Jestr

Journal of Engineering Science and Technology Review 9 (6) (2016) 87- 94

Research Article

JOURNAL OF
Engineering Science and Technology Review

www.jestr.org

Adaptive Scheduling Framework for Multi-Core Systems Based on the Task-Parallel Programming Model

H. M. LU[1], Y. J. CAO[2,*], J. J. SONG[1], T. Y. DI[1], H. Y. SUN[3] and X. M. HAN[1]

[1] *School of Computer Science and Engineering, Changchun University of Technology, Changchun 130012, China*
[2] *School of Software, Zhengzhou University, Zhengzhou 450000, China*
[3] *School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore*

Received 19 May 2016; Accepted 30 November 2016

Abstract

With the rapid development of multi-core processor systems, software parallelization has become the main approach in improving the efficiency of multi-core processors. However, the most updated multi-core parallel programming models have defects, such as poor scalability and intensive competition in processor core resources. To prevent congestion of system processor core resources and to improve equal distribution of processing resource and service efficiency, the adaptive co-scheduling problem of multi-core runtime systems was studied in this paper. First, on the basis of the online competition analysis method, a quantitative analysis of task schedulability was conducted. Second, a random work stealing strategy was combined with work stealing frequency to dynamically redistribute multi-core resources. Third, on the basis of closed-loop feedback control theory, an adaptive co-scheduling method that could obtain a dynamic perception of the degree of task parallelism was proposed, and multi-core adaptive co-scheduling system A-SYS (Adaptive SYStem) based on fine-grained task programming model was designed and implemented. Finally, the proposed framework was used to conduct performance analysis of multiple parallel tasks, and the performances of different algorithms were compared through a prototype system experiment. Experimental results indicated that the proposed adaptive scheduling method and a dynamic perception of core resources could effectively improve mutual competition between inter-core tasks and shared resources. Lower damage cost during task scheduling process, and significantly elevated the service efficiency of multi-core processors and equal distribution in resource allocation. Compared with traditional scheduling algorithm EQUI (EQUI-partitioning), A-SYS shortened the running time of application programs by nearly 50%, and as the number of application programs increased, the effect of A-SYS became more prominent. This finding is of significant reference value to performance problems caused by a continuous increase in the inner core scale of multi-core processors in the future.

Keywords: Multi-core processor system, Fine-grained task programming model, Runtime system, Adaptive scheduling

1. Introduction

In recent years, with the elevated circuit integration and dominant frequency of single-core processor chips, processor technology has encountered problems, such as manufacturing cost, power consumption, and heat dissipation. As a result, multi-core and multi-thread technology is steered toward a new direction in the development of processor systems. A multi-core processor, which is also called chip multiprocessor (CMP), integrates several processor cores with independent functions on the physical chip of one processor and takes the entire processor chip as a uniform structure to provide outward computing service. Unlike the traditional single-core processor, a multi-core processor increases the number of physical threads or tasks simultaneously executed by the entire processor multiple times by integrating several single-thread cores or multi-thread processing cores, thereby greatly improving the parallel processing capability of the processor system. At present, generalized processor products that integrate dozens of cores are available. With the progress of integrated circuit manufacturing process and increased computing demand, future processors will integrate hundreds and even thousands of cores. A continuous increase in the kernel scale of multi-core processor systems guarantees its sustainably enhanced abilities in computing and data processing; however, without parallel programming languages or support from the system software level, the improvement of this hardware capability cannot enhance application program performance, and this issue is one of the most severe challenges in the multi-core era [1].

The concepts of parallel programming and computation emerged early. However, even after several years, parallel computing has not become the mainstream of pervasive computing. The emergence and the rapid development of multi-core processor systems have made people realize that such systems are suitable only for designing and constructing parallel hardware, and the difficulty and challenges faced by application programs lie in the design of a parallelization method with high yield and a highly efficient parallel execution model. To improve the parallel programming ability of multi-core processor systems and ensure service efficiency of processor core resources and transportability of application programs in different

* E-mail address: caoyj@zzu.edu.cn

platforms, most multi-core programming models adopt the parallel model based on fine-grained tasks [2]. The term "fine-grained task" refers to a special object that includes several program instructions and can be independently executed. It is a smaller parallel granularity than the thread of an operating system. Compared with a system thread of an operating system, a fine-grained task has the following features: (1) establishment and termination of a task are more efficient and flexible than those n a system thread; (2) a task usually has a shorter executable code than a system thread does and requires a smaller expenditure on management and scheduling, and it can easily realize system load balancing; (3) a task is usually managed and scheduled by the runtime system and is relatively independent from the operating system, thereby greatly improving the transportability of the application program; and (4) task-based parallel programming can support irregular application better with a broader application range. However, traditional multi-core runtime systems still have numerous problems and deficiencies in supporting fine-grained tasks, thereby resulting in a low resource utilization rate of application programs and poor system scalability. Consequently, a single application program cannot take full advantage of processor core resources, and the execution performance of the application program cannot increase correspondingly as usable core resources increase.

To address the defects of current multi-core runtime systems, which easily cause intense competition among processor core resources and poor system scalability, an adaptive co-scheduling problem of multi-core runtime systems was studied in this paper. Resource distribution, runtime control, and task execution were considered an organic whole, and then an adaptive co-scheduling framework was proposed based on the concept of dynamic feedback control. A quantitative analysis method of scheduling based on work stealing strategy and work stealing frequency was studied from the aspect of task schedulability. Through online competition analysis, the performance analysis of the time complexity of the adaptive scheduling algorithm was designed and implemented.

2. State of the art

Parallel programming model, as an intermediate bridge that connects the application program developer and the hardware architecture, is crucial to pervasive parallel programming of multi-core processors. Good parallel programming model not only simplifies the programming process of parallel programs and lowers the design difficulty of application programs but also provides an application program with strong parallel execution capacity, thereby achieving a good balance between two mutually contradictory goals, namely, and software productivity and execution efficiency.

In a broad sense, parallel programming models that implement parallelization of application programs can be divided into data parallel programming model and task parallel programming model [3]. The data parallel programming model usually focuses on data, and it executes data parallel processing through reasonable data partitioning and balanced data distribution onto parallel computing nodes. In traditional high-performance computing, the data parallel programming model is applied extensively, and typical representatives of this model are MPI parallel programming model [4], which is based on message passing and the MapReduce parallel programming model proposed by Google [5]. The data parallel programming model is designed mainly for large-scale data processing. During this process, the program developer is responsible for data parallel granularity partitioning, data synchronization, and even system load balancing. Thus, the application range of this model is affected by various limitations. Compared with the traditional data parallel programming model, parallel programming model based on fine-grained tasks conducts parallelism expression from the computing task angle. In the parallel model based on fine-grained tasks, the programmers' emphasis is the identification of tasks or decomposition into computer subtasks, while the compiler and the runtime system are responsible for dynamic generation, task scheduling, and load balancing of tasks. The implementation method, which decomposes and separates task scheduling, provides strong productivity and execution capability [6], [7]. Research and practices in recent years indicate that the parallel model based on fine-grained tasks has become a powerful tool in implementing pervasive parallel programming. For example, a high-productivity computing systems project funded by the U.S. Defense Advanced Research Projects Agency used Chapel from Gray Corporation [8], Fortress of Sun Corporation (now Oracle Company) [9], and X10 from IBM [10], which are three programming languages developed on the basis of the parallel programming model based on fine-grained tasks. In the past several years, parallel programming models based on fine-grained tasks have rapidly become popular and have undergone development, including the concurrency library ForkJoin framework of Java 5[11], [12], the thread building blocks (TBB) of Intel [13], the task parallel library of the Microsoft .NET framework [14], and Cilk/Cilk++[15], [16]. OpenMP API specification mainly supported data parallel model before version 2.5, and the concept of task parallelization was imported into OpenMP 3.0 API specification [17], [18] in 2008.

In sum, various types of runtime systems that support multi-core programming model currently exist. Various runtime systems are mutually independent, but they cannot mutually exchange information. When multiple application programs operate concurrently, a lack of a unified resource distribution coordinating mechanism will cause malicious competition among core resources, thereby reducing the overall handling capacity of the system. To address the problems and deficiencies faced by multi-core runtime systems, this paper studied an adaptive co-scheduling method that is applicable to multi-core processor systems. Analysis and design were conducted from various aspects, such as optimized resource allocation, runtime control, and highly efficient task scheduling to enhance the easy usability, adaptability and cooperation of the runtime programming model, and to optimize and improve the overall efficiency and expandability of multi-core processor systems.

The remainder of this paper is organized as follows: Section 3 describes the adaptive scheduling framework based on fine-grained tasks and its basic principle, and it presents a state transition method based on the scheduling strategy of task sharing and worker threads of the work stealing strategy. Section 4 discusses the application of the model for performance analysis of multiple parallel task loading, and the performances of different algorithms are compared through a prototype system experiment. Section 5 presents relevant conclusions.

3. Methodology

3.1 Logical structure and functions of adaptive scheduling framework based on fine-grained tasks

Traditional multi-core programming model runtime systems have problems such as lack of flexibility of core resource allocation, lack of a unified coordinating mechanism, and low resource utilization rate of application program. The popular programming model OpenMP and Cilk runtime system are taken as references, and an adaptive co-scheduling prototype system Adaptive SYStem (A-SYS), which is applicable to multi-core processors, was built; its logical structure is shown in Figure 1. A-SYS divides the task loads of the system into two major types, namely, controlled tasks and uncontrolled tasks. Controlled tasks are the main control objects of A-SYS, where parallel application programs supported by runtime systems of multi-core programming models such as Cilk, OpenMP, and TBB are positioned. Uncontrolled tasks refer to traditional serial programs and parallel application programs that cannot be controlled by multi-core runtime systems, such as applications developed on the basis of PThread and MPI. A-SYS adopted relatively simple control strategies for uncontrolled tasks. For example, at the initial level of execution of application programs, corresponding processor core resources are distributed according to system load state. Dynamic control of this type of tasks will not be implemented during the operating process. After the task operation is completed, resource recycling is performed. Unified management of uncontrolled tasks conducted by A-SYS reduces bad competition between controlled tasks and uncontrolled tasks for processor core resources and could effectively improve and optimize the overall efficiency of multi-core processor systems.

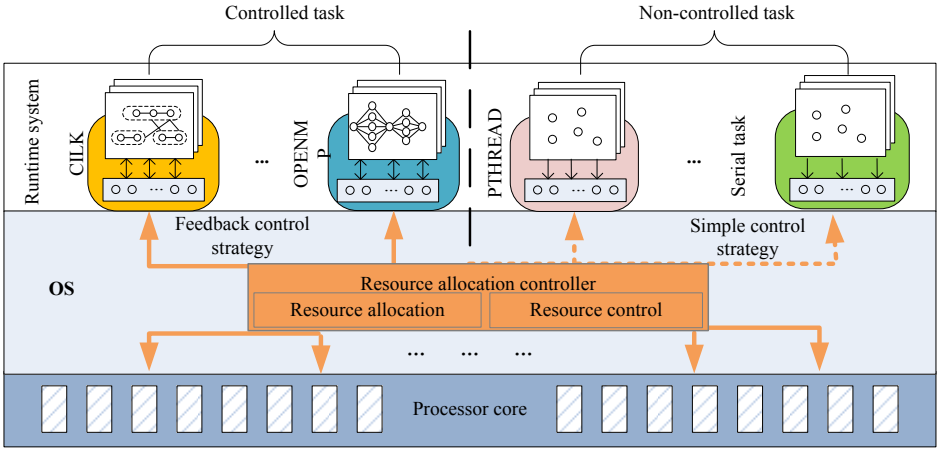


Fig. 1. Logic structure of adaptive co-scheduling system

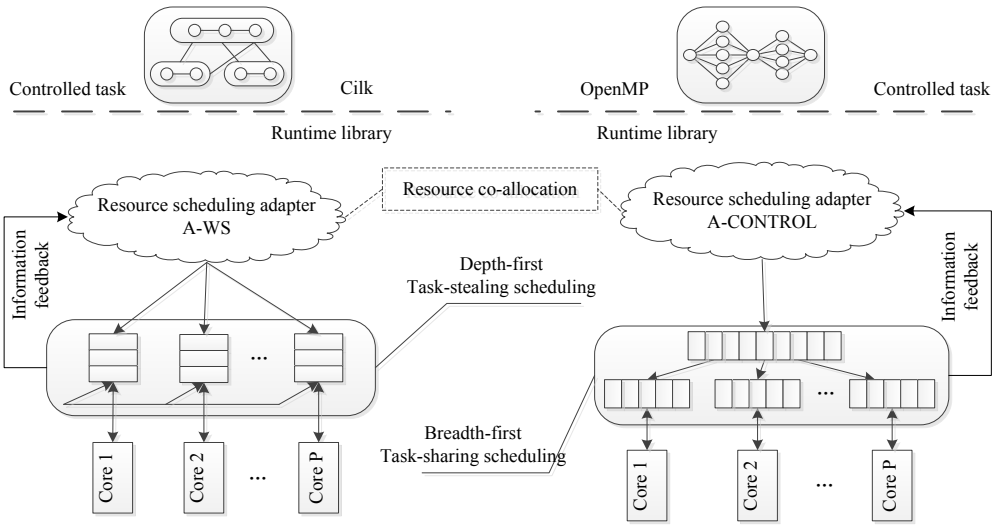


Fig. 2. Logic structure of adaptive co-scheduling of controlled tasks

Dynamic management and scheduling of controlled tasks are core parts implemented by A-SYS. As shown in Figure 1, A-SYS implemented resource coordinated distribution and dynamic scheduling of controlled tasks mainly through resource distribution control, runtime support library, and feedback control strategy between the two. In the A-SYS system, resource distribution controller was further subdivided into resource dynamic partitioning subsystem and resource dynamic control subsystem. The main functions of resource dynamic partitioning subsystem are to realize dynamic partitioning and grouping management of multi-core processor core resources according to feedback information and system load information of runtime and to reduce bad competition between different runtime systems for multi-core processor resources to improve and optimize overall system efficiency. The main function of the resource dynamic control subsystem is to achieve unified and coordinated distribution of processor resources through feedback control strategies to enhance the utilization rate of multi-core processor core resources.

In the aspect of runtime support library, A-SYS mainly made the following improvements to address the problems

in the traditional programming model runtime system: first, it removed the disadvantages of the manual static distribution processor core resources of the traditional runtime system and provided application programs with a runtime adaptive regulation ability; second, it imported a runtime dynamic feedback control mechanism and coordinated fair resource distribution between different runtime systems; and finally, it imported highly efficient acquisition mechanism runtime dynamic behavior characteristics of application programs to guide and optimize distribution and task scheduling of multi-processor system resources. Figure 2 shows a logical structure of adaptive co-scheduling of controlled tasks, and the A-SYS runtime library adopted two scheduling strategies with extensive applications, such as work sharing scheduling strategy based on breadth-first principle represented by OpenMP and work stealing strategy based on depth-first principle represented by Cilk. Between different runtime systems, A-SYS realized coordinated distribution of multi-core processor resources through provided resource distribution controller, as shown in modules in the virtual frame in Figure 2. Core resource distribution controller operates on an operating system, and it is the kernel module of dynamic distribution and management of multi-core processor core resources. In concrete implementation, resource distribution controller adopted the background Daemon method of Linux to realize dynamic distribution and control of processor core resources, and realized communication and data transmission with runtime system process through the methods of shared memory, semaphore, and the like.

3.2 Adaptive control of A-SYS runtime resources

Highly efficient realization of dynamic distribution and timely adjustment of processor core resources on the condition that normal execution of application programs is not affected is a complicated and difficult problem because of the following reasons: first, the resource regulation process must ensure normal operation with no interruption of application programs; second, ensuring accurate program execution results is necessary, that is, disorder between different program data and erroneous operating results need to be avoided; finally, additional expenditure due to resource regulation when the program is operating should be minimal to avoid significantly affecting program execution performance. In view of the above reasons, A-SYS adopted an indirect method to realize dynamic control and adaptive regulation of processor core resources, that is, A-SYS realized dynamic control of processor core resources through dynamic control of the working state of worker threads in the runtime system and dynamic mapping relation between the worker thread and the processor core.

To realize dynamic management of processor core resources, A-SYS established a one-to-one mapping relation between the system worker thread and the processor core, and realized runtime's indirect control of core resources by assigning different working states to each worker thread. In a runtime system (like OpenMP) that supports the task-sharing scheduling strategy based on the breadth-first principle, A-SYS assigned three correspondingly different states to each worker thread, namely, working, mugging, and sleeping. A state transition during the execution process is shown in Figure 3. The different working states of the worker thread are described as follows: (1) working represented that the worker thread was under a working state, that is, the local working queue and the stack were under non-null state; (2) sleeping indicated that the worker thread was under the sleeping state; (3) mugging, which is a special sleeping state that is also called transfer state, referred to a situation in which a worker was under the sleeping state, and its local queue and stack were under the non-null state, that is, this worker had uncompleted tasks, and other local queues were tabbed as "mugged."

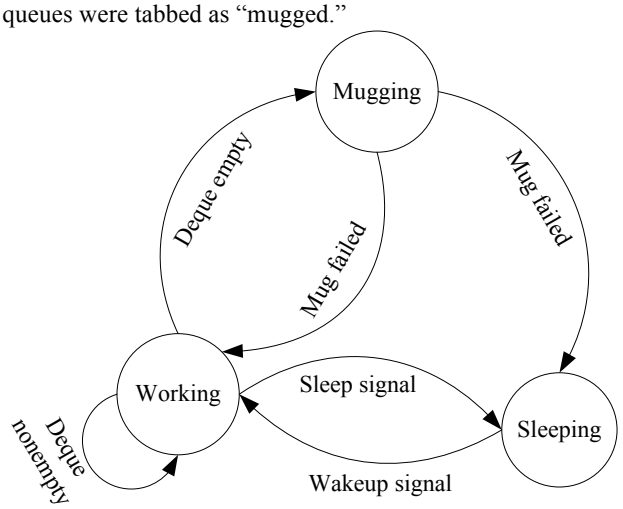


Fig. 3. State transition diagram of worker thread based on task-sharing scheduling strategy

In the task-sharing scheduling strategy based on the breadth-first principle, the basic working principle of an A-SYS runtime system that realizes adaptive regulation of core resources is that at system initialization, A-SYS establishes a corresponding system thread, namely, worker, for each processor core. Under initialization execution of the application program, A-SYS set one worker under the working state and scheduled the main thread of the application program to this worker, while other workers were under the sleeping state. During the execution process of the application program, according to an increase or a decrease in the distribution quantity of the processor core resource, A-SYS dynamically adjusted the working state of the corresponding worker thread to make it consistent with the actual distribution quantity of core resources.

In the runtime system (like Cilk) of work stealing strategy based on depth-first principle, each worker thread corresponded to four different working states, namely, working, mugging, sleeping, and stealing, and state transition during its execution process, as shown in Figure 4. The definitions of working, mugging, and sleeping states are consistent with those under the task-sharing scheduling strategy based on the breadth-first principle. The stealing state represented that the worker thread was under the stealing state, that is, the local working queue and stack were empty, and the working loads of other worker threads were stolen according to a set stealing protocol.

In the work stealing strategy based on the depth-first principle, the basic working principle of an A-SYS runtime system that realizes adaptive regulation of core resources is that at system initialization, A-SYS establishes a corresponding worker thread for each processor core. When the application program started to operate, A-SYS set one worker under the working state and other workers under the sleeping state; the main thread of the application program was scheduled to this worker for execution. During the execution process of the application program, A-SYS dynamically adjusted the working state of the corresponding worker thread according to an increase or a decrease in the distribution quantity of the processor core resource to make

it consistent with the actual distribution quantity of core resources.

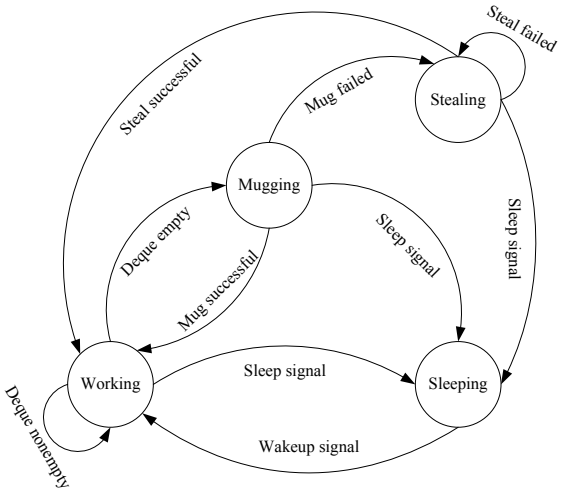


Fig. 4. State transition diagram of worker thread based on work-stealing strategy

4 Result analysis and discussion

4.1 Performance analysis of typical programming model runtime system

A detailed experimental verification of two typical programming models, namely, MIT Cilk and OpenMP 3.0, based on fine-grained tasks was conducted. The experimental platform was Sun Fire X4600 M2 32-core server with a 256 GB memory and Linux operating system kernel version 2.6.28. Test loads selected the standard test set officially released by Cilk [15] and the OpenMP standard test set Barcelona OpenMP Task Suite (BOTS) [19]. Detailed descriptions of the test sets are shown in Table 1 and Table 2.

Table 1. Cilk test set list and descriptions

No.	Load name	Scale	Load descriptions
1	CK	Searching depths of two parties are 10 and 13	Checkers
2	FIB	Series size is 45	Fibonacci series
3	FFT	Sequence scale is 228	Fast Fourier transform
4	Heat	Granularity is 10, number of columns is 8,192 and number of rows is 8,192	Heat dissipation problem based on finite difference method
5	LU	8192×8192 matrix	LU matrix decomposition
6	Strassen	4096×4096 matrix	Strassen algorithm is used to realize matrix multiplication

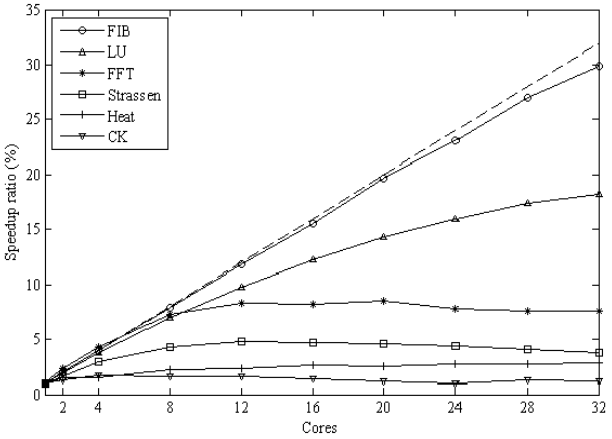
Table 2. BOTS standard test set list and descriptions

No.	Load name	Scale	Load descriptions
1	Alignment	100 sequence pairs	Protein molecular sequence alignment
2	Health	4 levels, each level has 36 cities	Columbia healthcare system simulation
3	Sort	128M integers	Use one group of mixed sorting algorithm to sort arrays
4	SparseLU	Main matrix size is 100 and submatrix scale is 50	Calculate LU decomposition of sparse matrixes
5	Strassen	2048×2048 maxtrix	Strassen algorithm is used to realize matrix multiplication

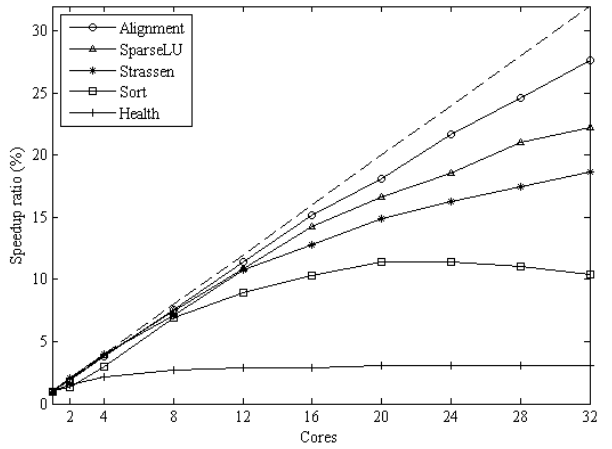
(1) Performance speed-up ratio experiment of single application program

Test results are shown in Figure 5. The experimental results indicate that as the number of processor cores used by application programs increased, most application programs could obtain performance speed-up ratios of different degrees. When a few processor cores were present in the system, for example, when the number of processor cores in Figure 5(a) did not exceed 8, the speed-up ratios of most Cilk application programs presented approximately a linear growth as the number of processor cores increased. However, as the number of processor cores increased further, the increasing amplitudes of performance speed-up ratios of both Cilk and OpenMP application programs and even those of some applications became relatively slow; the programs presented a descending tendency. The above experimental results indicated that present programming model runtime systems were effective when the number of processor cores was small, but as the number of system processor cores increased to more than 8 cores, their performance became poor, thus resulting in poor performance expandability of application programs. In sum, by only relying on a scheduling strategy provided by present programming model runtime systems, single parallel application programs could not sufficiently and effectively use all processor core resources. Moreover, blind distribution of excess processor

core resources could not ensure corresponding improvement of performance of application programs; instead, such performance would probably deteriorate.



(a) Clik

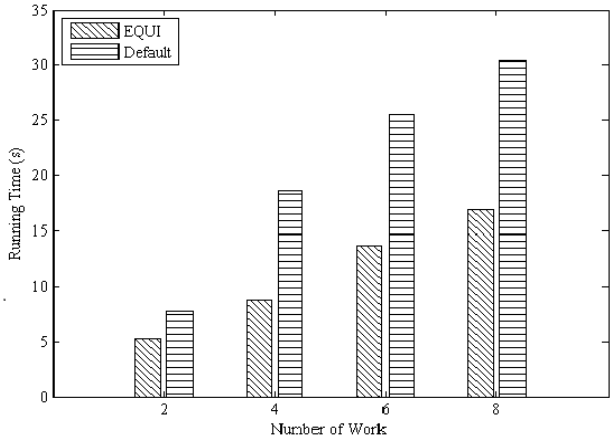


(b) OpenMP

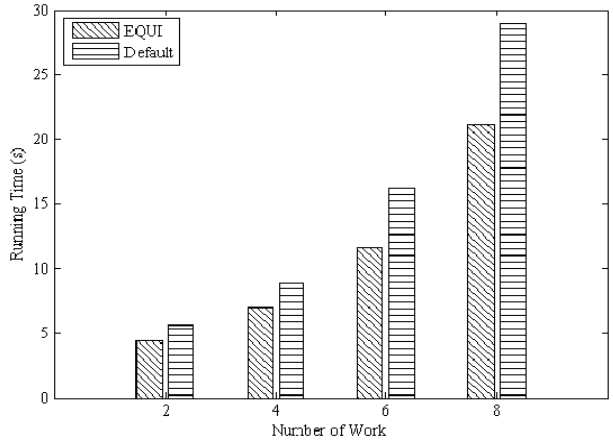
Fig. 5. Comparison of performance speed-up ratios of application loads

(2) Performance analysis under concurrent execution of multiple application programs

Previous analysis indicated that to achieve highly efficient usage of processor core resources, multi-core programming runtime systems usually included load balancing strategy of their internal threads. For example, OpenMP supported static, dynamic, and runtime scheduling strategies [20], and Cilk supported the work stealing scheduling strategy [21], [22]. However, these scheduling strategies were limited only to load balancing of the internal worker thread of a single application program without considering the optimized scheduling of resources of the entire processor system. In addition, traditional operating systems usually adopted a scheduling mechanism based on time slice and thread with a lack of a co-scheduling mechanism for multi-core runtime systems. The same experimental environment was used to conduct a comparative verification of performance under concurrent operation of multiple Cilk and OpenMP application programs. By submitting multiple different application programs and taking the minimum completion time of all application programs as the evaluation index, this experimental process conducted a comparative test between two different scheduling strategies, and experimental results are shown in Figure 6. The default represented the scheduling algorithm with default support from Cilk and OpenMP runtime systems, and each application program was statically distributed with 16 cores by the runtime system under its initialization; Equi-partitioning (EQUI) represented classical balanced distribution algorithms, that is, processor core resources were averagely distributed according to the number of application programs in the system.



(a) Clik



(b) OpenMP

Fig. 6. Comparison of concurrent operation time among multiple application programs

As shown in Figure 6, the minimum completion time of tasks taken as the evaluation index resulted in a significant difference between the scheduling algorithm with default support from Cilk and OpenMP runtime systems and the EQUI algorithm in program execution time. Consequently, as the number of concurrently operating application programs in the system increased, the difference between the two different algorithms in terms of the operating time of application programs became obvious. For example, when the number of concurrently operating programs in the system reached 8, the difference between the two scheduling strategies was doubled or more. The difference in the operating times of the application programs under two different scheduling strategies indicated that many problems still exist in present programming model runtime models in practical utilization. When multiple application programs simultaneously operated, resource competition would be easily generated. Consequently, the utilization rate of processor core resources lowered, and the runtime performance of application programs reduced.

4.2 Comparative analysis of A-SYS performance

During the experimental process, three different types of test loads were selected. The first type was the OpenMP standard

test set BOTS provided by Barcelona Supercomputer Center [19], and its detailed information is shown in Table 2. The second type was the Cilk standard test set [15], and its detailed information is shown in Table 1. The third type was serial program FIBs calculating Fibonacci number and realized on the basis of C language. Through a comparison with the traditional runtime system scheduling strategy, the actual performance of the A-SYS system was verified.

The experimental process is as follows: six Cilk test programs in Table 1, five OpenMP test programs, and one serial program FIBs in Table 2 generated 6×3+5×3+1×3 for a total of 36 test cases. Test loads in different quantities were randomly selected and submitted to the system in batches. The traditional scheduling strategy with default support from OpenMP and Cilk 5.4.6 runtime systems was taken as reference, and the completion time of the task set was taken as the evaluation index. A comparative verification of actual performance of A-SYS was conducted. During the experimental process, each OpenMP application load and Cilk application load were distributed with 16 default processor cores, and serial program FIBs was distributed with one default processor core. The experimental results are shown in Figure 7.

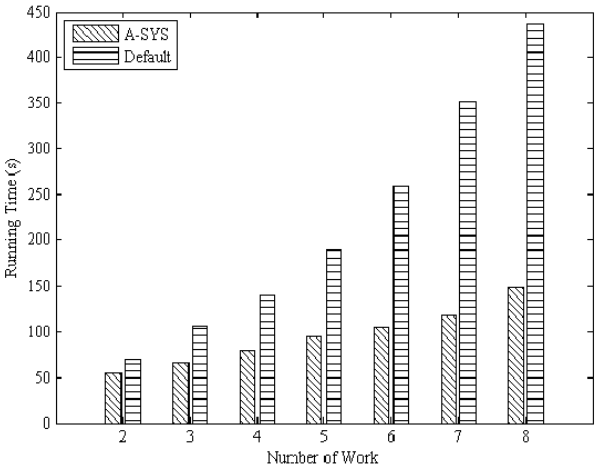


Fig. 7. Comparisons between the performances of A-SYS and the traditional scheduling algorithm

Experimental results indicated that compared with the scheduling strategy with default support from traditional OpenMP and Cilk runtime systems, A-SYS significantly improved the execution efficiency of application loads, and as the number of concurrent application loads in the system increased, its advantages became more obvious. For example, when the number of concurrent application loads in the system reached 8, compared with the traditional scheduling strategy, A-SYS reduced the execution time of application loads by nearly 60%. The above experimental results are due to two main reasons. First, A-SYS adopted resource adaptive scheduling through runtime, thus eliminating the defects of static distribution of processor core resources of the traditional programming model runtime system and effectively improving the service efficiency of processor core resources and the parallel execution ability of application loads. Second, according to the operating characteristics of application loads, A-SYS could realize coordinated distribution and dynamic partitioning of processor core resources to reduce bad competition between processor resources at different application loads and improve the overall efficiency of multi-core processor systems.

5. Conclusions

To solve the performance optimization problem faced by multi-core processor systems with an increase in the number of cores, an adaptive co-scheduling method applicable to multi-core parallel processor systems was proposed. Starting from a quantitative analysis of scheduling of tasks, the multivariate scheduling problem with constraints of mutual exclusive access to shared resources was modeled in this paper, and the blocking time of tasks under the random task stealing mechanism and schedulability loss generated by adaptive scheduling were analyzed. On the basis of feedback control theory and with the use of online competition optimization theory, an adaptive co-scheduling framework with function of dynamic perception of task parallelism degree was established. The following conclusions were drawn:

(1) Through the abstraction of the task scheduling problem of multi-core processor systems into an online scheduling problem model, an online competition analysis method was used for analysis; results indicate that the adaptive scheduling algorithm was of minor time complexity.
(2) The proposed resource adaptive scheduling algorithm for dynamic perception of task parallelism degree could realize stability and convergence rate of closed-loop feedback control model through control theory z-transformation rule and transfer function.
(3) Online analysis could acquire the communication and synchronization relationship between multi-core threads. Random work stealing scheduling strategy realized by dynamic compilation technique had less scheduling cost and was more applicable to multi-core processing systems.

An adaptive scheduling method with dynamic perception of core resources was proposed in this paper. This method is of significant applicability to the issue of performance optimization of multi-core processors in the future. A study was conducted with focus on homogeneous multi-core processor systems. Homogeneous multi-core processors are gradually maturing with technological progress. Thus, the next step is to study an adaptive scheduling method that is applicable to heterogeneous multi-core processor systems and is based on the proposed adaptive work stealing scheduling strategy to realize load balancing of multi-core processor systems under asymmetric processing performance.

Acknowledgements
This work was supported by the Science and Technology Development Plan of Jilin Province under the project No. 20150204005GX, the Significant Science and Technology Plan of Changchun City under the project No. 14KG082, the Industrial Technology Research and Development Special Project of Jilin Province under Grant No. 2011006-9. This work was also supported by the National Natural Science Foundation of China under the project No.U1304603, No. 11301488, and No. 61472049.

References

1.Pile D., "Microprocessors: electronic-photonic chip", *Nature Photonics*, 10(3), 2016, pp. 145-145.
2. Nandivada V K., Shirako J., Zhao J., Sarkar V., "A transformation framework for optimizing task-parallel programs", *ACM Transactions on Programming Languages and Systems*, 35(1), 2013, pp. 1-48.
3. Dongarra J., Abalenkovs M., Abdelfattah A., "Parallel programming models for dense linear algebra on heterogeneous systems", *Supercomputing Frontiers and Innovations*, 2(4), 2016, pp. 67-86.
4. Gropp W., Thakur R., "Thread-safety in an MPI implementation: requirements and analysis", *Parallel Computing*, 33(9), 2007, pp. 595-604.
5. Kumar R., Moseley B., Vassilvitskii S., "Fast greedy algorithms in mapreduce and streaming", *ACM Transactions on Parallel Computing*, 2(3), 2015, pp. 14-28.
6. Ayguadé E., Copty N., Duran A., Hoeflinger J., Lin Y., Massaioli F., Teruel X., Unnikrishnan P., and Zhang G., "The design of openMP tasks", *IEEE Transactions on Parallel and Distributed Systems*, 20(3), 2008, pp. 404-418.
7. Streit K., Doerfert J., Hammacher C., "Generalized task parallelism", *ACM Transactions on Architecture and Code Optimization*", 12(1), 2015, pp. 1-25.
8. Chamberlain B., Callahan D., and Zima H., "Parallel programmability and the chapel language", *International Journal of High Performance Computing Applications*, 21(3), 2007, pp. 291-312.
9. Allen E., Chase D., Hallett J., Luchangco V., Maessen J., Ryu S., Steele Jr G., Tobin-Hochstadt S., Dias J., Eastlund C., "The fortress language specification", *Sun Microsystems*, 139(1), 2005, pp. 140-152.
10. Cunningham D., Grove D., Herta B., "Resilient X10: efficient failure-aware programming", *ACM SIGPLAN Notices*, 49(8), 2014, pp. 67-80.
11. Leijen D., Schulte W., Burckhardt S., "The design of a task parallel library", *ACM SIGPLAN Notices*, 44(10), 2009, pp. 227-242.
12. Zambas C., Luján M., "Introducing aspects to the implementation of a java fork/join framework", *Algorithms and Architectures for Parallel Processing*, 5022(1), 2008, pp. 294-304.
13. Pheatt C., "Intel threading building blocks", *Journal of Computing Sciences in Colleges*, 23(4), 2008, pp. 298-298.
14. Leijen D., Schulte W., Burckhardt S., "The design of a task parallel library," *ACM SIGPLAN Notices*, 44(10), 2009, pp. 227-242.
15. Blumofe R D., Joerg C F., Kuszmaul B C., Leiserson C E., Randall K H., Zhou Y., "Cilk: An efficient multithreaded runtime system," *Journal of Parallel and Distributed Computing*, 37(1), 1996, pp. 55-69.
16. Leiserson C., "The Cilk++ concurrency platform," *The Journal of Supercomputing*, 51(3), 2010, pp. 244-257.
17. Chapman B., Huang L., "Enhancing openMP and its implementation for programming multicore systems", *Parallel computing: architectures, algorithms*, and applications, 15(2), 2008, pp. 3-18.
18. Broquedis F., Diakhaté F., Thibault S., Aumage O., Namyst R., Wacrenier P., "Scheduling dynamic openmp applications over multicore architectures", *OpenMP in a New Era of Parallelism*, 5004(1), 2010, pp. 170-180.
19. Massaioli F., Filippo C., Massimo B., "OpenMP parallelization of agent-based models", *Parallel Computing*, 31(10), 2005, pp: 1066-1081.
20. Duran A., "A proposal to extend the openMP tasking model with dependent tasks", *International Journal of Parallel Programming*, 37(3), 2009, pp. 292-305.
21. Liao C., Hernandez O., Chapman B., Chen W., Zheng W., "Openuh: An optimizing, portable openMP compiler", *Concurrency and Computation: Practice and Experience*, 19(18), 2007, pp. 2317-2332.
22. Yu F., Yang S C., Wang F., "Symbolic consistency checking of OpenMp parallel programs", *ACM SIGPLAN Notices*, 47(5), 2012, pp: 350-355.
23. Edmonds J., "Scheduling in the dark (improved result)," *Theoretical Computer Science*, 235(1), 2007, pp. 109-141.