An optimized workflow scheduling algorithm on CPU-GPU heterogeneous systems

An optimized workflow scheduling algorithm on CPU-GPU heterogeneous systems

Haoyang Ma[1], Juan Fang[1]

[1]Faculty of Information Technology, Beijing University of Technology

fangjuan@bjut.edu.cn

Abstract. Aiming at the workflow scheduling problem on CPU-GPU heterogeneous systems, this article proposes a workflow scheduling algorithm that optimizes task priority and processor selection phase. This article focuses on the relationship between processor processing time and task acceleration ratio, and uses optimistic finish time table. This article estimates each task's communication cost and computation cost on the basis of each task's acceleration ratio ri. Then the algorithm calculates the task's priority ranku based on communication cost and computation cost. In processor selection phase, this article takes the Earliest Finish Time (EFT) difference between the processor with the fastest task and the processor selected based on Heterogeneous Earliest Finish Time (HEFT) algorithm as the judgment condition k. If the difference between the earliest completion time of all child tasks is greater than k, the algorithm selects the processor selected by the HEFT algorithm; otherwise, the algorithm selects another one. This article uses the computational shared facility's kernel timer for simulation experiment, which bases on the University of Manchester's high performance computing cluster. Based on the simulation results, the proposed algorithm can reduce the maximum completion time and energy consumption, and improve the acceleration ratio. Compared with the HEFT algorithm, the task maximum completion time and energy consumption are reduced by 10% and 5% respectively, and the acceleration ratio is improved by 3%.

1. Introduction

1.1 Overview of heterogeneous system

Nowadays, the heterogeneous system based on CPU-GPU has increasingly become the main implementation architecture of heterogeneous computing [1]. In recent years, with the gradual upgrading of manufacturing technology, the computer architecture is also undergoing earth-shaking changes. Moore's Law, influenced by the laws of physics, manufacturing processes and other factors, has shifted the emphasis from increasing the number of transistors in a single core processor to increase the amount of processors that can be consolidated into a core plate. Influenced by this, a mainstream processor architecture also progresses from single-core architecture to multi-core architecture. In 2005, Intel and AMD respectively released their own dual-core architecture processors, which also marked the formal beginning of the evolution of processor architecture. A high performance computer in the field of high performance computing is basically composed of thousands of computing nodes. Increasingly, they are heterogeneous, including uncertain number GPUs as accelerators apart from multicore CPUs. The Summit that topped the list in 2018, consists of 4,608 computing nodes, each of which uses a CPU-GPU heterogeneous computing system similar to Titan's. The Summit consists of

two POWER9 CPUs and six Nvidia Tesla V100 acceleration cards. Therefore, this article mainly studies the heterogeneous system based on CPU-GPU.

Heterogeneous processors have different internal core structures, so different types of computing tasks can be assigned to different processor cores for parallel processing [2]. In 2005, the first example of a new series of processors for the broadband era, the Cell processor was born. It was jointly launched by IBM, Sony and Toshiba, and was used in the early days for gaming systems (PlayStation3TM) and computing accelerators, as well as various embedded applications. The Cell processor has a general-purpose processor running POWER instructions and eight auxiliary processors connected by a high-speed bus. The architectural diagram for the Cell processor is shown in Figure 1:

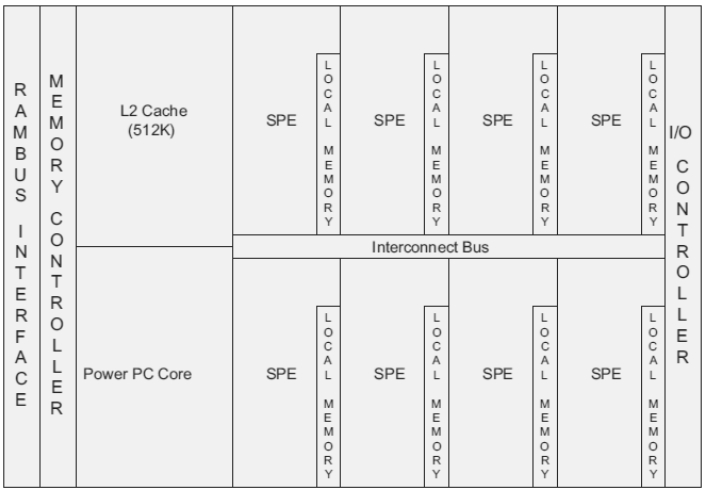

Figure 1. Cell Processor Architecture.

1.2 Related Work

According to the correlation between the scheduled tasks, task scheduling can be divided into two types: independent task scheduling and related task scheduling [3]. Task scheduling under heterogeneous multi-core processor platform can be divided into static task scheduling and dynamic task scheduling [4]. Static task scheduling [5] means that tasks are assigned to fixed processor cores on the premise of satisfying the load balancing of hardware resources and communication expenses. Dynamic task scheduling [6] means that tasks can be assigned to any processor kernel according to resource utilization. The scheduling problem of related tasks under heterogeneous multi-core processor platform. The main task is to map subtasks to multiple processor cores to achieve scheduling goals such as minimizing task completion time. Therefore, in the process of task scheduling, the difference of computing power of heterogeneous processor cores should be considered. At the same time, this article also need to consider the data communication between the processor cores. Therefore, the scheduling problems of related tasks all belong to NP-complete problems [7].

DAG task scheduling algorithms mainly include HEFT [8], CPP [9] and PEFT [10]. DAG task scheduling algorithm on heterogeneous multi-core processor platform mainly includes three types, which are table scheduling algorithm, cluster scheduling algorithm and replication-based scheduling algorithm.

Table scheduling algorithm is the most classic DAG scheduling algorithm. Table scheduling algorithms include static scheduling algorithms and dynamic scheduling algorithms. The basic idea of static scheduling algorithm is: according to the relationship between all tasks in DAG, the priority among tasks is determined with some strategy and a scheduling list is constructed. The task with the highest priority is then assigned the resource that will enable it to complete the fastest and scheduled execution begins, repeating the above steps until all tasks in the DAG are scheduled. After executing each task, the dynamic table scheduling algorithm recalculates the priority of the task according to the current situation of the processor kernel and the dependency relationship between the tasks. The static scheduling algorithm does not recalculate the priority, while the dynamic scheduling algorithm needs to

recalculate the priority. Common table scheduling algorithms include HEFT (Heterogeneous Finish Time) algorithm and CPP algorithm.

This article will design a more practical and efficient workflow scheduling algorithm. First, heterogeneous systems and existing algorithms are introduced. Because the task scheduling process mainly considers the priority setting and processor selection. Based on HEFT algorithm, the task priority setting and processor selection in the scheduling process are improved, and optimized for CPU-GPU heterogeneous system.

2. System model

2.1 Heterogeneous system model

This article uses the heterogeneous system model, which bases on the computational shared facility. When a DAG E is scheduled on a target platform P, and platform has two types of processing resources, P_C CPU resources and P_G GPU resources. The relationship between them is shown as follows

$$P = P_C + P_G \tag{1}$$

The article considers CPU cores separately, but considers the entire GPU as discrete [11], a node consisting of two GPUs and two quad-core CPUs will be considered as two GPU resources and eight CPU resources.

This article assumes that all tasks are atomic and cannot be aggregated into larger tasks and partitions across resources. All processors can only perform one task at any one time. All tasks can be performed.

2.2 Priority calculation model

To make better use of the CPU and GPU, this article need to use the acceleration ratio r_i of tasks to calculate the weighted average. For each task, the execution cost is expressed as

$$\overline{w_i} = \frac{w_C(t_i)P_C + r_i w_G(t_i)P_G}{P_C + r_i P_G} \tag{2}$$

for the communication cost between task edges, it can be expressed as:

$$\overline{c_{ij}} = \frac{A_{CC} * c_{ij}(C,C) + A_{CG}\left(r_i c_{ij}(G,C) + r_j c_{ij}(G,C)\right) + r_i r_j A_{GG} * c_{ij}(G,G)}{(P_C + r_i P_G) * (P_C + r_j P_G)} \tag{3}$$

For all the tasks in the DAG, this article recursively compute their $rank_u$, starting with the end task.

$$rank_u(t_i) = \overline{w_i} + \max_{t_k \in Ch(t_i)} \left(\overline{c_{ij}} + rank_u(t_j)\right) \tag{4}$$

2.3 Processor selection model

This article needs to assign each task to the processor with the shortest processing time during the processor selection phase. $avaid(p_m)$ is the earliest time when the processing resource p_m actually freely executes the task t_i; $AFT(t_k)$ is the time when the task t_k actually completes the execution. The earliest start time of a task on the processor p_m is

$$EST(t_i, p_m) = \max\{avaid(p_m), \max_{t_k \in Pa(t_i)}(AFT(t_k) + c_{ki})\} \tag{5}$$

The earliest completion time of task t_i on processor p_m is

$$EFT(t_i, p_m) = w_{im} + EST(t_i, p_m) \tag{6}$$

This article needs to use optimistic finish time (OFT), which is the earliest time that all tasks can be completed, regardless of all resource contention. For $p, p' \in \{C, G\}$, this article builds the OFT value table by DAG and set $OFT(t_i, p) = w_p(t_i)$. If t_i is an entry task, the recursive calculation is performed. OFT can be expressed as

$$OFT(t_i, p) = w_p(t_i) + \max_{t_j \in Pa(t_i)} \{\min_{p'} \{OFT(t_i, p') + \delta_{pp'} c_{ij}(p, p')\}\} \quad (7)$$

This article uses the earliest finish time to schedule task t_i to the processing resource p_m. in the priority list, only if p_m is not the fastest resource type. Calculates the difference between the earliest completion time on the processor p_m and p_f

$$k \coloneqq EFT(t_i, p_f) - EFT(t_i, p_m) \quad (8)$$

p_m is assumed to be $T_m \in \{C, G\}$.By assuming that each task t_i subtasks t_j are arranged on the type of resource t_j, minimize the OFT, waiting for the other parent task completion and ignore the potential need. this article estimates $E(Ch(t_i)|p_m)$, which is the earliest completion time of all subtasks, can be expressed as

$$E(Ch(t_i)|p_m) = \max_{t_j \in Ch(t_i)} (EFT(t_i, p_m) + c_{ij}(T_m, T_j) + w_{T_j}(t_j)) \quad (9)$$

if (10) is true, scheduling task t_i on p_m; otherwise it's scheduled on p_f.

$$k > E(Ch(t_i)|p_m) - E(Ch(t_i)|p_f) \quad (10)$$

2.4 Energy Consumption Model

During the execution of a DAG task, the processor may be in the state of executing a task or waiting to complete. Therefore, the total energy consumption of DAG task can be divided into two types: active energy consumption E_{active} and idle energy consumption E_{idle} [12,13,14]

$$E_{total} = E_{active} + E_{idle} \quad (11)$$

The actual execution time and energy consumption of each task are related to the assigned processor and execution frequency. The activity energy consumption is the accumulation of the execution energy consumption of all tasks.

$$E_{active} = \sum_{i=1}^{n} (P_i * t_i) \quad (12)$$

Idle energy consumption is the energy consumption of all processors in idle time before the completion of the application of the computing system, which can be express as

$$E_{idle} = \sum_{j=1}^{n} (P_{j,ilde} * t_{j,idle}) \quad (13)$$

There is no dependency relationship among the constraint n tasks, and each processor is isomorphic and has only one idle frequency and one active frequency. The corresponding power consumption is P_{active} and P_{idle}, respectively, then the total energy consumption E_{total} is

$$E_{total} = P_{active} * \sum_{i=1}^{n} t_i + P_{idle} * (makespan_{opt} * m - \sum_{i=1}^{n} t_i) \quad (14)$$

3. Optimization algorithm

Based on the above models, this article can conclude the algorithm:

Algorithm 1: Optimization Workflow Scheduling Algorithm

1: Using (2) to set the computation cost of all tasks

2: Using (3) to set the communication cost of all edges

3: Using (7) to compute the OFT table for all tasks

4: Using (4) to compute $rank_u$ for all tasks

5: Rank the tasks in descending order according to $rank_u$

6: for each task do

7: for each resource p_k do

8: Compute $EFT(t_i, p_k)$ using (5) and (6)

9: end

10: Set the minimum EFT in computing resources to p_m

12: if $w_{im} \neq \min\left(w_C(t_i), w_G(t_i)\right)$

13: Set the minimum value of EFT or w_{ik} in the computing resource to p_k

14: Compute s_m using (8)

15: Compute $E(Ch(t_i)|p_m)$and $E\left(Ch(t_i)\middle|p_f\right)$using (9)

16: if (10)

17: Schedule t_i on p_m

18: else

19: Schedule t_i on p_f

20: end

21: end

22: end

4. Experiments and results

4.1 Experimental Configuration

In the experiment, this article considers two simulation target platforms: a single GPU consisting of one GPU and one octa-core CPU, and multiple GPUs consisting of eight GPUs and four octa-core CPUs. At

the same time, this article designed two different groups of DAGs in the experiment. The first consists of eight DAGs, ranging from 35 to 1280 tasks. In particular, DAGS is based on the generic Cholesky decomposition implementation of the Tile Matrix, which uses the GEMM, SYRK, and TRSM BLAS cores, as well as the POTRF LAPACK routines [15]. The processing time of all tasks is the average of the actual time of the task kernel. The second is to set up a topology based on the standard task set [16], which contains 90 DAG, and at the same time can randomly generate CCR. This article calculates the corresponding CPU execution time through the method in [17]. And three copies of each DAG will be made to randomly generate the communication cost to keep the CCR within three different intervals.

4.2 Analysis of Results

As can be seen from Figure 2, in a single GPU platform, with the number of tasks increases, the maximum completion time of the HOFT-CO algorithm gradually decreases compared with the maximum completion time of the HEFT algorithm with the number of general tasks, and finally stays within the range of 5% reduction. In multi-GPU platforms, the HOFT-CN algorithm has always performed better than the HEFT algorithm, with a maximum reduction of 30% and an average reduction of 10% in the maximum completion time, compared with PEFT algorithm, which can only achieve an average reduction of 3%. In Figure 3, all 90 randomly generated DAG graphs are shown. Compared with multi-GPU platforms, the single GPU platform has less improvement. It can also be intuitively seen from the figure that the number of DAG reduced by time is much greater than that increased by time. In conclusion, by comparing with HEFT and PEFT, HOFT-CN has a better optimization in the maximum completion time.

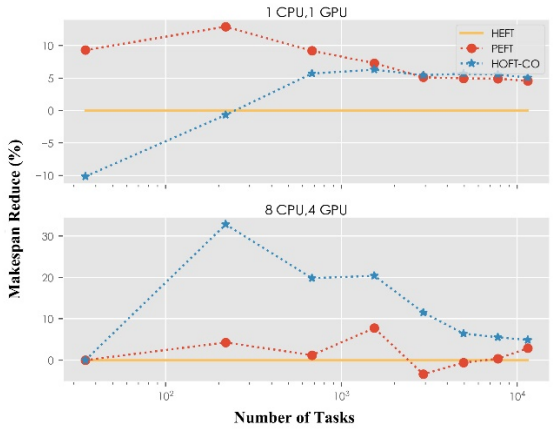

Figure 2. Comparison of makespan for Cholesky DAGs

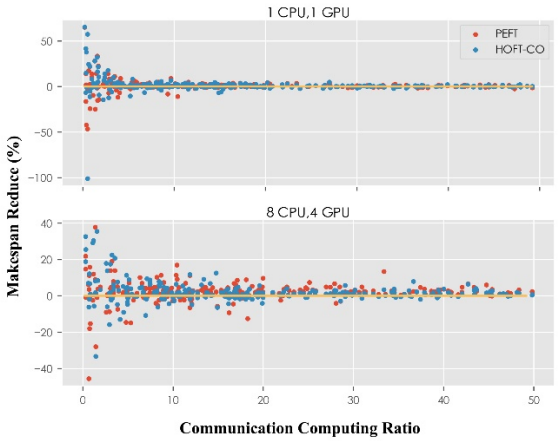

Figure 3. Comparison of makespan for Random DAGs.

Figure 4 shows all 90 randomly generated DAG sorted by CCR. The acceleration ratio on a single GPU platform is much smaller and has a narrower range than on multiple GPUs, such as Cholesky Dag. On multi-GPU platforms, the acceleration ratio is proportional to the number of GPUs. Figure 5 shows the acceleration ratio of workflow with 35~11480 tasks under different GPU number platforms. It can be seen that the acceleration ratio remains within a stable range on a single GPU platform. In the multi-GPU platform, the acceleration ratio gradually increases with the number of tasks increases, and finally stabilizes at a position proportional to the number of GPUs. After comparing the acceleration ratio of HEFT algorithm, it is found that the acceleration ratio of each stage has a good improvement. It also improves the application of scientific workflow with low CCR. To sum up, it can be seen that compared with HEFT, HOFT-CN has a better acceleration ratio.

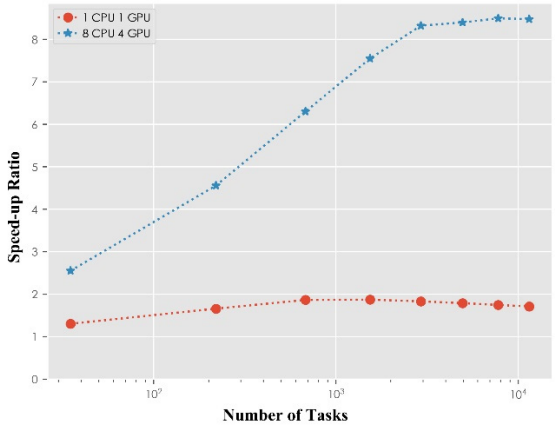

Figure 4. Comparison of speedup for Cholesky DAGs.

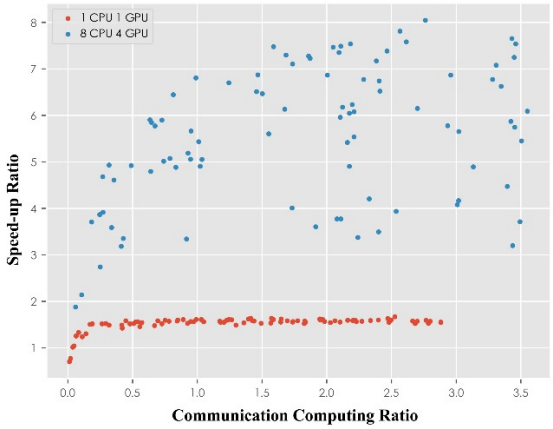

Figure 5. Comparison of speedup for Random DAGs.

Based on the energy model proposed in the (14), this article can calculate that reduce 5% than HEFT algorithm.

5. Conclusion

In CPU-GPU heterogeneous systems, traditional heuristic workflow scheduling algorithms cannot make good use of CPU-GPU resources. This article presents an improved workflow scheduling algorithm. Based on HEFT algorithm, the algorithm optimizes the task priority and processor selection stage, and introduces an optimistic completion schedule. Different from the traditional algorithm, this algorithm introduces the acceleration ratio in the setting of task priority, and avoids the problem of continuous occupation of GPU. Optimistic completion schedule is introduced in the processor selection stage and judged by the earliest completion time of the sub-task. Experimental results show that the algorithm is suitable for CPU-GPU heterogeneous environment. The task's makespan and energy consumption are reduced by 10% and 5% respectively. The acceleration ratio is increased 3% compared to the HEFT algorithm.

Acknowledgements

This work is supported by Beijing Natural Science Foundation (4192007), and supported by the National Natural Science Foundation of China (61202076), and this research was also sponsored by the International Research Cooperation Talent Introduction and Cultivation Project of Beijing University of Technology (No. 2021C02) along with other government sponsors. The authors would like to acknowledge the assistance given by Research IT, and the use of The HPC Pool funded by the Research Lifecycle Programme at The University of Manchester.

References

[1] Anakhi, Hazarika, Soumyajit, Poddar, Hafizur, & Rahaman. (2020). Survey on memory management techniques in heterogeneous computing systems. *IET Computers & Digital Techniques, 14*(2), 47-60.

[2] Panda, S. K. , & Jana, P. K. . (2014). An efficient task scheduling algorithm for heterogeneous multi-cloud environment. *International Conference on Advances in Computing* (pp.1204-1209). IEEE.

[3] Graham, R. L. , Lawler, E. L. , Lenstra, J. K. , & Kan, A. . (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics, 5*(1), 287-326.

[4] Pathan, R. M. , Voudouris, P. , & Stenstrom, P. . (2018). Scheduling parallel real-time recurrent tasks on multicore platforms. *IEEE Transactions on Parallel & Distributed Systems*, 1-1.

[5] Fechner, B. , Honig, U. , Keller, J. , & Schiffmann, W. . (2008). Fault-tolerant static scheduling

for grids. *IEEE International Symposium on Parallel & Distributed Processing*. IEEE.
[6] Agullo, E. , Beaumont, O. , Eyraud-Dubois, L. , & Kumar, S. . (2016). Are Static Schedules so Bad ? A Case Study on Cholesky Factorization. *IEEE International Parallel & Distributed Processing Symposium*. IEEE.
[7] Wu, J. , & Wu, J. X. . (2014). An srp-based energy- efficient scheduling algorithm for dependent real-time tasks. *International Journal of Embedded Systems, 6*(4), 335-350.
[8] Topcuoglu, H. , Hariri, S. , & Wu, M. Y. . (1999). Task scheduling algorithms for heterogeneous processors. *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*.
[9] Topcuoglu, H. , Hariri, S. , & Wu, M. Y. . (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*.
[10] Fechner, B. , Honig, U. , Keller, J. , & Schiffmann, W. . (2008). Fault-tolerant static scheduling for grids. *IEEE International Symposium on Parallel & Distributed Processing*. IEEE.
[11] Agullo, E. , Beaumont, O. , Eyraud-Dubois, L. , & Kumar, S. . (2016). Are Static Schedules so Bad ? A Case Study on Cholesky Factorization. *IEEE International Parallel & Distributed Processing Symposium*. IEEE.
[12] Zheng, W. , & Huang, S. . (2015). An adaptive deadline constrained energy-efficient scheduling heuristic for workflows in clouds. *Concurrency & Computation Practice & Experience, 27*(18), 5590-5605.
[13] Su, S. , Huang, Q. , Jian, L. , Xiang, C. , Peng, X. , & Kai, S. . (2015). Enhanced energy-efficient scheduling for parallel tasks using partial optimal slacking. *Computer Journal, 58*(2), 246-257.
[14] Lee, Y. C. , & Zomaya, A. Y. . (2011). Energy conscious scheduling for distributed computing systems under different operating conditions. *IEEE Transactions on Parallel & Distributed Systems, 22*(8), 1374-1381.
[15] Kumar, S. . (2017). Scheduling of Dense Linear Algebra Kernels on Heterogeneous Resources.
[16] Tobita, T. , & Kasahara, H. . (2010). A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling, 5*(5), -.
[17] Canon, L. C. , Marchal, L. , Simon, B. , & Vivien, F. . (2018). Online Scheduling of Task Graphs on Hybrid Platforms.