the CPUs and GPUs have their preference in tasks processed, collectively, using these two types of processors to jointly process computing tasks can improve the overall system performance.

A static scheduling strategy has relatively smaller scheduling overhead and is easy to implement, but it tends to cause uneven load allocation [12]. Dynamic scheduling refers to a scheduling scheme that is performed under uncertain operating environments and computing tasks [12]. Dynamic scheduling solves the problem that static scheduling cannot and can work with flexible CPU-GPU configurations and large uneven loads, but the resource and time overhead brought by dynamic scheduling is large, which can have a significant impact on the overall system performance.

B. *Multi-task model*

Baruah S demonstrated that multi-task scheduling for heterogeneous systems is an NP-hard problem [15]. The swarm intelligence algorithm provides a good solution for this type of problem, and there are many research results on multi-task assignment at home and abroad.

Andrew J. Page and others have developed schedulers for task allocation in dynamic heterogeneous distributed systems [16]. Experiments show that this algorithm achieves higher efficiency than other commonly used heuristic algorithms. The CPU-GPU heterogeneous computing system is a stand-alone heterogeneous environment. The computing speed of the two heterogeneous chips is very different, and the task scheduling algorithm designed for distributed systems is not applicable. The multi-task scheduling strategy for CPU-GPU heterogeneous computing systems mostly stays in rewriting specific applications into methods suitable for heterogeneous system execution to improve the performance of the system when executing the application. For example, Li D combined genetic algorithm and particle swarm optimization to study an efficient hybrid genetic algorithm and particle swarm optimization for molecular dynamics simulation on heterogeneous supercomputer load balancing [17]. Such solutions are not universal and cannot be adapted to general applications. And for the multitasking model, load balancing in some scenarios does not guarantee the strongest system performance obtained by the scheduling strategy. Therefore, this article will focus on how to improve the performance of applications executed on CPU-GPU heterogeneous platforms by binding the execution relationship between subtasks and processor cores.

III. DESIGN OF SCHEDULING STRATEGY FOR CPU-GPU HETEROGENEOUS COMPUTING SYSTEM

As one of the research hotspots in parallel computing models, loop processing is the most computationally intensive part of scientific computing programs. This paper refers to the parallel task models completed in a cycle as a

Authorized licensed use limited to: University of North Texas. Downloaded on August 08,2022 at 15:46:44 UTC from IEEE Xplore. Restrictions apply.

single task model. As the computing scenarios become more and more complex, there are many simultaneous single-task parallel loops in the system. These single tasks can be processed by the system according to a certain scheduling strategy. Assuming that there are M processor cores in a heterogeneous system, the system can process M single tasks at the same time. The multi-task model designed in this paper includes N single tasks, which are processed by a heterogeneous system composed of M computing cores. This paper refers to this type of parallel task model as a multi-task model.

A. Task scheduling strategy based on load-aware

Load balancing in a CPU-GPU heterogeneous computing platform means that the computing tasks loaded to the CPU and GPU are completed at the same time. Assume that the sum of the data amount of the computing task is S, and the speed that CPU and GPU calculate the task are V_{CPU} and V_{GPU}, respectively. The time taken by the CPU and GPU to participate in the calculation is T_{CPU} and T_{GPU}, respectively. Let $\delta(\delta > 0)$ be the degree of load unevenness in the system. A lower δ indicates that the load unevenness of the system is small. Δ can be expressed by formula (1).

$$\delta = \left|1 - \frac{T_{CPU}}{T_{GPU}}\right| \tag{1}$$

When $T_{CPU} = T_{GPU}$, the load is balanced. Let the ratio of the amount of data allocated by the CPU and GPU be P, and the formula in this balanced state is shown in (2).

$$S_{CPU} = S * \frac{P}{1+P}, \quad S_{GPU} = S * \frac{1}{1+P} \tag{2}$$

There are many factors that affect the stability of the system performance, and the resource allocation and time consumption overhead of the dynamic scheduling process can be prohibitive [19]. In this paper, we propose a task scheduling strategy based on load-aware.

Before processing parallel tasks, the computing tasks are stored in a two-way queue. When distributing computing tasks to different types of processors, the CPU is dispatched from the head of the queue and the GPU is dispatched from the tail of the queue. The core component of the strategy is resource prediction and allocation. Solving V_{CPU} and V_{GPU} is the key to the prediction of the strategy. The strategy is implemented according to the following four steps:

Step 1: Check whether it is the first time that the system runs the program. If so, 10% of data is equally distributed to the CPU and GPU to detect the computing capacity of the heterogeneous computing system. Otherwise, 10% of data is distributed according to the allocation ratio P stored by the system, and we evaluate whether the current allocation ratio is suitable at this time system status. The remaining 90% of the data volume is calculated in the third stage.

$$s_{CPU} = 0.1S * \frac{P}{1+P}, \quad s_{GPU} = 0.1S * \frac{1}{1+P} \tag{3}$$

Step 2: According to the CPU calculation time t_{CPU} and GPU calculation time t_{GPU} of the first stage of the execution, derive the current state of the system to calculate the performance ratio of the task. When the system runs the program for the first time, the ratio p is solved by formula (4), otherwise formula (5).

$$p = \frac{v_{CPU}}{v_{GPU}} = \frac{s_{CPU}/t_{CPU}}{s_{GPU}/t_{GPU}} = \frac{t_{GPU}}{t_{CPU}} \tag{4}$$

$$p = \frac{v_{CPU}}{v_{GPU}} = \frac{s_{CPU}/t_{CPU}}{s_{GPU}/t_{GPU}} = \frac{0.1S * \frac{P}{1+P}\Big/t_{GPU}}{0.1S * \frac{1}{1+P}\Big/t_{CPU}} = P\frac{t_{GPU}}{t_{CPU}} \tag{5}$$

The degree of load unevenness δ' in the detection phase is shown in formula (6). When $\delta' \leq \mu$, the calculation amount distribution ratio P is unchanged; When $\delta' > \mu$, the calculation amount distribution ratio P is updated as p, where μ is the threshold set by the system.

$$\delta' = \left|1 - \frac{t_{CPU}}{t_{GPU}}\right| \tag{6}$$

Step 3: Hybrid scheduling. Distribute 80% of the total data amount to the CPU and GPU according to the ratio determined in the previous step and the remaining 10% is dynamically scheduled. The processor that completes the calculation task first does not need to wait for the processor that completes later, and directly enters the queue in the dynamic scheduling stage. The processor that completes later also enters the queue in the dynamic scheduling stage after completing its allocated amount of task. This process will continue until the beginning and end of the queue meet, and all computing tasks have been completed.

Step 4: Calculate the computing time t'_{CPU} and t'_{GPU} of the CPU and GPU for static scheduling of computing tasks in the hybrid scheduling phase. The optimal allocation ratio P' for the task of the solving system is shown in formula (7).

$$P' = P\frac{t'_{GPU}}{t'_{CPU}} \tag{7}$$

The discrete degree of P and P' is ε, and ε is expressed by formula (8).

$$\varepsilon = \left|1 - \frac{P}{P'}\right| \tag{8}$$

When $\varepsilon > \nu$, update the CPU to GPU load ratio of the heterogeneous computing system to P', where ν is the threshold set by the system.

B. Task scheduling strategy based on genetic algorithm

A swarm intelligence algorithm provides a method for obtaining approximate optimal solutions for NP-hard problems, but it has the disadvantages, such as too low efficiency and falling into local optimal solutions. This paper proposes to improve the genetic algorithm using following steps.

Step 1: Initialize the genetic algorithm parameters. Each individual of the initial population represents a task scheduling scheme. The traditional genetic algorithm generates the initial population randomly. Considering the dependencies between tasks in this model, the method of

Authorized licensed use limited to: University of North Texas. Downloaded on August 08,2022 at 15:46:44 UTC from IEEE Xplore. Restrictions apply.

generating the initial population needs to be improved. The optimized population generation method is as follows:

(i) Calculate the height value $H(T_i)$ of all tasks according to the DAG diagram;

(ii) Assigned all tasks to the CPU and GPU randomly;

(iii) Sort the tasks that are assigned randomly on each core according to the $H(T_i)$ obtained in (i) from small to large, and the ranking result is the execution order of the tasks on the processing unit;

(iv) If the initial population size meets the requirements (100), go to step 2; otherwise, switch back to (ii);

Step 2: Calculate the fitness function of all individuals in the population, and rank all individuals in the population according to the fitness order. The selection of the fitness function directly affects the convergence speed of the algorithm and whether it can find the optimal solution.

Like most intelligent algorithms, genetic algorithms also judge whether a solution is good or bad based on the value of the fitness function. Let the scale of the current population be Sca, and the time required for the execution of the scheduling scheme S is recorded as $T_{total}(S)$, and the total running time of the current population T_{sum} is expressed by formula (9).

$$T_{sum} = \sum_{i=0}^{Sca-1} T_{total}(S), 0 \leq S \leq Sca - 1 \quad (9)$$

The fitness function value of the scheduling scheme S is defined as formula (10):

$$Fit(S) = \frac{T_{sum} - T_{total}(S)}{T_{sum}} \quad (10)$$

Step 3: Crossover. Crossover rate is set to 0.5. Cross operations on the two adjacent chromosomes sorted in step 2 to generate the offspring. Then we recalculate the fitness of the generated offspring and their parents, and we select a new one based on the fitness in descending order. Population and the size of the new population is consistent with the size of the parent population.

Step 4: Mutation. The choice of mutation probability P_m in the parameters of the genetic algorithm has a great influence on the behavior and the performance of the genetic algorithm. Genetic algorithms perform mutation operations on individuals at random. When the mutation probability is too large, it is easy to destroy the genes of excellent individuals with high adaptability in the population and enter a random search. However, it is difficult to introduce new genes with a low mutation rate, which causes the algorithm to stagnate in later iterations of the algorithm, which leads to the problems such as precocity and the local optimal solution. The mapping scheme formed at this time is not the global best solution. If it is determined through repeated experiments, the value of P_m is cumbersome. This paper proposes an adaptive mutation strategy. The P_m is shown in Equation (11) when $Fit_{max} \geq Fit$, otherwise it is set to 0.8.

$$P_m = k_m \frac{Fit_{max} - Fit_S}{Fit_{max} - Fit} \quad (11)$$

k_m is set to 0.2, Fit_{max} refers to the largest fitness function value among all the scheduling schemes in the population, Fit_S refers to the fitness of the scheduling scheme S, and Fit refers to the average fitness function value of all scheduling schemes in the population.

The specific operation of mutation is as follows. For each individuality, a random number p between $[0,1]$ is generated. If p is greater than P_m, then the individual performs the mutation operation. The process of mutation of a single chromosome is: the random position of the chromosome corresponds to a change in value, and the change of this value indicates the change of the processor number of the subtask. Next, we recalculate the fitness of the mutated individuals and their parents, and select a new population according to the fitness order. The new population size is consistent with the parent population size.

Step 5: If the maximum number of iterations (100) is reached, the task allocation scheme with the largest fitness function is selected; otherwise, we find the optimal solution of successive multi-generation populations respectively, and then determine whether the potential premature convergence occurs based on the Hamming distance between the optimal solutions of successive multi-generation populations. If no premature occurs, then go to step 3; if precocity occurs, then enable the injection strategy and move to the second step.

We describe the condition for judging premature convergence: it is judged as premature convergence when the Hamming distance between the optimal solutions of successive multi-generational populations is 0. The mechanism is based on detecting the Hamming distance between the best solutions of successive generations. If the best solution remains the same after successive generations, then the injection strategy is applied.

IV. Performance Evaluation of the Proposed Scheduling Strategy

The computing platform used in our experiments was Sugon W580-G20, running Linux, and the CPU was Intel Xeon (R) CPU E5-2620 by Intel; the GPU was NVIDIA Tesla P100 from Nvidia; the memory was 128Gb and the hard disk was 2Tb.

This paper selects three sets of loop processing parallel computing benchmark programs, of which MatMul is a classic bench program in the field of parallel computing; Pathfinder and Kmeans are from the latest version of the Rodinia benchmark suite; for the multi-task model, use the international general task map generation tool TGFF to generate random tasks for experiment [20].

A. Results of task scheduling strategy based on load-aware

In this section, we present our evaluation results using the three benchmark applications. Our results show that the scheduling strategy leads to a significant performance improvement in reducing the gap between the CPU and GPU computing programs of the same order of magnitude. The scheduling strategy based on load-aware stores the computation tasks in a bidirectional queue before processing the parallel tasks. Combined with a dynamic scheduling

Authorized licensed use limited to: University of North Texas. Downloaded on August 08,2022 at 15:46:44 UTC from IEEE Xplore. Restrictions apply.

strategy, the load balancing ability is stronger. Among them, the task scheduling strategy based on load-aware achieves significantly better performance than the static scheduling strategy and the hybrid scheduling strategy. For the three sets of MatMul, Pathfinder and Kmeans benchmarks selected in this paper, the performance of the systems has been improved by 29.97%, 27.62% and 26.63% respectively, with an average increase of 28.08%. Compared with the traditional scheduling scheme, the other two scheduling schemes improved performance by 12.96% and 17.30%, respectively. The performance evaluation is given in Figure 1. From the same set of examples with different data input sizes, we can observe that the real-time performance of the system is not stable, resulting in a large gap between the actual value and the theoretical value for the performance improvement of the static scheduling strategy as well as the static and dynamic scheduling strategy. Under the same environment, the maximum fluctuation is 9% and 8%, respectively. The fluctuation of the task scheduling strategy based on load-aware is less than 4%.

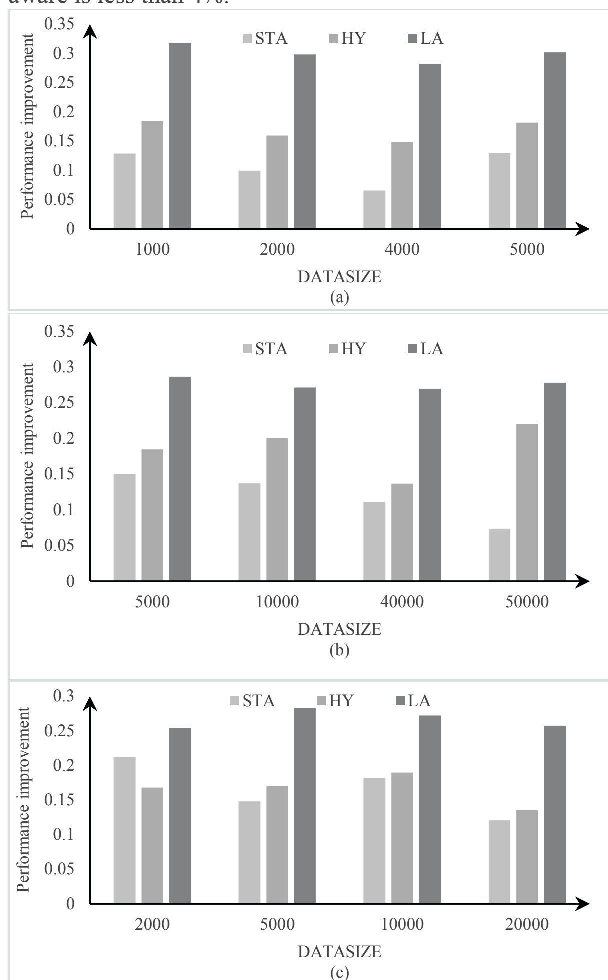

Fig. 1. Performance of static scheduling strategy (STA), hybrid scheduling strategy(HY) and load-aware scheduling strategy(LA). (a) MatMul. (b) Pathfinder. (c) Kmeans.

To summarize, the scheduling strategies proposed in this paper perform well for situations where the computing performance of the CPU and GPU are in the same order of magnitude. For scenarios where the performance of the CPU and GPU differ significantly, the improvement is not as significant as the earlier case. For example, for the Nbody instance from the official Nvidia test document, the GPU computing speed of this instance is nearly 2500 times that of the CPU. Nbody theoretically has a performance improvement of only 0.01%. The limitations of this type of instance is determined by the optimization method, regardless of the scheduling strategy. At the same time, the overhead in time caused by building task queues and 10% dynamic scheduling did not have a significant impact on the system performance.

B. Results of task scheduling strategy based on genetic algorithm

Convergence speed is one of the key indicators for evaluating the performance of swarm intelligence algorithms, and generally refers to the iteration of swarm intelligence algorithms to obtain the global approximate optimal solution time. In this subsection, three sets of test cases are created using TGFF, and then we performed experiments to test the performance of IM_GA and scheduling strategy. Our results show that, IM_GA has the fastest convergence speed among the three swarm intelligence algorithms, followed by a particle swarm algorithm, and the traditional genetic algorithm is the slowest.

As shown in Figures 2 and 3, the improved genetic algorithm in the performance of the scheduling strategy performs the best. The ranking of performance improvement of the scheduling strategy using three algorithms in TGFF-1 is GA <PSO <IMGA. IMGA strategy is 10.19% and 8.97% higher than the GA strategy and the PSO strategy, respectively. GA and PSO fall into the local optimal solution. In TGFF-2, the performance of the GA strategy is better than PSO, but GA and PSO are once again caught in the local optimal solution, both of which are 9.30% and 11.98% lower than the performance of the IM_GA strategy, respectively. For TGFF-3, the PSO strategy and the IM_GA strategy has the same performance, which is 11.84% higher than GA, and neither of them falls into the local optimal solution, while the convergence speed of IM_GA is much faster than PSO.

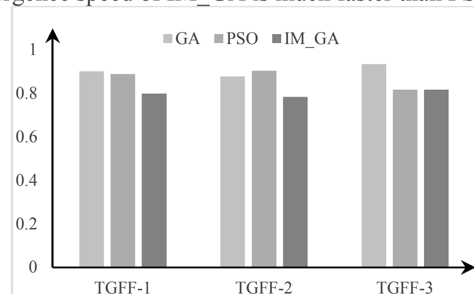

Fig. 2. Performance of GA, PSO and IM_GA optimization.

Authorized licensed use limited to: University of North Texas. Downloaded on August 08,2022 at 15:46:44 UTC from IEEE Xplore. Restrictions apply.

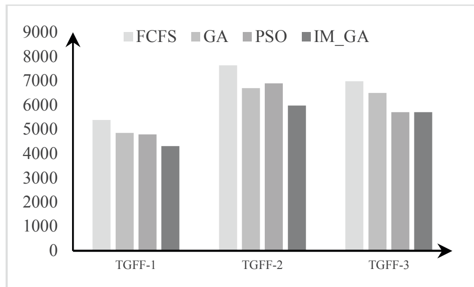

Fig. 3. Time consumption of FCFS,GA,PSO and IM_GA.

To summarize, the convergence speed of IM_GA is the fastest among the three algorithms. Moreover, the scheduling strategy using IM_GA has the best performance of all. In the three sets of benchmarks, the IM_GA scheduling strategy is better than GA scheduling strategy, which proves that IM_GA has a strong ability to jump out of the local optimal solution, since IM_GA uses an adaptive mutation mechanism and injection strategy.

Adaptive mutation can effectively retain the good individuals in the population and ensure that some good new individuals are effectively generated. The injection strategy is to inject a user-defined random number of artificial chromosomes (mapping solution). Whenever a potential precocity is detected when converging, this injection strategy will be triggered. It has a powerful complementary diversity mechanism, which enables the algorithm to jump out of the local optimal solution more easily, and solves the problem of premature easiness and avoids falling into the local optimal of the algorithm.

V. CONCLUSIONS

This paper proposes task scheduling strategies for two different types of parallel tasks. We first discuss the scientific calculations used for loop processing and propose a task scheduling strategy based on load-aware. This strategy can greatly reduce the waste of resources caused by the uneven load of heterogeneous processors. We also explore the multi-task model. A task scheduling strategy based on the genetic algorithm is proposed, which can bind the execution relationship between different types of tasks and heterogeneous processing cores. Our proposed techniques can achieve better performance than traditional genetic algorithms. Our experiment results show that the scheduling strategies in this work can significantly improve system performance.

ACKNOWLEDGEMENT

This work is supported by Beijing Natural Science Foundation (4192007), and the National Natural Science Foundation of China (61202076), along with other government sponsorships. The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements of our work.

REFERENCES

[1] Danowitz, Andrew, et al. "CPU DB: Recording Microprocessor History." Communications of the ACM, 55.4(2012):pp.55-63.期刊

[2] Chau R. "Process and Packaging Innovations for Moore's Law Continuation and Beyond." IEEE International Electron Devices Meeting, 2019:111-116.

[3] Lee V W, Kim C, Chhugani J, et al. "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU." Proceedings of 37th Annual International Symposium on ComputerArchitecture, 2010:451-460.

[4] Sparsh, Mittal, Jeffrey, et al. "A Survey of CPU-GPU Heterogeneous Computing Techniques. " Acm Computing Surveys, 47.4(2015):69.

[5] Mahfoudhi R, Achour S, Mahjoub Z. "Parallel triangular matrix system solving on CPU-GPU system." 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications, 2016:1-6.

[6] Shen J, Varbanescu A L, Zou P, et al. "Improving Performance by Matching Imbalanced Workloads with Heterogeneous Platforms." International Conference on Supercomputing, 2014: 241-250.

[7] Fang J, Leng Z Y, Liu S T, et al. "Exploring Heterogeneous NoC Design Space in Heterogeneous GPU-CPU Architectures." Journal of Computer Science and Technology, 30.1(2015):pp.74-83.

[8] Fang J, Zhang X B, Liu S J. "LLC Buffer Management Strategy Based on Heterogeneous Multi-core." Journal of Beijing University of Technology, 45.05(2019):pp.421-427.

[9] Iturriaga S, Nesmachnow S, Luna F, et al. "A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing systems." Journal of Supercomputing, 71.2(2015):pp.648-672.

[10] Shen W, Sun L, Wei D, et al. "Load-Prediction Scheduling for Computer Simulation of Electrocardiogram on a CPU-GPU PC." 2013 IEEE 16th International Conference on Computational Science and Engineering, 2013:213-218.

[11] Shulga D A, Kapustin A A, Kozlov A A, et al. "The scheduling based on machine learning for heterogeneous CPU/GPU systems." 2016 IEEE NW Russia Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), 2016:345-348.

[12] Bao Z, Chen C, Zhang W. "Task Scheduling of Data-Parallel Applications on HSA Platform." ICPCSEE, 2018:452-461.

[13] Ma K, Li X, Chen W, et al. "GreenGPU: A holistic approach to energy efficiency in GPU-CPU heterogeneous architecture." Proc. the 41st Int. Conf. Parallel Processing, 2012:48-57.

[14] Shen W, Luo Z, Wei D, et al. "Load-prediction scheduling algorithm for computer simulation of electrocardiogram in hybrid environments." The Journal of Systems and Software, 102.4(2015):pp.182-191.

[15] Baruah S. "Task partitioning upon heterogeneous multiprocessor platforms." Proceedings. RTAS 2004. 10th IEEE Real-Time and Embedded Technology and Applications Symposium, 2004:536-543.

[16] Page A J, Keane T M, Naughton T J. "Multi-heuristic dynamic task allocation using genetic algorithms in a heterogeneous distributed system." Journal of parallel and distributed computing, 70.7(2010) :pp.758-766.

[17] Li D, Li K, Liang J, et al. "A hybrid particle swarm optimization algorithm for load balancing of MDS on heterogeneous computing systems." NEUROCOMPUTING, 330.2(2019):pp.380-393.

[18] Ayari R, Hafnaoui I, Beltrame G, et al. "ImGA: an improved genetic algorithm for partitioned scheduling on heterogeneous multi-core systems." Design Automation for Embedded Systems, 22.6(2018):183-197.

[19] Gargi A, Kajal V, Santonu S. "Predicting Execution Time of CUDA Kernel Using Static Analysis." IEEE International Symposium on Parallel and Distributed Processing with Applications, 2018:948-955.

[20] Yi J, Zhuge Q, Hu J, et al. "Reliability-Guaranteed Task Assignment and Scheduling for Heterogeneous Multiprocessors Considering Timing Constraint." Journal of signal processing systems for signal, image, and video technology, 81.3(2015):pp.359-375.

Authorized licensed use limited to: University of North Texas. Downloaded on August 08,2022 at 15:46:44 UTC from IEEE Xplore. Restrictions apply.