Computers, Materials & Continua
DOI: 10.32604/cmc.2022.027147
Article

Tech Science Press

Resource Scheduling Strategy for Performance Optimization Based on Heterogeneous CPU-GPU Platform

Juan Fang[1,*], Kuan Zhou[1], Mengyuan Zhang[1] and Wei Xiang[2,3]

[1]Faculty of Information Technology, Beijing University of Technology, Beijing, 100124, China
[2]La Trobe University, Melbourne, VIC, 3086, Australia
[3]James Cook University, Cains, QLD, 4878, Australia
*Corresponding Author: Juan Fang. Email: fangjuan@bjut.edu.cn
Received: 11 January 2022; Accepted: 14 April 2022

Abstract: In recent years, with the development of processor architecture, heterogeneous processors including Center processing unit (CPU) and Graphics processing unit (GPU) have become the mainstream. However, due to the differences of heterogeneous core, the heterogeneous system is now facing many problems that need to be solved. In order to solve these problems, this paper try to focus on the utilization and efficiency of heterogeneous core and design some reasonable resource scheduling strategies. To improve the performance of the system, this paper proposes a combination strategy for a single task and a multi-task scheduling strategy for multiple tasks. The combination strategy consists of two sub-strategies, the first strategy improves the execution efficiency of tasks on the GPU by changing the thread organization structure. The second focuses on the working state of the efficient core and develops more reasonable workload balancing schemes to improve resource utilization of heterogeneous systems. The multi-task scheduling strategy obtains the execution efficiency of heterogeneous cores and global task information through the processing of task samples. Based on this information, an improved ant colony algorithm is used to quickly obtain a reasonable task allocation scheme, which fully utilizes the characteristics of heterogeneous cores. The experimental results show that the combination strategy reduces task execution time by 29.13% on average. In the case of processing multiple tasks, the multi-task scheduling strategy reduces the execution time by up to 23.38% based on the combined strategy. Both strategies can make better use of the resources of heterogeneous systems and significantly reduce the execution time of tasks on heterogeneous systems.

Keywords: Heterogeneous computing; CPU-GPU; Performance; Workload balance

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

1 Introduction

Nowadays, as GPUs show more and more powerful performance in terms of massively parallel computing [1], heterogeneous CPU-GPU architecture has become the current mainstream architecture. However, this structure also faces many challenges [2]. For example, the difference in heterogeneous cores leads to a decrease in resource utilization, which makes the system performance less than expected. Therefore, improving resource utilization in heterogeneous systems is an important research objective in the field of heterogeneous computing [3]. In order to improve the resource utilization and performance of heterogeneous systems, many researchers have found ways to improve the performance by improving a specific application implementation in a specific heterogeneous system. Wozniak et al. [4] tested the execution time of three different implementations of specific applications Hash Join, which proves the superiority of the CPU-GPU heterogeneous architecture. The idea of classification of GPU application based on the kernel structure proposed by Shen et al. [5] is worthy of reference. However, it is only effective for applications with a simple kernel structure. Rizvi et al. [6] have improved the implementation of deep neural convolution network applications and achieved remarkable optimization results. Although these schemes can significantly shorten the execution time, they are applied only to specific applications in specific environments.

At present, Mittal et al. [7] summarized studies on heterogeneous computing. They classify workload balancing strategies into static and dynamic. Next, the paper will discuss the related research of these two kinds of strategies. The static strategy needs to estimate the execution time of the application. Jung et al. [8] proposed a scheme combining static prediction and dynamic inference. The prediction accuracy is affected by the size of tasks, and it is difficult to achieve high prediction accuracy. Alavani et al. [9] proposed a GPU Compute Unified Device Architecture(CUDA) kernel execution time prediction method based on the code analysis. But the predicted result still has more than 20% error compared to the actual running result. Alsubaihi et al. [10] proposed a multi-objective optimization scheme for energy consumption and execution time. When the peak values of power and core temperature are limited, they quantify the benefits of energy consumption and execution time and obtain the comprehensive optimal execution scheme by making full use of the particle swarm optimization algorithm. However, this scheme increases additional overhead, and the optimization effect will be greatly reduced for small applications with short execution times. Li et al. [11] proposed an improved heterogeneous earliest finish time (HEFT) algorithm to improve the utilization of system resources. These tasks are too abstract to take advantage of the specific core characteristics About dynamic workload balancing strategies, Vu et al. [12] found that when workload distribution is applied to irregular applications, the amount of computation cannot be accurately predicted. Belviranli et al. [13] found that processing larger data blocks each time can improve the utilization of GPU, but the workload between CPU and GPU may become unbalanced. Therefore, they have made a comprehensive selection between GPU efficiency and resource utilization and achieved good results. This paper also focuses on the impact of input data attributes on efficiency, but the block size they mentioned refers to the data size, while the block size mentioned in this article affects the data structure. Navarro et al. [14] proposed a scheme that can adapt to most applications, but the search phase for irregular applications may last until the end. This method will degrade performance in individual special applications. Lin et al. [15] proposed a feedback-based workload distribution and frequency adjustment strategy. This strategy does not focus on how to reduce the execution time of the application on a specific core.

However, the effect of these solutions will be affected by the change of the heterogeneous system environment. The research about task scheduling in heterogeneous system pays little attention to the execution efficiency of specific core, which results in that the processing ability of heterogeneous core is not fully developed [16]. Therefore, to improve the performance of CPU-GPU heterogeneous systems, this paper focuses on the resource utilization of the heterogeneous system and the execution efficiency of the application in the specific core. According to the different requirements of single task and multi-task processing scenarios, two resource scheduling strategies are proposed, including single task combination strategy and multi-task scheduling strategy. On this basis, the improved ant colony algorithm is used to make full use of the characteristics of heterogeneous cores to quickly obtain a reasonable task allocation scheme.

2 Basic Method

Heterogeneous systems often encounter situations where a large number of tasks need to be handled [17]. In this case, in order to minimize the execution time of the entire task group, it is necessary to make a reasonable resource scheduling plan through global analysis under the basic conditions of understanding the characteristics of each task in the task group.

The scheduling strategy for a single task is generally based on task data as the operation granularity, while the scheduling strategy for multiple tasks is generally based on the whole task as the operation granularity. The operation granularity of strategies for single-task processing scenarios and multi-task processing scenario is quite different, and the latter is more advantageous when dealing with multiple tasks. The reason is shown in Fig. 1, in this sample, task A executes more efficiently on the GPU, Task B executes more efficiently on the CPU. The length of the rectangle represents the execution time of the task.

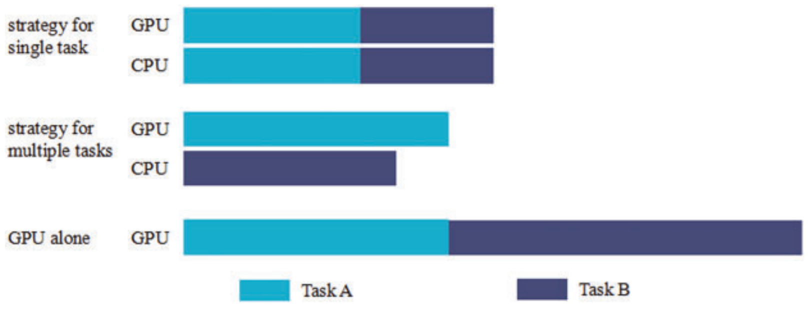


Figure 1: Comparison of special situations when different kinds of strategies deal with multiple tasks

When a task group containing task A and task B is executed using three different scenarios, the total execution time corresponds to the multi-task scheduling strategy, the combination strategy, and GPU alone, from less to more. The reason why multi-task scheduling strategy is better than GPU alone is that extra CPU is used to process some data and fully utilize heterogeneous core processing resources. The reason why multi-task scheduling strategy is better than combination strategy is that when combination strategy executes task A (B), CPU (GPU) processing efficiency is lower and core processing capability is not fully utilized. The multi-task scheduling strategy is to place tasks A and B on the most appropriate efficient cores to execute, without abstracting the heterogeneous cores into a single processing power, to make full use of the characteristics of the heterogeneous cores.

Although the above illustration is a special case, there are only two preconditions for a multi-task strategy to be superior to a combination strategy when dealing with multiple tasks. The first is that the efficiency difference between tasks on the CPU and the GPU cannot be too large. The second is that when there is a certain difference in execution efficiency, most tasks in a task group cannot be better suited to call one of the specific cores for processing. As long as the task groups satisfying these two conditions are processed, the scheduling strategy for multiple tasks can achieve better results than the scheduling strategy for single task. It is necessary to design a multi-task scheduling strategy.

3 Combination Strategy for Single Task

3.1 Workload Balancing Strategy

The innovation of this strategy is to add priority protection measures, this measure can be used to protect the core which can handle tasks faster, thus improving resource utilization and shortening task execution time. In fact, this paper designs two kinds of Workload Balancing (WB) strategies that take advantage of this measure, one is used this measure separately, and another is a combination strategy used this measure and the Block Size (BS) adjustment strategy mentioned earlier. These two strategies share some common processes, which will be explained first in this section.

Assume that the execution time of the application sample on the CPU and GPU are T_{cpu} and T_{gpu}, respectively, and the workload distribution ratio between GPU and CPU is R. The protection measures we propose in this paper is as follows: When the CPU execution time of the sample is far beyond the GPU, the workload distribution ratio R between the GPU and the CPU is increased so that the expected execution time of the CPU is slightly smaller than the GPU. When the CPU execution time is less than the GPU, the workload distribution ratio R between the GPU and the CPU is reduced, so that the expected execution time of the CPU is slightly larger than the GPU. This ensures that the utilization of efficient cores is not affected by other cores when dealing with applications that are suitable for different cores. The adjustment method of the workload distribution ratio is as shown in Eq. (1):

$$R = \begin{cases} \frac{T_{cpu}}{T_{gpu} * \beta} T_{gpu} \leq \frac{T_{cpu}}{\alpha} \\ \frac{T_{cpu}}{T_{gpu}} \frac{T_{cpu}}{\alpha} > T_{gpu} > \alpha * T_{cpu} \\ \frac{T_{cpu} * \beta}{T_{gpu}} T_{gpu} \geq \alpha * T_{cpu} \end{cases} \tag{1}$$

In this equation, α is the efficiency gap threshold, and the execution efficiency gap of the heterogeneous core exceeds the threshold, which will reduce the workload allocated to the inefficient core, sacrificing inefficient utilization to ensure that the utilization of the efficient core is maximized. β is the workload retention factor of the inefficient core, which determines the length of the inefficient core wait time. If the value is set too large, it may cause a short wait for the efficient core. If the value is set too small, the utilization of the inefficient core will be too low. The optimal values of these parameters are not affected by the hardware, but by the application group being processed. Based on the experience of many tests, the proposed system get a set of available parameter settings, so in the subsequent experiments, the values of these two parameters were set to 10 and 0.95, respectively.

3.2 BS-WB Combination Strategy

This combination strategy, which takes the block size adjustment strategy as its main body, is also divided into two phases: the search phase and the stable execution phase. The main changes and adjustments are in the search phase. During the search phase, the change of block size leads to a change in the efficiency of the GPU execution, which in turn causes the execution efficiency ratio on the GPU and CPU to change constantly. This has caused a lot of trouble in maintaining workload balancing among heterogeneous cores. Therefore, each iteration in the search phase requires a retest of the sample to change the distribution ratio. The distribution ratio R in the stable execution phase is obtained as shown in Algorithm 1.

Algorithm 1 Obtaining task distribution ratio in BS-WB combination strategy

```
Input: Task A
Output: Task distribution ratio R
1:  Initialization setting blocksize=128, it=0, AOC=amount of computation per iteration
2:  While blocksize<1024 and it<number of iterations do
3:    AOC *0.01 executes on CPU and AOC *0.01 executes on GPU
4:    Record Tcpu, Tgpu, and use Eq. (1) to calculate task allocation ratio R
5:    AOC *0.98/(1+R) executes on CPU and AOC *0.98*R/(1+R) executes on GPU
6:    Record the blockszie-Tcpu-Tgpu correspondence in the table
7:    Blocksize+64, it+1
8:  End while
9:  Look up the table to get the Tcpu and Tgpu in the iteration rounds with the shortest task
     execution time, and use the Eq. (1) to calculate the task distribution ratio R.
10:   Return task distribution ratio R
```

In order to ensure that the workload balancing strategy does not cause performance degradation due to extreme conditions, protection measures are performed after the sample test, and the distribution ratio is checked and adjusted again to protect the efficient core. In the search phase, the GPU execution efficiency and workload distribution ratio of each block size is updated before each iteration. When the search phase is over, there is no need to perform sample testing for the subsequent iteration. The remaining data is processed by querying the previous records to use the best block size and the corresponding workload distribution ratio to perform the protection measures until all iterations of the application are completed.

4 Multi-Task Scheduling Strategy Based on Ant Colony Algorithm

4.1 Implementation of Multi-task Scheduling Strategy

This section focuses on the implementation of the multi-task scheduling strategy proposed in this paper. Multiple tasks scheduling strategy can be divided into two main steps, the first step is to obtain information about each task, and the second step is to determine the task allocation scheme based on the information obtained. Based on this idea, the multi-task scheduling strategy proposed in this paper is also divided into two components. The first part is a sample test, through which the expected execution time of tasks in the task group on the current CPU/GPU is understood, so as to provide the basic material for the global assignment of tasks. The second part takes the expected execution time as the basis material, and takes execution time as the optimization target, obtains the task allocation scheme quickly through the ant colony algorithm, and completes the execution on the most appropriate core according to the scheme. Next, the implementation steps of these two parts and their design principles are explained in detail.

Nowadays, as the number of cores in heterogeneous systems continues to increase, it becomes more difficult to determine the matching relationship between tasks and cores. In this case, it is not persistent to pursue the optimal task allocation scheme [18]. It has been proved that ant colony algorithm can quickly get task processing scheme suitable for heterogeneous multi-core environments. In addition, in order to further reduce the pressure on task allocation caused by the number of heterogeneous cores, the proposed paper divide all the cores of CPU/GPU heterogeneous system into CPU group and GPU group. Task allocation schemes are formulated in the group instead of individual cores.

The algorithm is divided into two main phases: the initialization phase and the iteration phase. The improvement is in the iteration phase. The initialization phase of the algorithm is explained first.

Step 1: The first step in the initialization phase is to process the initialization of related parameters, such as the number of tasks in the task group and the estimated execution time of each task.

Step 2: The second step in the initialization phase is to initialize the ant colony algorithm-related parameters, which are set as shown in Tab. 1.

Table 1: Initial value of parameters in the ant colony algorithm

Symbol of Parameter	Description of Parameter	Value
N	Number of iterations	100
n	Number of ants	50
g	Pheromone influencing factor	5
	Local decay factor	0.1
	Global decay factor	0.2
m	Pheromones left by ants	0.2
H	Initial pheromone	10

After the initialization phase, the iteration phase of the algorithm begins. In order to adapt to the current environment and meet the needs of getting stable task allocation schemes, this paper makes two key improvements in this phase. The iteration phase of the improved ant colony algorithm is described in detail below.

Step 3: The next task is taken from the task group, recorded as task i, and the probability of being selected for each type of core of the task is calculated. The probability of executing task i on core x,pix is affected by execution time and pheromones. The influence of execution time on selection $Ppix$ is calculated in the Eq. (2).

$$Ppix = \frac{\rho ix}{\sum_{y=1}^{|T|} \rho iy} \tag{2}$$

In which |T| is the collection of currently optional cores, and ρ is the number of pheromones left over by the corresponding path. The influence of pheromones on the selection process $Ptix$ is shown in the Eq. (3).

$$Ptix = \frac{tix}{\sum_{y=1}^{|T|} tiy} \tag{3}$$

In which tix is the estimated execution time of task i on core x. In order to reduce the computational complexity and shorten the execution time of the algorithm, the calculation method of this step has been modified. The probability of choosing to place task i on core x, pix can be calculated by the following Eq. (4)

$$pix = \frac{Ptix + g * Ppix}{1 + g} \tag{4}$$

In which g is pheromone influence factor. P_{ptx} is the pheromone influence calculated by formula (2).

Step 4: Arrange the cells according to the probability of each option, and make a random selection by generating random numbers. Increase the task counter once, and then decide if the task counter reaches the maximum number of tasks. If not, return to step 3. Otherwise, zero the task counter and restart the task acquisition in step c) from the beginning, and proceed to step 5.

Step 5: Update the local pheromone. Assume that the pheromone is ρix. After the ants pass the current cycle, update the pheromone of the corresponding path with the Eq. (5).

$$\rho ix = \rho ix * (1 - Pl) + m \tag{5}$$

Then increase the ant counter once. If the ant counter is 1, the current set of routes selected by the ant and its estimated execution time is directly recorded as the local optimal solution. Otherwise, compared with the local optimal solution currently obtained in the record, the solution with less execution time is considered as the new local optimal solution. Next, it is further determined that if the ant counter is equal to the number of ants set in Tab. 1, the ant counter is cleared and goes to step 6, otherwise, return to step 3 and continue the iteration process.

Step 6: Updates global pheromone with the Eq. (6). Add iteration counter once. If the iteration counter is 1, the scheme obtained in this cycle is recorded as the global optimal solution, and then return to step 3, otherwise go to step 7.

$$\rho i_x = \rho i_x * \left(1 - P_g\right) \tag{6}$$

Step 7: Recalculate the execution time difference between the new solution and the old solution, update the pheromone of the path corresponding to the optimal solution using Eq. (7) as follow, and take the solution with less execution time as the new global optimal solution.

$$\rho i_x = \rho i_x * \left(\frac{2T_{old} - T_{new}}{T_{old}}\right)^k \tag{7}$$

Step 8: Determines whether the iteration counter has reached the set number of iterations N, and returns step c) if it has not, otherwise the solution in the current record is the final result. This improved ant colony algorithm is shown in Algorithm 2.

Algorithm 2 Improved ant colony algorithm

```
Input: Task group to be processed
Output: Task allocation scheme
1:  Complete the initialization of ant colony algorithm-related parameters according to Tab. 1
    (including task information such as the Number of tasks in the task group)
2:  For iterationcounter = 0 to Number of iterations do
3:    For antcounter = 0 to Number of ants do
4:      For taskcounter = 0 to Number of tasks do
5:      Get the task and complete the calculation of the selection probability based on Eqs. (2)–(4)
6:        Generate random number to complete route selection
7:      End for
8:      Updating local pheromones using Eq. (5)
9:      Updating Local Optimal Solutions
```

(Continued)

Algorithm 2 Continued

```
10:         End for
11:         Updating global pheromones using Eq. (6)
12:         Adjust the pheromone of the corresponding path according to the Eq. (7)
13:         Update global optimal solution
14:     End for
15:     Return global optimal solution as task allocation scheme
```

At the end of the above steps, the strategy obtains the required task allocation scheme through the improved ant colony algorithm. The execution process can be completed by assigning tasks to the corresponding core according to the obtained scheme, so as to optimize the execution time and improve the performance of heterogeneous systems.

4.2 Differences between Multi-task Scheduling Strategy and BS-WB Combination Strategy

In terms of optimization objectives, the same point of the two strategies is to reduce the execution time of tasks and improve system performance. The difference is that the design of BS-WB combination strategy focuses on a single task, and the ultimate goal is to complete the task in the shortest time. The multi-task scheduling strategy is designed to deal with the task group composed of multiple tasks. It pays attention to the overall resource management, making up for the shortcomings of the former, which pays too much attention to the characteristics of individual tasks. The design goal is to deal with large-scale tasks in the shortest time.

The core idea of BS-WB combination strategy is to give priority to workload balancing, try to ensure that CPU and GPU are in a working state in the whole process of heterogeneous processing tasks, and ensure resource utilization. The main idea of the multi-task scheduling strategy is to give priority to the work efficiency of the core, and then consider the problem of workload balance after ensuring the maximum execution efficiency of heterogeneous cores.

Finally, the scope of application of both strategies is discussed. Although the combination strategy does work as expected for a small number of tasks, factors such as the proportion of serial code, the difference in task execution efficiency on heterogeneous cores will affect the effectiveness of the strategy. The multi-task scheduling strategy proposed by us can play a significant role in multi-task scenarios as expected. The problem with this strategy is that the sample test incurs additional overhead, which results in the optimization of the strategy being severely affected by the additional execution overhead. In summary, although tasks with large differences in CPU/GPU execution efficiency are still not applicable, flexible use of these two strategies can effectively reduce the execution time for any number of tasks, thereby improving system performance.

5 Experimental Results

5.1 Experiment about Workload Balancing Strategy

For the workload balancing strategy as previously proposed, the experiment first runs eight different applications directly on the GPU and records the execution time, then gradually increases the number of CPU cores enabled in the experiment. As the number of CPU cores participating in task processing increases, the overall efficiency of GPU and CPU changes gradually. For the five experiments, the number of CPU cores enabled was set to 2, 4, 8, 16, and 32, respectively.

The effect of this strategy that the proposed system get from the experiment is shown in Fig. 2 below. Since Nbody, Barnes-Hut, lud, lavaMD, and leukocyte are much more efficient on GPU than on CPU, the effect of keep workload balancing is not obvious, with a reduction of only 0.41%, 5.74%, 3.14%, 0.84%, and 5.46%, respectively. Since the difference in execution efficiency between GPU and CPU is less than an order of magnitude, this strategy works well on Hotspot, resulting in a 14.83% reduction in total execution time. In addition, since the execution time of applications Kmeans and myocyte on the CPU is not significantly different from that on the GPU, this strategy works best in Kmeans and myocyte instances. The execution time of GPU is reduced by 58.01% and 79.12%, respectively, compared with the default GPU method.

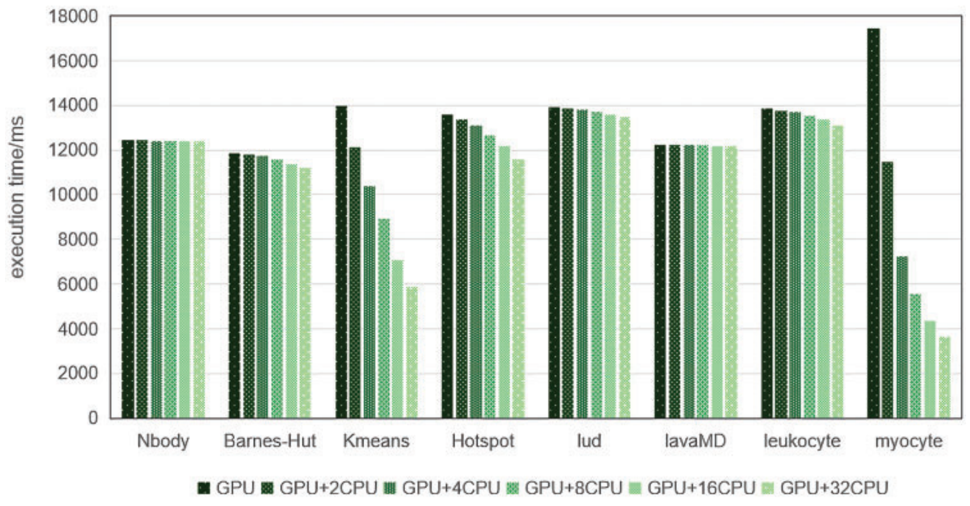


Figure 2: Effect of workload balancing strategy for different CPU configurations and different instances

Through these experiments, it could be found that the effect of workload balancing strategy is always positive no matter what task the system faces due to the protection measures. Overall, the use of this strategy resulted in an average 20.94% reduction in execution time. Looking at the execution time reduction for each instance, it can be found that this strategy can produce a better effect for the task with a few efficiency differences between CPU and GPU.

5.2 Experiment About BS-WB Combination Strategy

Four implementation strategies used in this experiment are as follows: the original GPU implementation (Original), the GPU implementation of the block size adjustment strategy (Block size adjustment), the CPU-GPU heterogeneous implementation of the workload balance strategy that prioritizes the protection of efficient cores (Workload balance) and the BS-WB combination strategy (Combination strategy).

The experiment result is shown in Fig. 3 below. For Nbody and lavaMD, the effect of BS-WB combination strategy is not obvious. The workload balancing strategy is not applicable due to the difference in execution efficiency between CPU and GPU. Since the default block size still applies to the current GPU environment, the block size adjustment strategy is not applicable either. The result is that the execution time is reduced by only 1.01% and 3.18%, respectively, compared with the original GPU method. For Barnes-Hut, Kmeans, lud and myocyte, the combination strategy produced significant optimization results, reducing execution time by 29.38%, 58.50%, 47.25% and 78.82%, respectively.

But most of these benefits come from one of these branch strategies. It does not reflect the superiority of the combination strategy over the sub-strategies. In fact, these experiments on two applications, Hotspot and leukocyte, best reflect the advantages of the combination strategy when compared with its branch strategies. Not only can both branch strategies achieve good results when used separately, but the final execution time is reduced by 19.98% and 14.97% when using the BS-WB combination strategy, which is significantly better than the results of using one of the branch strategies alone.

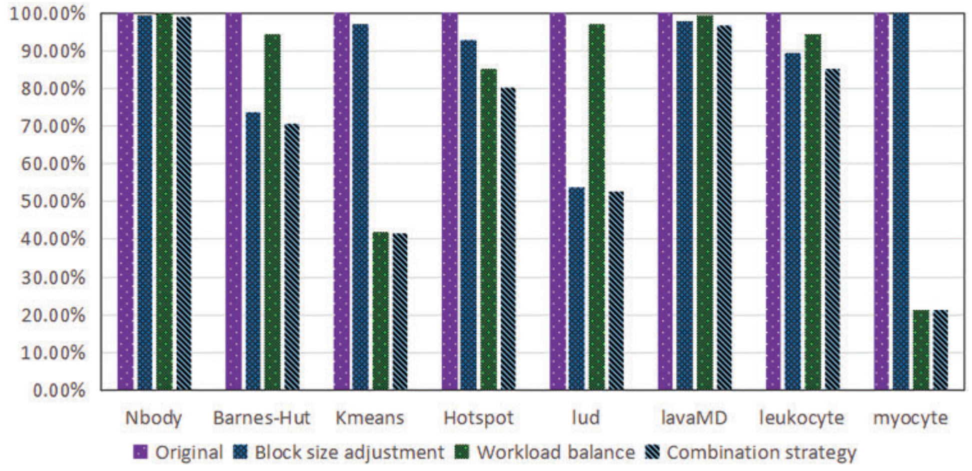


Figure 3: Effect comparison of different strategies in different applications

It could be found that the combination strategy proposed in this paper reduces the execution time of these applications by 29.13% on average compared with the default GPU method, and significantly improves the performance of heterogeneous systems.

5.3 Experiment About Multi-task Scheduling Strategy

To test the effectiveness of the multi-task scheduling strategy proposed in this paper in responding to different scenarios, this paper construct four task groups that have different numbers of tasks and the data size of the tasks varies. The proposed system use these task groups to experiment with this strategy. BS-WB combination strategy was used to process the same work to observe the differences in the scope and effectiveness of these strategies. In addition, the figure corresponding to the experiment contains the following items: Task execution time on CPU when using multi-task scheduling strategy (recorded as MTS-CPU), task execution time on GPU when using multi-task scheduling strategy (recorded as MTS-GPU), show whether the workload distribution is reasonable by comparing MTS-CPU with MTS-GPU; Total time spent using multi-task scheduling strategy (recorded as MTS), by comparing MTS with MTS-CPU&MTS-GPU, the proposed system can get the additional overhead incurred by the process of getting a task processing scheme; Total time spent using the combination strategy (recorded as BW-WB), by comparing BW-WB with MTS, the proposed system can get the difference in the effectiveness of the two strategies when dealing with the same task. Next, the composition of each task group and its reasons are presented, followed by data and diagrams illustrating the actual effect.

For the first experiment, task group A is designed to test the effectiveness of the multi-task scheduling strategy when dealing with scenarios outside its scope of application. This task group is suitable for GPU to handle, which results in the multi-task scheduling strategy that cannot guarantee the workload balance. Task group B is to tests the impact of task group configuration on the optimization effect when workload balance can be ensured. The composition of task group A is shown in Tab. 2, and the composition of task group B is shown in Tab. 3.

Table 2: Configuration of task group A

Composition of Task Group (Task Name-Task Size)	
Nbody-131072∗200	lud-8000∗8
Barnes-Hut-131072∗200	lavaMD-60
Kmeans-494020∗10	leukocyte-100
Hotspot-1024∗200	myocyte-1000∗320

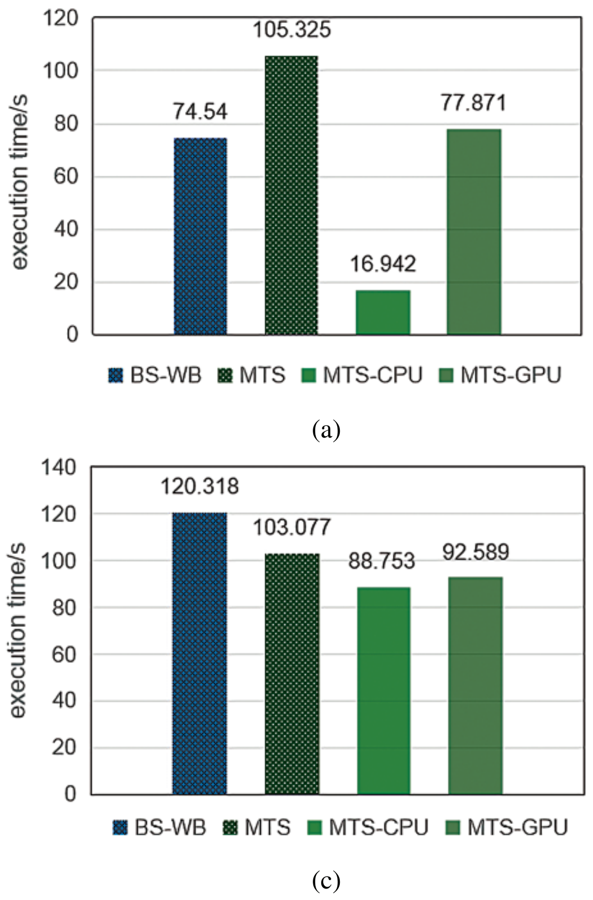
Table 3: Configuration of task group b

Composition of Task Group (Task Name-Task Size)	
Nbody-131072∗200	lud-80000∗8
Barnes-Hut-131072∗200	lavaMD-60∗10
Kmeans-494020∗10	leukocyte-100∗10
Hotspot-1024∗200	myocyte-1000∗320

The specific experiment result is shown in Fig. 4 below. For task group A, the tasks spent 17 s to execute on the CPU, but 77.87 s on the GPU, which results in a serious waste of resources. The execution time of using the multi-task scheduling strategy is 4.46% longer than the BS-WB combined strategy. Taking the cost of the sample test into account, the total time spent is 41.3% longer than the BS-WB combination strategy. It is proved that not all multitasking scenarios are more suitable for multi-task strategy, so the scope of application of the strategy needs to be considered.

For task group B, multi-task scheduling strategy forms a stable workload balance when dealing with this task group. In this case, the task execution time of using the multi-task scheduling strategy is 11.12% less than that of the combined strategy. However, due to the additional overhead caused by sample testing, the total time consumption of the multi-task scheduling strategy is still 7.3% more than that of the combined strategy. The main reason is that half of the CPU's working time is to executing tasks suitable for GPU. It can be seen that for a multi-task scheduling strategy, it is more important for each core to spend most of its time executing tasks suitable for itself than to pursue load balancing unilaterally.

For the other two task groups, task Group C is designed to observe the effect of multi-task scheduling strategy versus BS-WB combination strategy when tasks can be executed on the most appropriate core. Task Group D is the best-case design based on the results of two previous experiments. The task group has a balanced proportion of CPU/GPU tasks, with 32 tasks. This task group is used to test the optimal performance of multi-task scheduling strategies relative to BS-WB combination strategies under optimal conditions. The composition of task group C is shown in Tab. 4, while that of task group D is shown in Tab. 5.



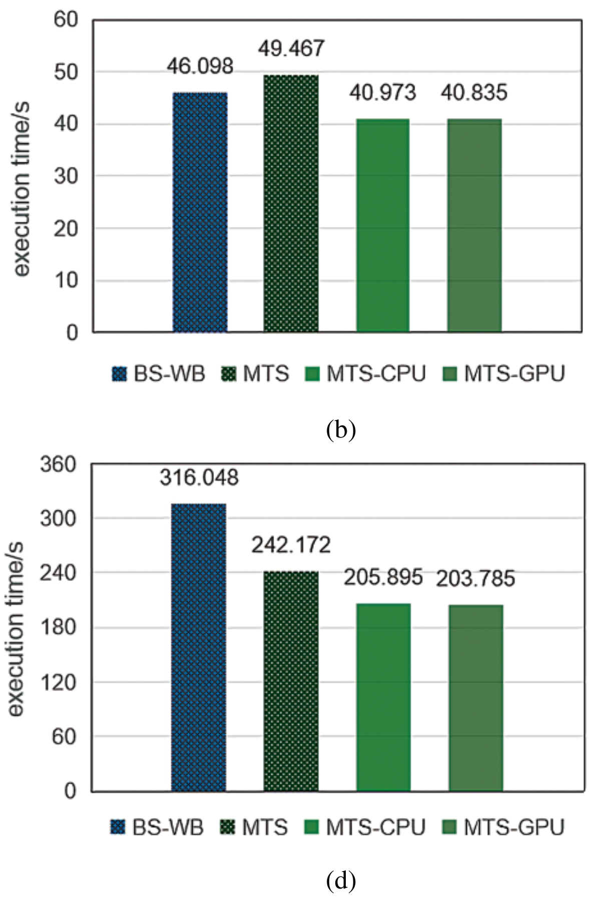


Figure 4: Effect of multi-task scheduling strategy when processing task group A(a), B(b), C(c), D(d)

Table 4: Configuration of task group c

Composition of Task Group (Task Name-Task Size)	
Barnes-Hut-131072∗200	leukocyte-100∗100
Kmeans-494020∗10	myocyte-1000∗320
Hotspot-1024∗200	myocyte-2000∗320
lud-80000∗8	myocyte-3000∗320

Table 5: Configuration of task group d

Composition of Task Group (Task Name-Task Size)		
Nbody-131072∗100	Hotspot-512∗200	leukocyte-70∗10
Nbody-131072∗200	Hotspot-512∗300	leukocyte-80∗10
Barnes-Hut-131072∗100	Hotspot-1024∗100	leukocyte-90∗10

(Continued)

Table 5: Continued

Composition of Task Group (Task Name-Task Size)		
Barnes-Hut-131072∗200	Hotspot-1024∗200	leukocyte-100∗10
Barnes-Hut-131072∗300	Hotspot-1024∗300	myocyte-500∗320
Barnes-Hut-131072∗400	lud-20000∗8	myocyte-1000∗320
Kmeans-494020∗5	lud-40000∗8	myocyte-1500∗320
Kmeans-494020∗10	lavaMD-50	myocyte-2000∗320
Kmeans-494020∗15	lavaMD-60	myocyte-2500∗320
Kmeans-494020∗20	leukocyte-50∗10	myocyte-3000∗320
Hotspot-512∗100	leukocyte-60∗10	

For task group, C, the execution time of tasks using multi-task scheduling strategy is much lower than that using combination strategy, with a relative reduction of 23.05%. And even after considering the sample test time before task allocation, the total time spent using this strategy is still 14.33% less than that of the combination strategy. It is proved that if the workload balance can be maintained while the tasks are executed on the appropriate core, the effect of the multi-task scheduling strategy is much better than that of the combination strategy.

For task group D, the task execution time on the CPU and GPU differs by only about 1%, resulting in an excellent workload balancing with a 34.86% reduction in task execution time compared to the BS-WB combination strategy and a relatively short sample testing time, with a 23.38% reduction in total time spent using the multi-task scheduling strategy compared with the combination strategy. It proves that the multi-task scheduling strategy has a great advantage over the BS-WB combination strategy when dealing with task groups in the optimal scope of application.

In summary, when faced with many different types of task groups, the BS-WB combination strategy and the multi-task scheduling strategy are not only significantly better than those executed by GPU alone but also have their advantages and scope of application. Flexible use of these two strategies can ensure stable and efficient handling of multiple types of task groups by CPU/GPU heterogeneous systems. In particular, when using a multi-task scheduling strategy to handle scenarios within its scope of application, it can reduce the execution time by up to 23.38% compared with the BS-WB combination strategy, which demonstrates the impact and necessity of this multi-task scheduling strategy.

6 Conclusion

This paper proposes several resource scheduling strategies to reduce task execution time based on CPU-GPU heterogeneous systems. First, a combined strategy is proposed for single-task processing, which improves the efficiency of task execution on GPU by adjusting the block size in the linear exploration process. Then, a multi-task scheduling strategy is proposed for the processing of multiple tasks. This strategy considers the efficiency of the core as the main consideration, uses the improved ant colony algorithm to get the task allocation scheme. Finally, the proposed system selected eight instances to test our resource scheduling strategies in a specific heterogeneous CPU-GPU environment. The experimental data demonstrate that the scope of application of these strategies is in line with our expectations and that these strategies can achieve significant optimization results when dealing with tasks within the scope of application. In the future work, on the one hand, the proposed system refers to the commonly used energy consumption optimization method, considers the energy consumption, and refers to the relevant research that considers the energy consumption and execution time together, to further improve the strategy. On the other hand, there is still the possibility of improving the block size adjustment strategy. The proposed system can solve the processing problem of irregular applications by improving the dynamic parallel method, or the work can try to further optimize the block size search process so that it can get the best solution in one iteration. The two strategies of improving size block strategy and adding energy consumption factor can not only play a significant performance optimization effect in their application field but also stably improve the performance of heterogeneous CPU-GPU systems when processing tasks, which is of great research significance.

Acknowledgement: This work is supported by Beijing Natural Science Foundation (4192007), and supported by the National Natural Science Foundation of China (61202076), along with other government sponsors. The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. We would also like to thank all teachers and students in our laboratory for helpful discussions.

Funding Statement: This work is supported by Beijing Natural Science Foundation [4192007], the National Natural Science Foundation of China [61202076], and Beijing University of Technology Project No. 2021C02.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] M. Khairy, A. G. Wassal and M. Zahran, "A survey of architectural approaches for improving GPGPU performance, programmability and heterogeneity," *Journal of Parallel & Distributed Computing*, vol. 127, no. 1, pp. 65–88, 2019.

[2] X. R. Zhang, W. F. Zhang, W. Sun, X. M. Sun and S. K. Jha, "A robust 3-D medical watermarking based on wavelet transform for data protection," *Computer Systems Science & Engineering*, vol. 41, no. 3, pp. 1043–1056, 2022.

[3] H. Zheng and D. Shi, "A multi-agent aystem for environmental monitoring using boolean networks and reinforcement learning," *Journal of Cyber Security*, vol. 2, no. 2, pp. 85–96, 2020.

[4] K. A. Wozniak and E. Schikuta, "Classification framework for the parallel hash join with a performance analysis on the GPU," in *IEEE Int. Symp. on Parallel and Distributed Processing with Applications*, Guangzhou, China, pp. 675–682, 2017.

[5] J. Shen, A. L. Varbanescu, X. Martorell and H. Sips, "Matchmaking applications and partitioning atrategies for efficient execution on heterogeneous platforms," in *Int. Conf. on Parallel Processing*, Beijing, China, pp. 560–569, 2015.

[6] S. T. H. Rizvi, G. Cabodi, D. Patti and G. Francini, "GPGPU accelerated deep object classification on a heterogeneous mobile platform," *Electronics*, vol. 5, no. 4, pp. 88, 2016.

[7] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Computing Surveys*, vol. 47, no. 4, pp. 69, 2015.

[8] N. Jung, H. Baek and J. Lee, "A task parameter inference framework for real-time embedded systems," *Electronics*, vol. 8, no. 2, pp. 116, 2019.

[9] G. Alavani, K. Varma and S. Sarkar, "Predicting execution time of CUDA kernel using static analysis," in *IEEE Int. Symp. on Parallel and Distributed Processing with Applications*, Melbourne, VIC, Australia, pp. 948–955, 2018.

[10] S. Alsubaihi and J. L. Gaudiot, "A runtime workload distribution with resource allocation for CPU-GPU heterogeneous systems," in *IEEE Int. Parallel and Distributed Processing Symposium Workshops*, Lake Buena Vista, FL, USA, pp. 994–1003, 2017.

[11] Y. Z. Li, W. M. Tang and G. X. Liu, "HPEFT for hierarchical heterogeneous multi-DAG in a multigroup scan UPA system," *Electronics*, vol. 8, no. 5, pp. 498, 2019.

[12] T. T. Vu and B. Derbel, "Parallel branch-and-bound in multi-core multi-CPU multi-GPU heterogeneous environments," *Future Generations Computer Systems*, vol. 56, no. 1, pp. 95–109, 2016.

[13] M. E. Belviranli, L. N. Bhuyan and R. Gupta, "A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures," *ACM Transactions on Architecture and Code Optimization*, vol. 9, no. 4, pp. 57, 2013.

[14] A. Vilches, R. Asenjo, A. Navarro, F. Corbera, R. Gran *et al.,* "Adaptive partitioning for irregular applications on heterogeneous CPU-GPU chips," *Proceeded Computer Science*, vol. 51, no. 4, pp. 140–149, 2015.

[15] F. S. Lin, P. T. Liu, M. H. Li and P. A. Hsiung, "Feedback control optimization for performance and energy efficiency on CPU-GPU heterogeneous systems," in *Int. Conf. on Algorithms and Architectures for Parallel Processing*, Granada, Spain, pp. 388–404, 2016.

[16] X. R. Zhang, X. Sun, X. M. Sun, W. Sun and S. K. Jha, "Robust reversible audio watermarking scheme for telemedicine and privacy protection," *Computers Materials & Continua*, vol. 71, no. 2, pp. 3035–3050, 2022.

[17] A. V. Krishna and A. A. Leema, "Etm-iot: Energy-aware threshold model for heterogeneous communication in the internet of things," *Computers Materials & Continua*, vol. 70, no. 1, pp. 1815–1827, 2022.

[18] S. R. Hassan, I. Ahmad, J. Nebhen, A. U. Rehman, M. Shafiq *et al.,* "Design of latency-aware iot modules in heterogeneous fog-cloud computing networks," *Computers Materials & Continua*, vol. 70, no. 3, pp. 6057–6072, 2022.