Heterogeneous Multiprocessor Matching Degree Scheduling Algorithm Based on OpenCL Framework

To cite this article: Juan Fang et al 2019 IOP Conf. Ser.: Mater. Sci. Eng. 490 042045

View the article online for updates and enhancements.
Heterogeneous Multiprocessor Matching Degree Scheduling Algorithm Based on OpenCL Framework

Juan Fang\textsuperscript{1,*}, Tingwen Yu\textsuperscript{1}, Zelin Wei\textsuperscript{1}

\textsuperscript{1}Faculty of Information Technology of Beijing University of Technology, Beijing 100124, China

*Corresponding author e-mail: fangjuan@bjut.edu.cn

Abstract. In a heterogeneous multi-core system, the merits of the task scheduling algorithm have a great impact on the efficiency of the system. This paper implements a matching degree scheduling algorithm based on heterogeneous multiprocessor. Before the actual scheduling, the static task feature value and the processor core configuration feature value are mapped into the same European space, and the "program-core-distance" matching value is calculated by using the WED. In the actual scheduling, the matching value and the size of the input byte are obtained, and the tasks are matched with the processor under the influence of the two to determine the execution order of the tasks on each processor. The scheduling method based on OpenCL framework is compared with the traditional scheduling algorithm. It shows that the scheduling scheme implemented in this paper has significantly improved efficiency when executing tasks.

1. Introduction

With the development of multi-core processors, computers begin to gradually implement parallel execution among processes. The research focus of heterogeneous multi-core architectures is how to maintain system load balancing, improve system operating speed, and reduce task run time\cite{1}. In order to study more efficient scheduling algorithms based on heterogeneous multi-core processors, researchers began to attempt to combine the idea of combinational optimization problems with task scheduling to find efficient task scheduling algorithms. Li D\cite{2} proposed a thermal-aware scheduling method by using hardware performance counters to detect system operating conditions. The algorithm implements the scheduling algorithm with DVFS (Dynamic Voltage and Frequency Scaling) under the condition of system full load, through register acquisition system condition plus temperature sensor to determine the process characteristics and reduce the priority of relevant process. Ghiasi S\cite{3} used a task-to-frequency scheduler to allocate tasks to heterogeneous multi-core processor systems. Each processor was running on fixed but different frequencies. Chen Q\cite{4} proposed an adaptive load-aware task scheduler. The scheduler includes a task scheduler based on historical task distributors and priority. The purpose is to balance parallel applications on asymmetric multi-core. The amount of load.

At present, the scheduling problem based on heterogeneous multi-core processor architecture is still in its infancy. There are many different processor cores in the heterogeneous multi-core processor. In order to give full play to the advantages of heterogeneous multi-core architecture, this paper implements a matching degree scheduling (MDS) algorithm based on heterogeneous multi-processor.
2. Static Task Division Method

2.1. Division Strategy
The design idea of the MDS algorithm implemented in this paper is to use the Weighted Euclidean Distance (WED) to match the abstract processor core configuration feature value with the static extracted task feature value in the European space, and obtain the “program-core-distance” matching value for actual scheduling use. Figure 1 shows a specific diagram.

Compile the task to be executed, and extract the required set of k feature values \{X_i|(i = 1,2,3,...k)\} during the compilation process. Each feature value is mapped into a set of feature vectors, that is, \(X_i = (x_{i1},x_{i2},...,x_{ik})\) is the feature vector describing the i-th \((1, 2,..., K)\) feature values. The mapping function corresponding to the task feature value is designed as \(\{R(X_1),R(X_2),...,R(X_k)\}\). Extract feature values for n processor resources, marking each processor as \(C_i(i = 1,2,...,n)\), the set of k feature values corresponding to each processor is \(C_i = (C_{i1},C_{i2},...,C_{ik}).\) Use the WED formula to derive the WED between each processor core and task, \(D_i(i = 1,2,...n)\) represents this WED, represents the weight coefficient of each dimension in the European space. The WED formula can be obtained as:

\[
D_i = \left( \sum_{j=1}^{k} w_i \times [R(X_j) - C_{ij}]^2 \right)^{\frac{1}{2}}
\]

2.2. Platform Foundation
This paper divides the task into the processor core with a high degree of processor matching, based on the results of the match between the hardware characteristics information of the CPU and the GPU and the characteristic information of the task. In order to give full play to the characteristics of CPU and GPU, this article uses the OpenCL framework for actual scheduling. During the scheduling process, the host program is responsible for dividing the task into devices for execution. The device is responsible for executing the corresponding kernel after receiving the instruction. In this article, the device is a multi-core GPU and CPU.

2.3. Platform Foundation
This paper uses Clang to compile the front end to compile. Clang is the compiler front end of LLVM (Low Level Virtual Machine). An important step in the static extraction of feature values is the overriding of the HandleTranslationUnit function. The feature values extracted in this paper are as follows:

- Instruction dependent distance[5]. The distance between two instructions that access the same register.
- Branch conversion rate[6]. The rate at which the program executes when it encounters a branch.
- Stack distance[7] distribution. The data cache size required for data localization.

The abstraction of the processor core configuration feature values corresponding to these three static feature values is as follows:

- Pipeline width. The number of instructions that can be executed simultaneously in a parallel task.
• Branch predictor size. Branch predictors are used in multicore processors to predict the direction of branches in advance.
• L1 data cache size. A cache is a type of cache that is integrated inside the processor and is used to store data temporarily while the processor is processing data.

By using the mapping function, the static task eigenvalues are calculated to match the values of the processor core configuration eigenvalues required in each dimension of the Euclidean space. The design idea of these mapping functions is to divide the instructions into n groups according to the eigenvalue requirements. \( X_i (i = 1, 2, \ldots, n) \) represents the number of tasks falling in each group. \( W_i (i = 1, 2, \ldots, n) \) is the mapping coordinate weight of each dimension in the European space. The weighted average weight obtained is the matching value, and the formula is:

\[
R = \frac{\sum_{i=1}^{n} W_i \times X_i}{\sum_{i=1}^{n} X_i}
\]

3. Task Scheduling Method

3.1. Scheduling Method Introduction

In the actual scheduling process, not only the matching relationship between the OpenCL kernel program and the processor is taken into consideration, but also the size of the bytes input by the host into the device will also affect the efficiency of task execution in the scheduling process, as shown in Figure 2.

![Figure 2 OpenCL tasks](image)

MDS comprehensively considers these two factors. Before scheduling, the two will first sort all the tasks to be executed into a queue through a certain matching function. Both ends of the queue are strongly related to CPU processor resources and GPU processor resources. At the time of scheduling, both ends of the queue are simultaneously dequeued to the processor cores that are strongly associated with each other. When the tasks in the middle of the queue are dequeued, that is, the degree of correlation between the tasks and the two processor cores is similar, the Task scheduling is performed on the processor core that is idle first. As shown in Figure 3.

![Figure 3 Real-time scheduling diagram](image)
3.2. Matching Function

Compared with the CPU, the floating-point computing capability of the GPU is much larger than that of the CPU, and it is more suitable for computing large amounts of data. The CPU is more suitable for more logical operations or more interactive operations.

For CPU devices, the smaller the "distance", the smaller the number of input bytes, the higher the priority of the task on the CPU device. Assuming that the predictor predicts that there are a total of n tasks to be executed on the CPU device, setting a priority index \( R_i (i = 1, 2, \ldots, n) \), the formula for calculating the index can be obtained:

\[
R_i = \frac{d_i}{d_{\text{min}}} + \frac{p_i}{p_{\text{min}}}
\]  

(3)

Where \( d_i (i = 1, 2, \ldots, n) \) represents the "distance" obtained after each task passes through the predictor, \( d_{\text{min}} \) represents the minimum distance, \( p_i (i = 1, 2, \ldots, n) \) represents the input data size of each task, \( p_{\text{min}} \) represents the smallest input data. The smaller the index, the higher the priority of the task on the CPU device. For GPU devices, unlike a CPU devices, the smaller the "distance", the greater the number of input bytes, the higher the priority of the task on the GPU device. The formula is:

\[
R_i = \frac{d_i}{d_{\text{min}}} + \frac{p_i}{p_{\text{min}}}
\]  

(4)

Sort the "distance" of all tasks in ascending order. \( s_i (i = 1, 2, \ldots, n) \) represents the ranking after reordering, \( d_{(s_i)} \) is the "distance" corresponding to the reordering. The larger the index, the higher the priority of the task on the GPU device.

4. Experiment and Result Analysis

4.1. The Experiment Environment

The scheduling scheme implemented in this paper is implemented on the W580-G20 GPU server. The server's heterogeneous multi-core processor OpenCL platform is the NVIDIA GPU platform and the Experimental OpenCL2.1 CPU only Platform platform. The NVIDIA GPU platform includes two OpenCL GPU devices, all of which are Tesla p100. The Experimental OpenCL2.1 CPU only Platform platform contains two OpenCL CPU devices, all of which are Intel(R) Xeon(R) CPU E5-2620 v3. Table 1 shows the detailed parameters of the experimental platform.

| Parameter information | CPU | GPU |
|-----------------------|-----|-----|
| Model                 | Intel(R) Xeon(R) CPU E5-2620 v3 | Nvidia Tesla P100-PCIE |
| The number of core    | 8   | 3840 |
| OpenCL version        | OpenCL 2.1 LINUX                   | OpenCL 1.2 CUDA 9.1.84 |
| OpenCL platform       | Experimental OpenCL 2.1            | NVIDIA CUDA |
| L1 cache size         | 32 k | 256 k |
| Branch predictor size | 8K GShare | 1K GShare |
| Memory size           | 16 G |       |
| Operating system      | Ubuntu 16.04.3 LTS                 |       |
| Translater            | GCC, Clang                          |       |

Table 1 Experimental Platform Information

Feature information is abstracted to extract hardware processor core configuration feature values, and the number of processor cores is treated as pipeline width. This is because the core data represents the pipeline width without regard to the dependency of the task instructions. The abstracted feature values are Pipeline width, Branch predictor size and L1 cache size.
4.2. The Experiment Result and Analysis

The code value of the six benchmarks extracted by Clang and the "program-core-distance" match value are shown in Table 2.

| Static code feature value | Average instruction dependent distance | Average branch conversion rate | Average Stack distance distribution | Core | Distance |
|---------------------------|----------------------------------------|-------------------------------|------------------------------------|------|-----------|
| backprop                  | 2.94                                   | 0.44                          | 2.69                               | CPU  | 26737.7   |
| cfd                       | 9.62                                   | 0.38                          | 6.58                               | GPU  | 31949.6   |
| gaussian                  | 5.32                                   | 0.86                          | 844                                | GPU  | 16271.6   |
| hotspot                   | 1.87                                   | 0.56                          | 6.19                               | CPU  | 44256.7   |
| kmeans                    | 2.39                                   | 0.54                          | 3.66                               | CPU  | 40913.6   |
| streamclus                | 8.13                                   | 0.27                          | 5.51                               | GPU  | 41754.0   |

In order to verify that the MDS implemented in this paper can effectively improve the efficiency of task execution, this paper selects the experimental results of several traditional scheduling schemes as comparisons. The following is a description of the comparative experiment:

- FCFS: Tasks are allocated to any idle device during the scheduling process.
- ALL_ON_CPU: All tasks are dispatched to the multi-core CPU during the scheduling process.
- ALL_ON_GPU: All tasks are dispatched to the multi-core GPU during the scheduling process.
- "Program-core": Only based on the "program-core" matching group and the input data size.
- MDS: This paper implements the scheme. The tasks are scheduled and allocated under the combined effect of the "program-core-distance" matching group and the input data size.

Figure 4 shows the execution time spent by different scheduling schemes when performing the same task. As can be seen from Figure 4, the FCFS execution time is 388.705 ms, the ALL_ON_CPU execution time is 672.0665 ms, the ALL_ON_GPU execution time is 363.4491 ms, the "program-core" execution time is 305.7885 ms, and the MDS scheduling scheme execution time is 276.4231 ms. The MDS has the shortest execution time.

![Figure 4 Execution time of task scheduling](image)

Figure 5 shows the FCFS scheduling scheme as the 1.00 baseline and the acceleration ratio of the remaining scheduling schemes. The ALL_ON_CPU acceleration ratio is 0.578 relative to the reference
line. The ALL_ON_GPU acceleration ratio is 1.069. It can be seen from the figure that the "program-core" has an acceleration ratio of 1.271 relative to the baseline, which is an effective solution. The MDS proposed in this paper, compared with the baseline, the acceleration ratio is 1.406, which is obviously better than the three schemes. Compared to the "program-core", the MDS has improved efficiency by 10%.

![Figure 5 Acceleration ratio of task scheduling](image)

5. Conclusion
This paper implements a MDS algorithm based on heterogeneous multiprocessor based on OpenCL framework on W580-G20 GPU server. Compared with the traditional scheduling scheme FCFS, the algorithm improves the efficiency by 40.6% when performing tasks. Compared with the "program-core" matching algorithm, the algorithm improves the efficiency by 10% when performing tasks. The other two algorithms, ALL_ON_CPU and ALL_ON_GPU, have no significant improvement in task execution efficiency or even decline compared to FCFS. This is because when using these two methods for scheduling, tasks are allocated to the CPU or GPU. Some tasks that are more suitable for execution on other devices do not get the best way to perform, which affects efficiency. It is proved that the scheduling scheme implemented in this paper speeds up the task execution efficiency while improving the utilization of processor resources.

Acknowledgments
This work is supported by the National Natural Science Foundation of China (Grant No. 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.

References
[1] Roy A, Aydin H, Zhu D. Energy-efficient primary/backup scheduling techniques for heterogeneous multicore systems[C]// Green and Sustainable Computing Conference. IEEE, 2018:1-8.
[2] Li D, Chang H C, Pyla H K, et al. System-level, thermal-aware, fully-loaded process scheduling[C]// IEEE International Symposium on Parallel and Distributed Processing. IEEE, 2008:1-7.
[3] Ghiasi S, Keller T, Rawson F. Scheduling for heterogeneous processors in server systems[C]// Conference on Computing Frontiers. ACM, 2005:199-210.
[4] Chen Q, Guo M. Adaptive workload-aware task scheduling for single-ISA asymmetric multicore architectures[J]. Acm Transactions on Architecture & Code Optimization, 2014, 11(1):1-25.
[5] Dongdong L U, Jun H E, Yang J, et al. Memory dependence prediction method based on instruction distance[J]. Journal of Computer Applications, 2013, 33(7):1903-1907.

[6] Quinones E, Parcerisa J M, Gonzales A. Improving Branch Prediction and Predicated Execution in Out-of-Order Processors[C]// IEEE, International Symposium on High PERFORMANCE Computer Architecture. IEEE, 2016:75-84.

[7] Bilardi G, Ekanadham K, Pattnaik P. Efficient Stack Distance Computation for a Class of Priority Replacement Policies[J]. International Journal of Parallel Programming, 2013, 41(3):430-468.