A Task Parallel Programming Framework Based on Heterogeneous Computing Platforms

Liu Yuan, Yinggang Dong, Yangyang Li, Rui Zhang, and Haiyong Xie

Abstract Along with the development of big data and artificial intelligence, high-performance heterogeneous parallel computing technology has received more and more attention from the industry. On the one hand, heterogeneous computing can significantly improve the computational efficiency. But on the other hand, it can also make the programming more difficult. Such bottlenecks make it harder to give full play to the advantages of heterogeneous hardware. There is currently no comprehensive solution to meet the efficient task scheduling requirements of heterogeneous computing systems. Therefore, this paper introduces a task parallel programming framework based on heterogeneous computing, including the design of programming model, adjusting task granularity, and task scheduling. A ST-HEFT-based static task scheduling method for heterogeneous computing system is proposed to improve

L. Yuan (✉) · Y. Li · R. Zhang
China Academy of Electronics and Information Technology, No. 11 Shuangyuan Road, 100041 Shijingshan District, Beijing, China
e-mail: lyuan@csdslab.net

Y. Li
e-mail: liyangyang@live.com

R. Zhang
e-mail: 50224401@qq.com

L. Yuan · Y. Li · R. Zhang · H. Xie
National Engineering Laboratory for Public Safety Risk Perception and Control by Big Data (NEL-PSRPC), No. 11 Shuangyuan Road, 100041 Shijingshan District, Beijing, China
e-mail: hxie@ustc.edu.cn

Y. Dong
Institute of Network Technology, Beijing University of Posts and Telecommunications, No. 10 Xitucheng Road, 100876 Haidian District, Beijing, China
e-mail: 18710221160@163.com

H. Xie
Advanced Innovation Center for Human Brain Protection, Capital Medical University, No. 119 South Fourth Ring Road West, 100070 Fengtai District, Beijing, China

School of Cyber Science, University of Science and Technology of China, No. 96 Jinzhai Road, 230027 Hefei, Anhui, China

S. M. Thampi et al. (eds.), *Intelligent Systems, Technologies and Applications*,
Advances in Intelligent Systems and Computing 1148,
https://doi.org/10.1007/978-981-15-3914-5_13

computing efficiency. Simulation results show that the framework can obtain better average acceleration ratio. The difficulty of parallel programs for developers can be reduced, and the varying capabilities of heterogeneous components can be fully utilized.

Keywords Heterogeneous computing · Programming model · Task granularity · Task scheduling · DAG

1 Introduction

With the development of information technology, computing platforms such as data and graphics processors have promoted the rapid development of artificial intelligence technology represented by deep neural networks. The heterogeneous hardware such as GPU, FPGA, and CPU shows different advantages on computing capability and power consumption, which can support various specific application domains. However, the heterogeneous computing components are not fully utilized to support these various artificial intelligence applications. In addition, from the perspective of historical development, the development of software framework lags far behind the development of hardware. The lack of efficient task scheduling software might limit the application scenarios of heterogeneous hardware, which can also reduce the efficiency and flexibility of the heterogeneous computing system.

The most prominent research direction of heterogeneous computing [1] in the field of high-performance computing includes GPU-based heterogeneous computing technology represented by Nvidia, AMD/ATI, and FPGA-based reconfigurable computing technology. Most high-performance computing systems today rely on GPU to achieve ultra-high performance. FPGA [2] has high customizable performance and low power consumption, while GPU has a large amount of parallel execution resources and high storage bandwidth. The computational acceleration through the heterogeneous computing acceleration components can often achieve better processing performance than the traditional CPU processing platform. Due to the various performance, cost, and complexity of CPU, GPU, and FPGA, there will be a task scheduling optimal problem to design a mapping approach of heterogeneous components to obtain a combination of programmability, execution performance, design overhead, and power consumption. Therefore, this paper provides a programming framework as a media between developers and various parallel computing devices.

We present a heterogeneous computing model framework as Fig. 1 to enhance the flexibly of heterogeneous system. It elaborates on several aspects of hardware and software programming model, adjustment of task granularity, and task scheduling. PyOpenCL [3] is used to divide the tasks into DAG model. A ST-HEFT-based task scheduling method for heterogeneous system is proposed to improve computing efficiency. The contribution of the paper is threefold.

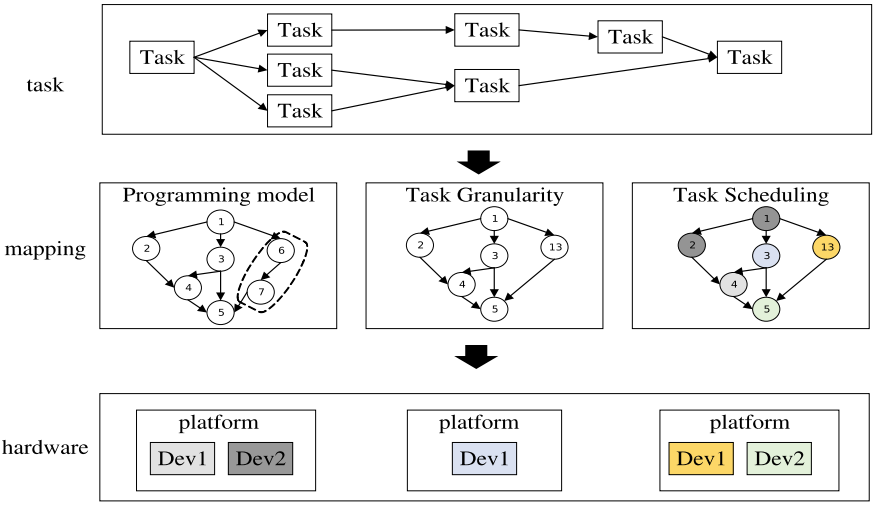

Fig. 1 Heterogeneous computing model framework

1.1 Design Hardware and Software Programming Model

We present a hardware cost model to evaluate the performance of operation results under different hardware architectures, which is designed with reference to literature [4, 5]. The operating effect can be evaluated before the process actually runs. Based on the hardware cost model, some special parallel models have been provided, such as MapReduce, Fork-Join [6]. In order to fully demonstrate the superiority of heterogeneous hardware systems, a directed acyclic graph (DAG) software programming model with task partitioning and task synchronization is proposed based on PyOpenCL [7]. Tasks can be divided into fine granular subtasks efficiently, which can be calculated by the kernel actually running or evaluated by the hardware cost model. Then, the task scheduling problems can be transformed into the model optimization of the DAG graph.

1.2 Design a Task Granularity Adjustment Algorithm

Since the DAG model optimization is a typical NP problem [8], which is very complicated and needs a long running time. This paper introduces a scheme of node combination for the DAG graph. Through the aggregation of multiple nodes reasonably, the complexity of the task scheduling algorithm can be reduced and the scheduling performance can be improved. The mapping between tasks and heterogeneous computing components can be solved much easier.

1.3 Design a Task Scheduling Algorithm

To solve the task scheduling problem, heuristic table task scheduling algorithms, such as HEFT and CPOP [9], and task scheduling algorithms based on random search, such as genetic algorithm [10, 11] and simulated annealing algorithm [12], have been introduced. But the impact of heterogeneous hardware has not been considered enough. In this paper, a task scheduling algorithm for heterogeneous systems is designed to minimize the length of DAG. A new table-driven scheduling algorithm ST-HEFT based on HEFT is proposed to satisfy high real-time requirements.

2 Programming Model

2.1 Hardware Cost Model

To evaluate the performance of operation results, the computing characters of different hardware architectures should be designed. The cost model can be divided into initialization cost (T_{init}), data transmission cost (T_{tran}), and computing cost (T_{com}) as shown in Fig. 2.

Device initialization cost T_{init} refers to the time delay during initialization. When performing large-scale data computing, the initialization time can often be negligible, since the data transmission time and the computing time are much larger. The data transmission cost T_{tran} depends on two aspects: the size of the data to be transmitted and the bandwidth of the data transmission. As an example, the transmission aspects of TITAN-V GPU are shown in Table 1. The time of data transmission can be transfer size divided by the bandwidth.

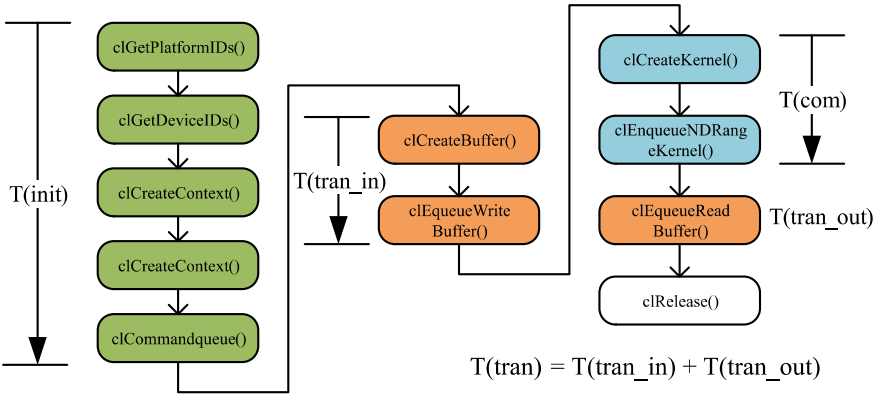

Fig. 2 Hardware cost model based on OpenCL

Table 1 TITAN V bandwidth data

Direction	Transfer size (bytes)	Bandwidth (MB/s)
Host to device	33,554,432	4582.0
Device to host	33,554,432	4241.4
Device to device	33,554,432	463,571.1

And according to the transmission direction of data between host and device, T_{tran} is divided into two parts: $T_{\text{tran\_in}}$ and $T_{\text{tran\_out}}$. The device computing cost T_{com} refers to the application runtime on the device, depending on the data computing time and the memory access time. The actual computing cost of the device can be evaluated by the PyOpenCL execution model kernel runtime, which can be obtained by the PyOpenCL API. The OpenCL API can provide the correct way to test kernel execution time. Through the hardware model, we can evaluate the computing and transmission overhead of different hardware before the task actually runs, which can be used to build the software programming model.

2.2 Software Programming Model

Different computing devices in heterogeneous computing systems usually undertake different computing tasks. Although there are dependencies between devices, most of the boundary of computing can be very clear. Therefore, the task-level programming model is more suitable to represent the heterogeneous computing systems, and a graph-based parallel programming structure can be very useful for reducing the complexity of parallel programs.

The basic idea of the parallel mode adopted in this paper is to divide large tasks into small tasks and then aggregate the small tasks reasonably to get the results. Each application subtask runs on PyOpenCL. PyOpenCL is a Python package of OpenCL that provides various OpenCL interfaces in Python through a library of functions. PyOpenCL greatly simplifies the call to OpenCL and improves programming efficiency. Using the PyOpenCL structure, we can get the transformation overhead by the transformation bandwidth, and the computing overhead can be obtained by the OpenCL kernel's running time. Then, a directed acyclic graph (DAG) software programming model with task partitioning and task synchronization can be proposed. The communication and computing cost, and the relationship between each task for heterogenous hardware can be represented by DAG.

The DAG task graph is represented by $G = (V, E, P, D, B, T)$, and V represents the task nodes set ($V = \{v_1, v_2, v_3, v_4, \ldots\}$), $|V|$ represents the number of task nodes, in the figure below, $|V| = 10$. E represents the edge set ($E = \{e_{i,j} | e_{i,j} = \langle v_i, v_j \rangle, e_{i,j} \in V \times V\}$), $e_{i,j}$ means node v_i to v_j has dependency. P represents the hardware devices set ($P = \{p_1, p_2, p_3, \ldots\}$), $|P|$ represents the kinds of devices. D represents the transfer data size set ($D = \{d_{i,j} | e_{i,j} \in E\}$). B represents the

Table 2 Computation data size of task graph in Fig. 3 [9]

Node	p_1	p_2	p_3
v_1	14	16	9
v_2	13	19	18
v_3	11	13	19
v_4	13	8	17
v_5	12	13	10
v_6	13	16	9
v_7	7	15	11
v_8	5	11	14
v_9	18	12	20
v_{10}	21	7	16

bandwidth set ($B = \{b_{i,j} | b_{i,j} \in P \times P\}$). T represents the computation cost set ($T = \left\{t_{i,j} \middle| v_i \in V, p_j \in P, t_{i,j}\right\rangle 0\}$).

In a heterogeneous environment, the computation cost of the different devices (p_1, p_2, and p_3) for each task node is shown in Table 2. The p_1, p_2, and p_3 represent various hardware processors, such as p_1 as CPU, p_2 as Titan V GPU, and p_3 as T4 GPU. The edge weight represents the communication cost of the two tasks. The communication cost set can be ($C = \{c_{i,j} | e_{i,j} \in E\}$). $c_{i,j}$, which is for transferring data from task v_i (scheduled on p_m) to task v_j (scheduled on p_n), can be defined by:

$$c_{i,j} = d_{i,j} / b_{m,n} \tag{1}$$

As mentioned above, the problem of task scheduling can be transferred to find the nearest and simplest schedule path for the DAG diagram. In order to directly show the differences between the HEFT algorithm in literature [9] and the proposed algorithm, an example of the DAG task graph which shown in Fig. 3 and the computation cost table which shown in Table 2 in literature [9] will be still used in this paper.

Through the hardware model and the software model, the tasks can be divided into DAG models, which can be used to reflect the computing and transmission overhead for the tasks through various hardware. The algorithm expresses the task with the directed acyclic graph DAG by analyzing the task execution characteristics. So, the research on the task parallel programming can be transformed into the research of the DAG graph.

3 Adjust Task Granularity

Although there were some automatic task division methods elaborated [13], the entire DAG diagram was traversed every time, which increases the difficulty of shortest

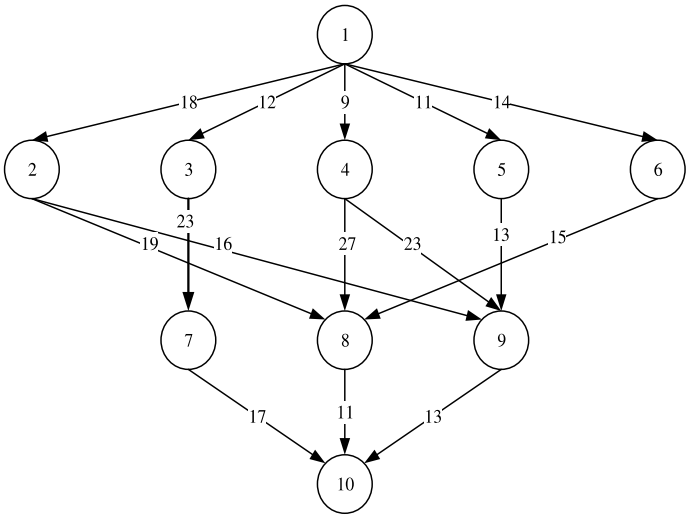

Fig. 3 A sample task graph DAG with 10 tasks [9]

path seeking. At the same time, the optimization of shortest path seeking is a typical NP problem. The complexity of the task scheduling is almost very high.

This paper designs a code adaptive partitioning algorithm based on DAG graph to reduce the complexity and improve the performance of the task scheduling algorithm. A scheme of node combination in the DAG graph is proposed to reduce the numbers of DAG nodes and connections, so that it can effectively reduce the computing cost to find the shortest path of DAG model. The mapping between tasks and heterogeneous computing components can be executed easily.

3.1 Algorithm Description

The major steps in proposed task granularity algorithm will be proposed as follows:

Step 1: Copy. If the node v_0 is the entry node, it will be copied and merged with its all successor nodes to increase the scheduling efficiency.

Step 2: Sort. The ordering of this algorithm is based on the priority according to Eq. (3) and the dependencies of all nodes in the graph. $\text{in}(v_i)$ is the maximum schedule length from the entry node to the v_i, and $\text{out}(v_i)$ is the maximum schedule length from the node v_i to the exit node. The nodes with one precursor node are placed in the to-be-combined set TP, and the nodes in the graph are sorted topologically according to the priority $\text{PRIOR}(v_i)$.

$$T(v_i) = \max_{p_j \in P}\left\{t_{i,j}\right\} \tag{2}$$

$$\text{PRIOR}\ (v_i) = in(v_i) + \text{out}(v_i) + T(v_i) \tag{3}$$

Step 3: Merge. How to choose the appropriate node to merge is the key problem to the division work. To increase the efficiency of partitioning, algorithm must also consider the nodes on non-critical paths. The node v_i with the highest priority will be taken off from the TP.

If it only has one direct precursor node v_j, $v_j \in V_{\text{pre}}(v_i)$, and the node v_j also has only one direct successor node v_i, the node v_i is a candidate point which can be incorporated with v_j.

If it has one direct precursor node v_j and the node v_j has more than one direct successor nodes v_k, $v_k \in V_{\text{succ}}\left(v_j\right)$. Then if Eq. (4) can be satisfied, the node v_j will be merged with the node v_i.

$$\text{PRIOR}(v_i) \geq \max\{\text{PRIOR}(v_k) + T(v_i)\}\left(v_k \in V_{\text{succ}}\left(v_j\right), k \neq i\right) \tag{4}$$

And the priority of all the direct/indirect successor nodes and precursor nodes of v_j should be updated as Eq. (3).

Otherwise, the next node with the second-highest priority is taken off. If there is no node in TP, the operation will be terminated.

3.2 Algorithm Results

Using the task granularity algorithm, the DAG as Fig. 3 can be transferred to Fig. 4. From Fig. 4, the number of nodes can be reduced to eight. It shows that the algorithm can avoid repeating the traversal of the DAG map and can adaptively select the appropriate nodes for the merge operation. The algorithm aims to reduce the complexity of the algorithm and improve the performance of the task scheduling algorithm, which is superior to similar algorithms.

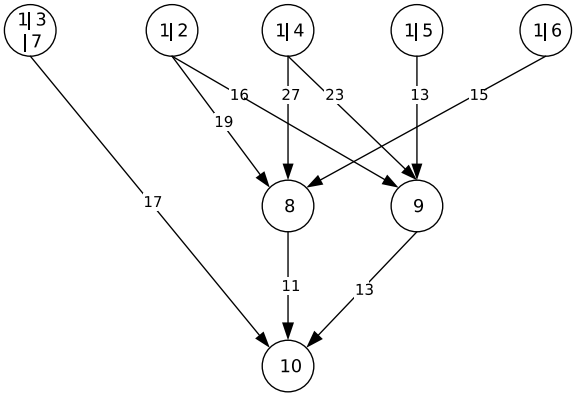

Fig. 4 Results of code adaptive partitioning algorithm

4 Task Scheduling

Task scheduling is the key point of the task parallel programming framework. The task scheduling problem is NP-complete in general case; it is often difficult to obtain the optimal solution, especially in the computing environment on heterogeneous processors. At the same time, it challenges the complex of task scheduling. Therefore, it is of great significance to study and design low-complexity and high-performance task scheduling algorithms. At present, many researchers have proposed solutions for task scheduling problems [14–17], including genetic algorithm, simulated annealing algorithm, and the table-driven scheduling algorithm. The task scheduling algorithms based on genetic algorithm and simulated annealing algorithm are more complex, and the running time is much longer than the heuristic table-driven scheduling algorithm. Therefore, the heuristic table-driven scheduling algorithm is more suitable for the system with high real-time requirements. In literature [18], an IFEFT algorithm which is based on HEFT algorithm is proposed. However, the algorithm does not consider the influence of transmission rate between different devices. The ST-HEFT algorithm not only considers the influence of the transmission rate between devices, but also activates the communication costs of the predecessor nodes and the successor nodes to avoid repeatedly calculating.

4.1 Algorithm Description

The table-driven scheduling algorithm can be divided into two key steps: task sorting and task mapping. After the task priority queue is established according to the algorithm, the tasks in the queue can be, respectively, scheduled.

The earliest start time $\mathrm{EST}(v_i, p_j)$ and the earliest completion time $\mathrm{EFT}(v_i, p_j)$ of task node v_i on device p_j will be defined as follows [9]:

$$\mathrm{EFT}(v_0, p_j) = 0 \tag{5}$$

$$\mathrm{EST}(v_i, p_j) = \max\left(\mathrm{avail}[p_j], \max\left(\mathrm{AFT}(v_k) + c_{k,i}\right)\right)\left(v_k \in V_{\mathrm{pre}}(v_i)\right) \tag{6}$$

$$\mathrm{EFT}(v_i, p_j) = \mathrm{EST}(v_i, p_j) + t_{i,j} \tag{7}$$

$\mathrm{avail}[p_j]$ represents the available time of device p_j, $\mathrm{AFT}(v_k)$ represents the actual end time of task v_k. $V_{\mathrm{pre}}(v_i)$ is the direct precursor tasks of task node v_i.

Then, the task scheduling problems will be trying to find the shortest length of the DAG model Makespan $= \max\{\mathrm{AFT}(v_{\mathrm{exit}})\}$.

So, the table-driven scheduling algorithm can be as follows:

Step 1: Sort. $c_{\mathrm{PC}}(v_i)$ represents the maximum communication cost between v_i and the direct precursor task v_m. $c_{\mathrm{SC}}(v_i)$ represents the maximum communication cost

between the task v_i and the direct successor task v_n. Here, we increase the ratio of the maximum communication value of the precursor node to the total communication value (10) and activate it by the step function (11). And according to Eq. (12), the w_{total} value of task v_i on each device p_j is calculated. Rank(v_i) represents the average of the $w_{\text{total}}(v_i, p_j)$ computed by task v_i on all devices. All tasks in the task queue will be sorted in descending order of rank(v_i).

$$c_{\text{PC}}(v_i) = \max\{c_{i,m}\} \quad (v_m \in v_{\text{pre}}(v_i)) \tag{8}$$

$$c_{\text{SC}}(v_i) = \max\{c_{i,n}\} \quad (v_n \in v_{\text{succ}}(v_i)) \tag{9}$$

$$u = c_{\text{PC}}(v_i)/(c_{\text{PC}}(v_i) + c_{\text{SC}}(v_i)) \tag{10}$$

$$r = \begin{cases} 0, u < 0.5 \\ 1, u \geq 0.5 \end{cases} \tag{11}$$

$$\begin{aligned} w_{\text{total}}(v_i, p_j) = {} & r \times c_{\text{PC}}(v_i) + (1 - r) \times c_{\text{SC}}(v_i) \\ & + \max\{w_{\text{total}}(v_k, p_j) + t_{i,j}\}(v_k \in v_{\text{succ}}(v_i)) \end{aligned} \tag{12}$$

$$\text{Rank}(v_i) = \frac{1}{|P|} \times \sum_{p_j} w_{\text{total}}(v_i, p_j) \tag{13}$$

Step 2: Task mapping. Every time select the task at the head of the task queue. Then according to Eq. (14), define the maximum communication cost LH between task v_i and the exit task on each device. And allocate the task to the device with the minimal product ST of earliest completion time EFT and the longest communication cost LH.

$$\text{LH}(v_i, p_j) = \begin{cases} w_{\text{total}}(v_i, p_j) - t_{i,j} - c_{\text{PC}}(v_i), & r = 0 \\ w_{\text{total}}(v_i, p_j) - t_{i,j}, & r > 0 \end{cases} \tag{14}$$

$$\text{ST}(v_i) = \min\{\text{EFT}(v_i, p_j) \times \text{LH}(v_i, p_j)\} \tag{15}$$

4.2 Algorithm Results

As an illustration, Fig. 5b presents the schedules obtained by the IFEFT algorithm and simulated annealing (SA) algorithm for the sample DAG of Fig. 3. The simulated annealing algorithm determines a solution by random search and then tries to jump out of the local optimal solution to find the approximate optimal solution. The SA

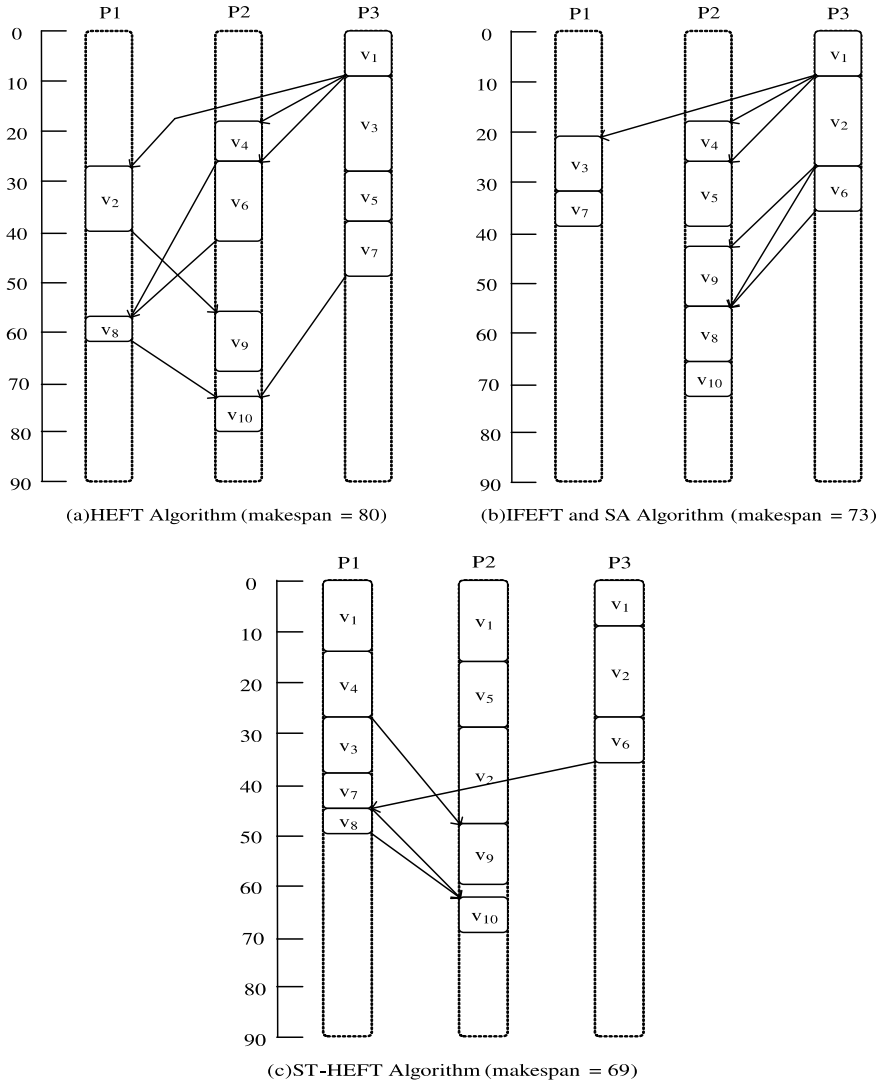

Fig. 5 Scheduling of task graph in Fig. 4 with HEFT, IFEFT, SA, and ST-HEFT algorithms

algorithm is added to the test, which in order to compare the result of heuristic table schedule with random search schedule.

Figure 5c presents the schedules obtained by the ST-HEFT algorithm for the sample DAG of Fig. 3. And since the ST-HEFT algorithm is based on task replication, task v_1 is copied to device p_1 and device p_2, and task v_2 is copied to device p_2 thereby eliminating communication time. The schedule length, which is equal to 69, is shorter

than the schedule length of HEFT algorithm which uses average transmission rate shown in Fig. 5a.

The complexity of ST-HEFT algorithm is only $O\left(|E|^2 \times |P|\right)$, which is the same as HEFT algorithm, but SA is $O(|E| \times a \times b \times \log(-\alpha))$. a, b are the number of outer loops and inner loops of the SA algorithm, respectively; α is temperature change rate of the SA algorithm ($\alpha < 1$). So, table schedule algorithm is more suitable for systems with static task-scheduling requirements.

4.3 Performance Test Experiments

In this section, we present the comparative evaluation of ST-HEFT algorithm and HEFT algorithm. And the random search algorithm (simulated annealing experiment) is added to the performance test, which in order to compare the performance of heuristic table schedule with random search schedule. For this purpose, comparison metric is defined firstly, and we randomly generated application graphs to test these algorithms.

Finally, we can get the results of each algorithm by doing three following sets of experiments.

4.3.1 Comparison Metric

The performance comparisons of the algorithm are based on the metric [18]:

$$\text{speedup} = \min\left\{\sum_{v_i \in V} w_{i,j}\right\} \Big/ \text{Makespan} \tag{16}$$

Numerator $\min\left\{\sum_{v_i \in V} w_{i,j}\right\}$ is the sequential execution time which is computed by assigning all tasks to a single processor that minimizes the cumulative of the computation costs. According to Eq. (16), the larger the speedup, the better the performance of the algorithm.

4.3.2 Randomly Generate DAG Graph

In order to randomly generate DAG graph and computation cost, some parameters are needed to set.

1. The number of tasks in the DAG graph ($|V|$).
2. Communication to computation ratio (CCR). It represents the ratio of the average communication cost to the average computation cost.
3. The number of devices ($|P|$).

In each experiment, the values of DAG parameters are assigned from the corresponding sets given below:

- $\text{SET}_{|V|} = \{20, 40, 60, 80, 100\}$.
- $\text{SET}_{\text{CCR}} = \{0.1, 0.3, 0.5, 1.0, 2.0\}$.
- $\text{SET}_{|P|} = \{2, 4, 6, 8, 10\}$.

Then, in each set of experiments, we randomly generate 50 DAG graphs for each parameter and take the average of fifty results.

4.3.3 Performance Results

Experiment 1: Firstly, the impact of the number of DAG nodes on each algorithm is tested. In this experiment, the performances of the algorithms are compared with respect to various graph sizes (see Fig. 6). The average calculation time of each DAG model is 40, and the average communication time is 32 which is 80% of the average calculation time (CCR = 0.8), the number of devices is 5 ($|P| = 5$).

According to the experimental results shown in Fig. 6, the average speedup of ST-HEFT is 4% higher than HEFT algorithm and 17% higher than SA algorithm.

Experiment 2: the impact of the number of devices on each algorithm is tested. In this experiment, the number of DAG nodes is 20 ($|V| = 20$), CCR = 0.8.

According to the experimental results shown in Fig. 7, the average acceleration ratio of ST-HEFT is 4% higher than HEFT algorithm and 9% higher than SA algorithm.

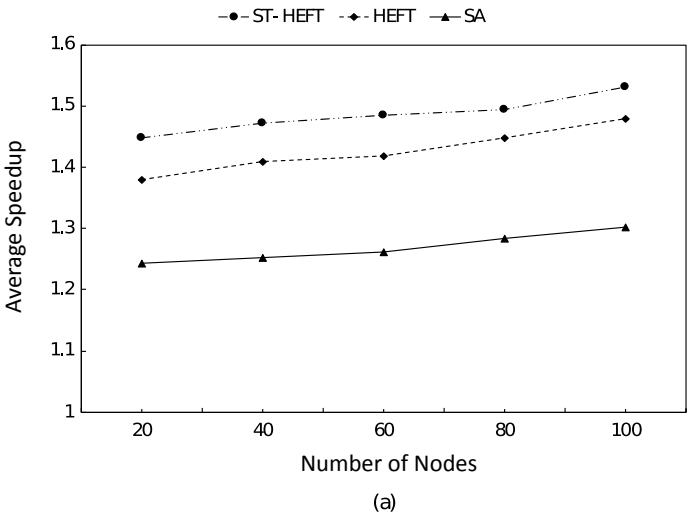

Fig. 6 Performance test for the number of DAG nodes

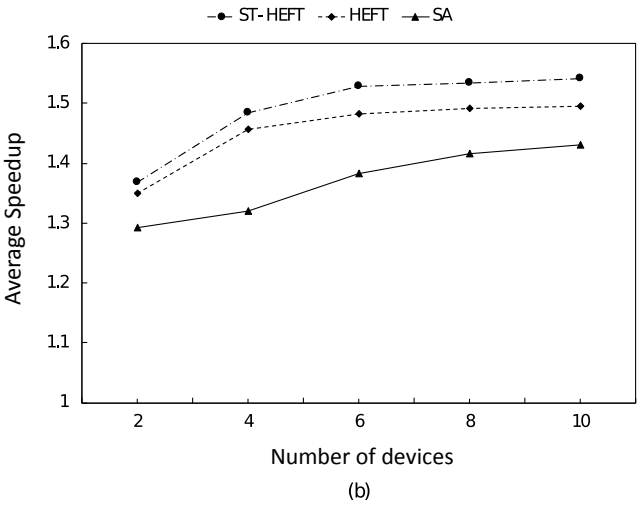

Fig. 7 Performance test for the number of devices

Experiment 3: the impact of the communication to computation ratio on each algorithm is tested. In this experiment, the number of DAG nodes is 20 ($|V| = 20$), the number of devices is 5 ($|P| = 5$).

According to the experimental results shown in Fig. 8, the average accelera-

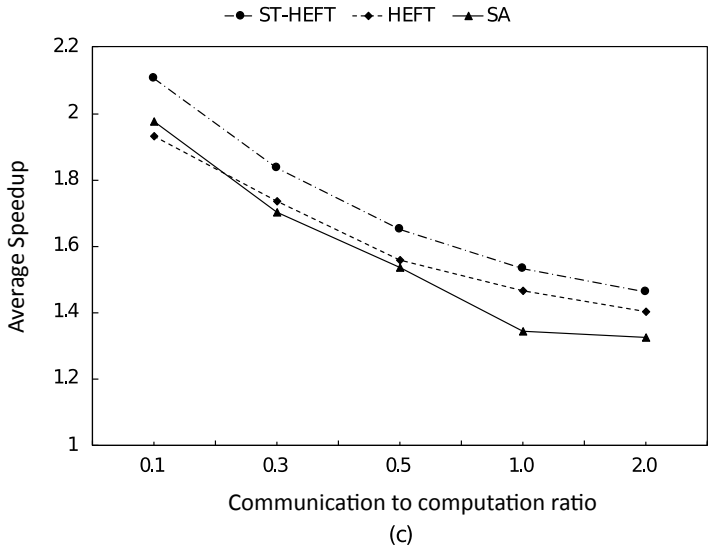

Fig. 8 Performance test for CCR

tion ratio of ST-HEFT is 6% higher than HEFT algorithm and 9% higher than SA algorithm.

Combined with the above three experiments, we can get: The experimental results show that the average performance of the ST-HEFT algorithm is better than that of the HEFT algorithm. And the performance of the heuristic table schedule algorithm is better than the performance of the simulated annealing algorithm.

5 Conclusion

In this paper, we propose a task parallel programming framework to improve the efficiency of heterogeneous computing. Through the design of programming model, task granularity adjustment, and task scheduling, a more efficient and flexible task scheduling framework is provided. We present a hardware and software cost model to divide the tasks into a DAG model. Then, a task granularity adjustment method is introduced to reduce the complexity of the task scheduling. A new table-driven scheduling algorithm based on ST-HEFT is proposed to satisfy high real-time requirements for heterogeneous systems. Simulation tests show that the ST-HEFT can reduce the scheduling length and improve the average speedup obviously. The proposed algorithm can obtain a better average performance. It shows that the framework can reduce the development threshold of parallel computing programs for developers and maximize the utilization of the capabilities of various computing devices in heterogeneous computing platforms.

Acknowledgements This work was supported by the CETC Joint Advanced Research Foundation (Grant No. 6141B08080101, 6141B08010102) and the CETC Industry Development Funds (Grant No. AI20190603015).

References

1. Xie, G., Li, R., Li, K.: Heterogeneity-driven end-to-end synchronized scheduling for precedence constrained tasks and messages on networked embedded systems. J. Parallel Distrib. Comput. **83**, 1–12 (2015)
2. Kaptanoglu, S., Bakker, G., Kundu, A., et al.: A new high density and very low cost reprogrammable FPGA architecture In: FPGA'99, pp. 3–12 (1999)
3. Stone, J.E., Gohara, D., Shi, G.: OpenCL: a parallel programming standard for heterogeneous computing systems. Comput. Sci. Eng. **12**(3), 66–73 (2010)
4. Dan, Z., Rongcai, Z., Lin, H., et al.: A parallelization cost model for GPU. In: International Conference on Computer and Communication Technologies in Agriculture Engineering (cctae 2010) (2010)
5. Zhang, D., Zhao, R.C., Han, L., et al.: A Parallelization cost model for FPGA. Adv. Mater. Res. **181–182**, 623–628 (2011)
6. Wang, L., Cui, H.M., Chen, L., et al.: Research on task parallel programming model. Ruanjian Xuebao/J. Softw. **24**(1), 77–90 (2013)

7. Klöckner, A., Pinto, N., Lee, Y., et al.: PyCUDA and PyOpenCL: a scripting-based approach to GPU run-time code generation. Parallel Comput. **38**(3), 157–174 (2012)
8. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(3), 384–393 (1975)
9. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
10. Omara, F.A., Arafa, M.M.: Genetic algorithms for task scheduling problem. In: Foundations of Computational Intelligence, vol. 3, pp. 479–507. Springer, Berlin, Heidelberg (2009)
11. Hou, E.S.H., Ansari, N., Ren, H.: A genetic algorithm for multiprocessor scheduling. IEEE Trans. Parallel Distrib. Syst. **5**(2), 113–120 (1994)
12. Kalashnikov, A.V., Kostenko, V.A.: A parallel algorithm of simulated annealing for multiprocessor scheduling. J. Comput. Syst. Sci. Int. **47**(3), 455–463 (2008)
13. Ayed, M., Gaudiot, J.L.: An efficient heuristic for code partitioning. Parallel Comput. **26**(4), 399–426 (2000)
14. Valls, V., Ballestin, F., Quintanilla, S.: A hybrid genetic algorithm for the resource-constrained project scheduling problem. Eur. J. Oper. Res. **185**(2), 495–508 (2008)
15. Arabnejad, H., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. Parallel Distrib. Syst. **25**(3), 682–694 (2013)
16. Yoo, M., Gen, M.: Scheduling algorithm for real-time tasks using multiobjective hybrid genetic algorithm in heterogeneous multiprocessors system. Comput. Oper. Res. **34**(10), 3084–3098 (2007)
17. Topcuoglu, H., Hariri, S., Wu, M.Y.: Task scheduling algorithms for heterogeneous processors. In: Proceedings Eighth Heterogeneous Computing Workshop (HCW'99), pp. 3–14. IEEE (1999)
18. Li, Y., Zhou, C., Wang, Q.: A new list scheduling algorithm in heterogeneous distributed computing environment. Comput. Eng. **44**(8), 43–47 (2018). https://doi.org/10.19678/j.issn.1000-3428.0048099