COMMUNICATION BETWEEN COROUTINES ON SUNWAY MANY-CORE PLATFORM

Shaodi Li[1], Junmin Wu[2], Yi Zhang[2] and Yawei Zhou[2]

[1]School of Software Engineering, University of Science and Technology of China, Suzhou, China
[2]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

ABSTRACT

Communication between parallel programs is an indispensable part of parallel computing. SW26010 is a heterogeneous many-core processor used to build the Sunway TaihuLight supercomputer, which is well suited for parallel computing. Our team is designing and implementing a coroutine scheduling system on SW26010 processor to improve its concurrency, it is very important and necessary to achieve communication between coroutines for the coroutine scheduling system in advance. Therefore, this paper proposes a communication system for data and information exchange between coroutines on SW26010 processor, which contains the following parts. First, we design and implement a producer-consumer mode channel communication based on ring buffer, and designs synchronization mechanism for condition of multi-producer and multi-consumer based on the different atomic operation on the MPE (management processing element) and the CPE (computing processing element) of SW26010. Next, we design a wake-up mechanism between the producer and the consumer, which reduces the waiting of the program for communication. At last, we test and analyse the performance of channel in different numbers of producers and consumers, draw the conclusion that when the number of producers and consumers increases, the channel performance will decrease.

KEYWORDS

Coroutine, SW26010, Many-core, Parallel Communication, Synchronization

1. INTRODUCTION

The SW26010 is a heterogeneous many-core processer, which is a china-made high-performance processer developed by Wuxi Jiangnan Institute of Computing Technology. It belongs to the Sunway series. It has good performance in super computer and high performance computing area. And it is the main building-block of the current world's third fastest supercomputer: Sunway Taihu Light[1]. This processor has been used in many fields of high performance computing, such as computational mechanics[2], bioinformatics[3], deep learning[4] and so on. But for a long time, application development on this processor has several difficulties such as high learning costs, highly associated with hardware, hard to migrate and so on. A SW26010 processor consists of 4 management processing elements (MPE, also called master core) and 256 computing processing elements (CPE, also called slave core). However, one CPE of SW26010 processor can only run one thread and it doesn't support blocking and switching, which limits its parallel ability. Therefore, our team uses the idea of coroutine, and designs a coroutine running framework on SW26010 processor to replace the direct use of threads on the CPE, which breaks

David C. Wyld et al. (Eds): ITCSE, NLCA, ICAIT, CAIML, ICDIPV, CRYPIS, WiMo - 2020
pp. 57-67, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.100505

through the parallel restriction of Sunway many–core processor, and makes the upper applications be run more efficiently.

As an indispensable part of the coroutine running framework, the communication between coroutines need to be discussed. Since the communication between threads on Sunway many-core processor is mainly based on batch data transfer, and there is no fine-grained communication method suitable for ordinary programs, this paper designs a channel communication method which can exchange messages between coroutines on either MPE or CPE of Sunway processor, and provides a guarantee for cooperation of parallel coroutines.

This paper includes the following parts: First, the channel communication in producer-consumer mode is implemented based on ring buffer, and then, to ensure that no errors occur on the condition of multi-producer or multi-consumer competing with each other, the mechanism of synchronization is designed based on the different atomic operations on the MPE and CPE, which ensures the correctness of data transmission. Next, we design a wake-up mechanism of producer and consumer, which reduces the waiting of the program for communication. At last, this paper tests the performance of channel in different numbers of producers and consumers.

2. BACKGROUND AND RELATED WORK

2.1. SW26010 Many-Core Processor

The SW26010 is a heterogeneous many-core processor independently developed and designed by Wuxi Jiangnan Institute of Computing Technology of China. The heterogeneous many-core architecture combining on-chip computing array cluster and distributed shared storage. Sunway many-core processor is commonly used for the execution of high-performance computing programs, its hardware architecture is shown in Figure 1.

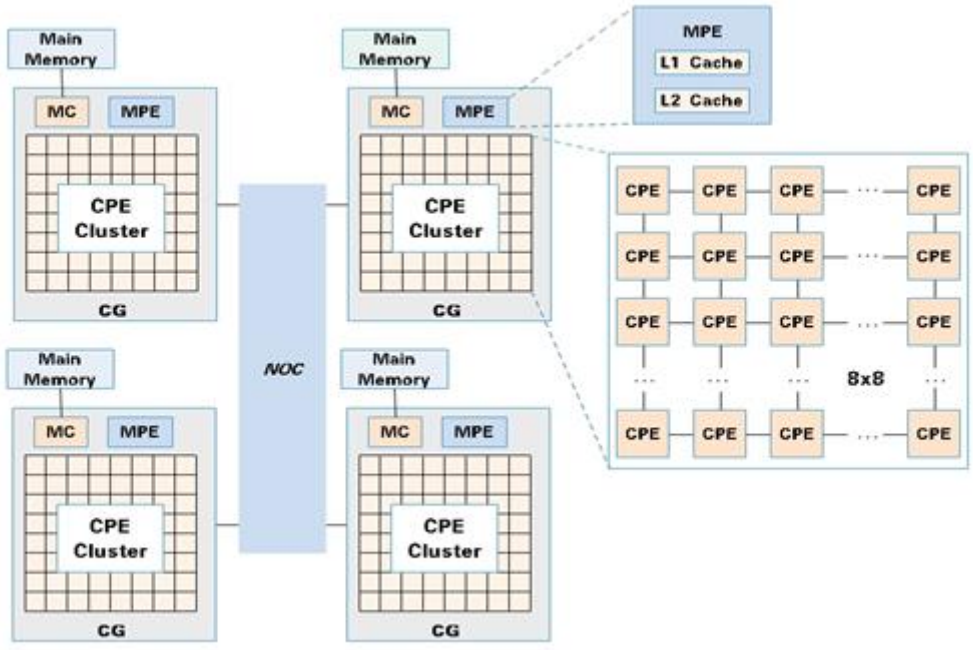

Figure 1. SW26010 processor.

Each SW26010 chip contains 260 cores, which are divided into four core groups (CGs). Each core group contains a management processing element (MPE, or master core), and subordinate 64 computing processing elements (CPE, or slave core). The frequency of the MPE and CPE is 1.45GHz. 64 CPEs are combined into a CPE cluster organized as an 8×8 mesh. Each core group is connected with an 8GB memory by a memory controller (MC), and the four core groups are connected by the network on-chip (Noc). SW26010 processor is designed based on the alpha instruction set, in which the MPE support complete alpha instruction set, while the CPE support simplified alpha instruction set. The whole chip can provide computing peak performance over 3TFlops. As for the storage structure, the MPE and the CPE both can access the main memory, Each MPE has 32KB L1 data cache and 256 KB L2 instruction/data cache to ensure the high efficiency of read and write to the main memory, while the CPE has no cache for memory read and write, resulting in inefficient access to main memory. But each CPE contains a 64KB local device memory (LDM), which can store the data needed for program running on the core. Each CPE can read and write its own LDM quickly, but cannot access LDM of other CPEs. A CPE can copy data from main memory to its LDM or write data back to main memory in batches by Direct Memory Access (DMA), which is the main data transmission method in high performance computing programs.

The operating system of SW26010 is a customized Linux flavour running on the first MPE, C/C++ and FORTRAN programs are supported on the MPE and C, FORTRAN programs are supported on the CPE. The MPE and CPE of Sunway many-core processor have different running environments, so the programs on the MPE and CPE need to be compiled separately, and then packaged in a single executable file by mixed compilation, finally submitted to work queue for execution. It can be seen from the calculation structure of SW26010 processor that the computing power of CPEs accounts for more than 98% of the computing power of the whole chip, so the development of application on SW26010 processor needs to give full play to the computing power of CPEs. In general, application development on SW26010 is based on the parallel execution of the MPE and CPE. Computing tasks are divided into small blocks and assigned to CPEs to execute, and the MPE executes communications or other parts that the CPE cannot run. This way, the core computing part of the program will be executed by the CPE, and the MPE only responsible for management part.

2.2. Implementation of Coroutine on SW26010 Processor

Coroutine is a user-controlled way of switching programs and achieving concurrency without operating system scheduling. The concept of coroutine is not complex. The basic principle is that when a program is running, it can actively give up its own control of running so that the thread can switch to other programs. Therefore, there are some simple coroutines implementations[5]. However, a good implementation of coroutine requires more detailed design in terms of scheduling and communication[6]. Owing to less system resource costs than threads, coroutine is often used in high-concurrency scenarios such as web crawlers, distributed system[7], simulation mechanism[8] and so on. In SW26010 processor, there are only one thread runs on a CPE. This scheme doesn't support blocking and switching, and limits its parallel ability. The use of coroutine can break through the concurrency restriction of the CPE, and can achieve multiple concurrency on a CPE only with one single thread. So our team decided to develop a framework of coroutine on SW26010 processor. Based on the master-slave parallel structure of SW26010 processor, we design and implement a coroutine library which combines dispatch, execution, communication and other modules. The coroutine framework based on the athread interface provided by SW26010, using thread on the CPE as coroutines instead of using it directly. In this way, upper applications can achieve higher concurrency and gain more efficiency. Coroutines on SW26010 processor is shown in Figure 2.

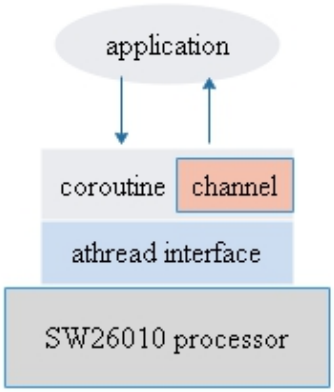

Figure 2. Coroutines on SW26010 processor

The implementation of coroutine on SW26010 Processor consists of the following parts:

1. Scheduler: The scheduler is running on the MPE, which creates a coroutine, initializes the coroutine, and assigns the coroutine to an execution queue of an executor on a CPE, waiting for the executor to execute.

2. Executor: Executor run on the CPE, and a CPE can only run one executor so each core group contains 64 executors, executors can execute specific programs. Each executor contains two queues, one is a runnable queue, the other is a wait queue, runnable queue contains coroutines that can be executes, wait queue contains coroutines blocked because of communication or other reason.

3. Communication module: If coroutines needs to cooperate with each other, they need to communicate and exchange data. The communication module of coroutine is called channel, a coroutine can send messages to others by using channel. This paper is mainly introduce the communication module.

3. DESIGN OF COMMUNICATION BETWEEN COROUTINES

3.1. Data Structure of Channel

Channel's data structure is based on ring buffer. Ring buffer[9] is a first-in-first-out data structure that reduces duplicate address operations and increases stability relative to queues[10]. Ring buffer is widely used in various fields[11]. It is easy to separate data writing from reading by using ring buffer, avoids competition between reading threads and writing threads, and reduces using of locks. We use ring buffer as channel’s infrastructure. The working principle of ring buffer is shown in the Figure 3:

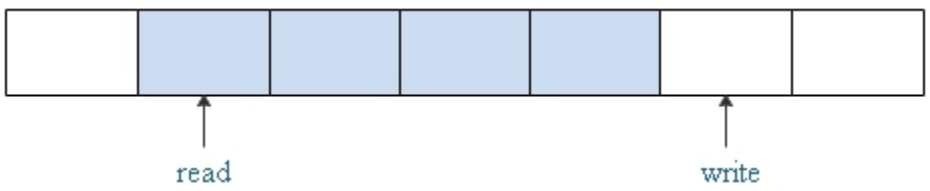

Figure 3. Ring buffer structure

As shown in the Figure 3, in a fixed size buffer, there are two pointers: read and write. When some data is written to buffer, write increases. When some data is read from buffer, read increases. Using ring buffer, we can simply realize producer-consumer mode. Because the producer only affects the write pointer and the consumer only affects the read pointer, when there is only one producer and one consumer, we do not need to lock the buffer, which increases the efficiency of communication. The channel structure including ring buffer is as follows:

```
1.  typedef struct {
2.      char *buffer;
3.      int capacity;
4.      int elem_size;
5.      int read;
6.      int write;
7.      int to_read;
8.      int to_write;
9.      list read_queue;
10.     list write_queue;
11. }channel;
```

Figure 4. Data structure of channel

In the channel structure, buffer refers to the buffer where data is stored, elem_size refers to the size of a message, capacity refers to the maximum number of messages stored in the buffer, write refers to the location where the message will be written, read refers to the next message can be read, to_read and to_write are used to ensure parallel synchronization when multiple producers or consumers are involved, which will be described in the next section in detail. The two lists are used to store coroutines waiting on the channel when send or receive fails, which will be described in Section 3.3. The data structure of channel is stored in main memory, so that both the MPE and the CPE can access.

3.2. Design of Parallel Synchronization Mechanism

With ring buffer, messages can be delivered safely without synchronization mechanism in the case of single producer and single consumer. But when there are multiple producers or consumers, the contention of multiple threads for the same data may result in data coverage. Therefore, we need to take certain measures to ensure that the data in the channel is correct[12]. In x86 instruction computers, CAS (compare and swap) atomic operation is often used to deal with multithreading competition[13]. In SW26010 processor, the MPE and CPE have different degrees of instruction support. CAS operation is supported on the MPE, but not on the CPE. We first use CAS operation to deal with multithreading competition on the MPE.

3.2.1. Parallel Synchronization Mechanism on the MPE

CAS (compare and swap) can compare and exchange data in one instruction, which is commonly used in the unlocked algorithm. Its common form is as follows:

CAS (dest, oldval, newval)

Where dest is the data address, oldval is the current value, and newval is the new value. When the value pointed to by dest is equal to oldval, the value will be updated to newval and true will be returned, otherwise it will not be updated and false will be returned. When two threads use CAS instruction at the same time, only one thread can succeed, and other threads will fail, thus we ensures that only one thread can complete CAS operation and process data. Using CAS operation to build the parallel synchronization mechanism of message sending is as follows:

```
1.    do {
2.        if (full(chan)){
3.            co_swap_out();
4.        }
5.        temp = chan->write;
6.        next = temp + 1;
7.        ok = CAS(&chan->write, temp, next);
8.    } while (!ok);
9.    //copy data here
```

Figure 5. Message send with parallel synchronization on the MPE

When sending data to the channel, the producer first determine whether the channel is full. If channel is full, the producer cannot send a message to the channel, then it gives up the control right and let other coroutines run. If the channel is not full, the producer first reads the write pointer and then updates the value of write with CAS operation. If it succeeds, it means that no other coroutines successfully change the write value, a message can be sent to buffer according to the write pointer. Note that while other coroutines may operate on write values when current coroutine writes data to the buffer, data coverage will not occur because we have determined the write location in the buffer. In this way, we ensure the synchronization of channel message sending on the MPE.

3.2.2. Parallel Synchronization Mechanism on the CPE

While CAS can be used to realize the synchronization of parallel programs and ensure the correctness of communication on the MPE, it is not supported on the CPE. There is only one atomic operation supported on the CPE, which can modifies data. Its interface is as follows.

updt(_n_, _addr_)

This operation represents adding _n_ to the data pointed by _addr_. Parallel synchronization of channel is more difficult to achieve on the CPE because the atomic operation changes data directly without comparison. This paper uses the mechanism shown below to synchronize channels, as shown in Figure 6.

```
1.    while (1){
2.        if (full (chan)){
3.            co_swap_out();
4.        }
5.        temp = chan->write;
6.        if (temp == chan->to_write){
7.            updt_addw(1,&(chan->write));
8.        } else {
9.            continue;
10.       }
11.       if (chan->write == temp + 1){
12.           //copy data here
13.           updt_addw (1,&chan->to_write);
14.           return 0;
15.       } else {
16.           updt_addw (-1,&chan->write);
17.           continue;
18.       }
19.   }
```

Figure 6. Message send with parallel synchronization on the CPE

In order to synchronize the writing of buffer using atomic operations, to_write, a comparison of the write pointer is introduced. When to_write is equal to write, it indicates that no coroutine is sending messages to the channel. When they are not equal, it indicates that a coroutine is sending. At the beginning, we read the value of variable write, save it in the variable temp, and compare it with the value of to_write. If they are not equal, it means that other producer has modified the value of the write. At this time, the write value should be read again. If they are equal, it means that other producer has finished sending to the channel, then this producer will modify the value of write. Since comparison and update cannot be done in one instruction, comparison and update may still be performed by two producers in the order of 6.-6.-7.-7., there is still a case where two producers have modified the value of variable write, so we read the value of variable write again and compare it with the value saved by the local variable temp. If the current value of variable write equals to temp+1, which indicates that only one atomic operation has been performed, message can be send to channel in next line, and the value of variable to_write can be updated to complete one message sending. If the value of variable write is not equal to temp+1 at this time, it means that other threads have also made atomic updates, this producer should reduce the value of variable write by atomic update, let the write value revert to the state that was before this producer accessed. This send can be considered a failure, and we do it again from the beginning. In this way, we ensure that when multiple producers send data to the channel, at most one producer can find that after atomic operation, the value of variable write equal to the value of variable temp+1, and other producers will fail. Thus, the synchronization of messages in the channel is ensured.

Although the synchronization mechanism on the CPE can also ensure that data will not be overwritten or be read repeatedly, it is more complex than the CAS operation on the MPE, and has the possibility of invalid operation, so the performance loss is higher than that of the MPE.

3.2.3. Different Modes of Channels

Although using the synchronization mechanism can ensure the correctness of the messages in channel, it will also result in decreasing the communication efficiency. Therefore, in order to maximize the communication efficiency, this paper designs different communication modes for different number of producers and consumers. Different modes can be chosen according to the actual needs to maximize the efficiency of communication. There are four modes in total:

Single producer-single consumer: only one producer and one consumer. In this case, there is no synchronization, and the efficiency is the highest.

Single producer- multi consumer: only one producer, but multiple consumers. In this case, the consumer read buffer needs to be synchronized, but the producer can send messages directly.

Multi producer-single consumer: multiple producers, but only one consumer. In this case, the producer write buffer needs to be synchronized, and the consumer can receive messages directly.

Multi producer-multi consumer: multiple producers and multiple consumers. In this case, both reading and writing of buffer needs to be synchronized, which is also the default mode of channel.

3.3. Blocking and Wakeup Mechanism of Channel

In the process of communication, sometimes the program wants to communicate cannot communicate normally, for example, the producer cannot send message when the channel is full. At that time, the program has no choice but to wait. When it is implemented in multi-threaded mode, the mechanism of cyclic access or thread switching can be chosen. However, the thread switching consumes much system resources, which will lead to the performance degradation. But for the program based on coroutines, the switching consumes less resources, we choose to let the coroutine block and switch when the communication cannot be carried out. If a coroutine is blocked and switched off running queue, other coroutines is going on running, which reduces the time cost for waiting. When a producer sends messages to channel, it will first determine whether the channel buffer is full. If it is not full, it will send a message and continue to run. If it is full, the message cannot be sent, and the producer coroutines will enter a block, and the executor will transfer the execution right to other coroutines to run. The blocked coroutine will be recorded on the waiting queue of the channel. The blocked coroutine does not wake up automatically or be awakened by executor, but it wakes up when a consumer takes a message out of the channel so that the channel is no longer full. At that point, the blocked coroutine back to the running queue to continue run, and has a high probability successfully send messages. Similarly, when the channel is empty, the consumer coroutine will also be blocked and be awakened by a producer coroutine. This kind of mutual wake-up mechanism allows a program to directly give up the execution right in the case of unable to communicate, and let other coroutines run instead of waiting in a loop. It also does not consume a lot of system resources like thread switching, and effectively uses the operation ability of the processor.

4. EXPERIMENTAL RESULTS AND ANALYSIS

4.1. Performance Test of Channel

In order to understand the specific performance of channel communication, it is necessary to test the operation performance of channel under different conditions. This paper uses such a process to simulate the use of channel under real conditions:

First, a channel is created. In order to get the minimum time of one message sending and receiving, and to test the highest performance of channel, the capacity of channel in this experiment is large enough to accommodate all message data, so there will be no blocking. Then, the producer produces messages. In this test, a randomly generated integer value is used as the message data, and the message is sent to the channel by the producer many times, the average sending time is taken as the time of one message sending. Next, the consumer obtains data from the channel, and the average receiving time is taken as the time of one message receiving. Since channel has different modes, which will affect the competition between producers or consumers, we test channel performance under different conditions, which including different cores and different number of producers and consumers. And the result is shown in Table 1.

Table 1. Sending/Receiving time under different conditions.

number of producer/consumer	1	10	32
MPE send	0.13μs	0.22μs	0.22μs
MPE receive	0.12μs	0.23μs	0.22μs
CPE send	1.37μs	30μs	457μs
CPE receive	1.95μs	54μs	791μs

4.2. Analysis

It can be seen from Table 1 that the communication efficiency on the CPE is lower than that on the MPE, and with the increase of producers/consumers, the sending/receiving time for one message increase whether on the MPE or CPE, but the increase on the CPE is much higher than that on the MPE. The process time of one sending/receiving on the MPE is about the same when there are 10 and 32 producers/consumers, but on the CPE, the sending/receiving time of 32 producers/consumers is much longer than that of 10 producers/consumers.

There are two reasons why the communication efficiency of the CPE is lower than that of the MPE. First is that the speed accessing main memory from the CPE is lower than that of the MPE. Second, the synchronization mechanism on the CPE will cause much more decrease of efficiency when producer or consumer increases. More processes competing, higher the probability of invalid operation, then the average communication time increases.

5. CONCLUSIONS

In this paper, we design the producer-consumer mode inter-core channel communication based on the coroutine implementation on SW26010 processor. We design the data structure based on ring buffer, the synchronization mechanism based on different atomic operations of the MPE and CPE, and the mechanism of mutual wake-up between producers and consumers, so that the security and efficiency of communication are guaranteed. At last, we test and analyse the performance of channel, draw the conclusion that the performance of channel on the CPE is

lower than that on the MPE due to the lower efficiency of read/write to memory, and the channel performance will decrease when the number of producers and consumers increases.

This study provides an effective communication guarantee for the implementation of the coroutine on SW26010 processor, provides an efficient communication interface for the development of upper application, and improves the efficiency of program execution, and explores the communication capability of SW26010 processor. Because the performance of channel is mainly affected by the memory read/write efficiency and parallel synchronization, the future directions for research are to reduce the read/write to memory and reduce the competition between coroutines.

ACKNOWLEDGEMENTS

This work is supported by the National Key Research and Development Program of China under Grant 2018YFB1003601

REFERENCES

[1] Fu, H., Liao, J., Yang, J., Wang, L., Song, Z., Huang, X., ... & Zhao, W. (2016). The Sunway TaihuLight supercomputer: system and applications. Science China Information Sciences, 59(7), 072001.

[2] Duan, X., Xu, K., Chan, Y., Hundt, C., Schmidt, B., Balaji, P., & Liu, W. (2017, September). S-Aligner: Ultrascalable Read Mapping on Sunway Taihu Light. In 2017 IEEE International Conference on Cluster Computing (CLUSTER) (pp. 36-46). IEEE.

[3] Wang, X., Liu, W., Xue, W., & Wu, L. (2018, February). swSpTRSV: a fast sparse triangular solve with sparse level tile layout on sunway architectures. In Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (pp. 338-353).

[4] Fang, J., Fu, H., Zhao, W., Chen, B., Zheng, W., & Yang, G. (2017, May). swdnn: A library for accelerating deep learning applications on sunway taihulight. In 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 615-624). IEEE.

[5] Bailes, P. A. (1985). A low-cost implementation of coroutines for C. Software: Practice and Experience, 15(4), 379-395.

[6] Pauli, W., & Soffa, M. L. (1980). Coroutine behaviour and implementation. Software: Practice and Experience, 10(3), 189-204.

[7] Hui-Ba, L. I. , Yu-Xing, P. , & Xi-Cheng, L. U. . (2008). A programming pattern for distributed systems. computer engineering & science.

[8] Xu, X., & Li, G. (2012, October). Research on Coroutine-Based Process Interaction Simulation Mechanism in C++. In Asian Simulation Conference (pp. 178-187). Springer, Berlin, Heidelberg.

[9] Zhang-juna, Y. A. O., Shu-yub, C. H. E. N., & Yaoa, L. U. (2012). Research and Implementation of High-performance Ring Buffer [J]. Computer Engineering, 8.

[10] Feldman, S., & Dechev, D. (2015). A wait-free multi-producer multi-consumer ring buffer. ACM SIGAPP Applied Computing Review, 15(3), 59-71.

[11] Bergauer, H., Jeitler, M., Kulka, Z., Mikulec, I., Neuhofer, G., Padrta, M., & Taurok, A. (1996). A 1-GHz Flash-ADC module for the tagging system of the CP-violation experiment NA48. Nuclear

Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 373(2), 213-222.

[12] He, Z. (2012). On algorithm design and programming model for multi-threaded computing (Doctoral dissertation, Georgia Institute of Technology).

[13] Michael, M. M. (2003, August). CAS-based lock-free algorithm for shared deques. In European Conference on Parallel Processing (pp. 651-660). Springer, Berlin, Heidelberg.

AUTHOR

Shaodi Li, postgraduate student of School of Software Engineering, University of Science and Technology of China. Research direction is parallel computing and communication.