**Indonesian Journal of Electronics, Electromedical, and Medical Informatics (IJEEEMI)**
**Vol. 3, No. 1, February 2021, pp. 15-20**
**DOI: 10. 35882/ijeeemi.v3i1.3** **ISSN: 2656-8624**

# Communication Coroutines For Parallel Program Using DW26010 Many Core Processor

Ajit Singh
Department of Computer Science, Patna Women's College
Patna University, Patna 800004, Bihar, India

**Article Information**

**Article History:**
Received Feb 5, 2021
Revised Feb 19, 2021
Accepted Feb 21, 2021

***Keywords:***
Coroutine
SW26010
Many-core
Parallel Communication
Synchronisation

**Abstract**

**Communication between parallel programs is an indispensable part of parallel computing. SW26010 is a heterogeneous many-core processor used to build the Sunway Taihu Light supercomputer, which is well suited for parallel computing. There is the designing and implementing a coroutine scheduling system on the 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. The designing and implementation a producer-consumer mode channel communication based on ring buffer, and it designs synchronization mechanism for condition of multi-producer and multi-consumer based on the different atomic operation on MPE (management processing element) and CPE (computing processing element) of SW26010. There is also the designing of a wake-up mechanism between the producer and the consumer, which reduces the waiting of the program for communication. The testing and analysis of 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.**

**Corresponding Author**:
Ajit Singh
Ajit_singh24@yahoo.Com
Department of Computer Science
Patna Women's College, Patna University, India

## I. INTRODUCTION

Performance processer developed by Wuxi Jiangnan Institute of Computing Technology. It belongs to Sunway series. It has good performance in super computer and high-performance computing area. And it is the main building-block of the current worlds 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 the processor has several difficulties like high learning costs, highly associated with hardware, hard to migrate and so on. A SW26010 processor consists of 4 management processing element (MPE, also called master core) and 256 computing processing elements (CPE, also called slave core). However, the slave core of the 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 CPEs, which breaks through the parallel restriction of Sunway many core processors, 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 master and slave core, which ensures the correctness of data transmission. Next, i 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.

## II. BACKGROUND AND RELATED WORK

### *A. SW26010 many-core processor*

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 is adopted. Sunway multi-core processor is commonly used in the execution of high-performance computing programs. Its hardware architecture is shown in the Fig. 1.

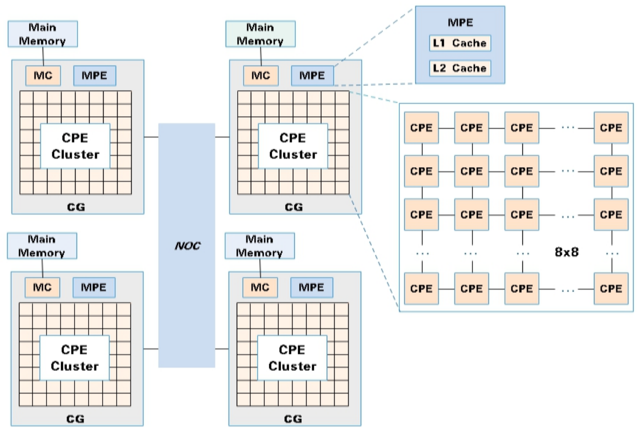

Fig. 1. SW26010 processor

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 the subordinate 64 computing processing elements (CPE, or slave core). The frequency of the master and slave core is 1.45GHz. 64 slave cores 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). The Sunway 26010 processor is designed based on the alpha instruction set, in which the master core supports the complete alpha instruction set, while the slave core supports the simplified alpha instruction set. As for the storage structure, the master and slave cores can both access the main memory, Each MPE has 32KB L1 data cache and 256 KB L2 instruction/data cache to ensure fast read and write of the main memory, while the slave core has no cache for memory read and write, resulting in inefficient access to memory. But each slave core contains a 64KB local device memory (LDM), which can store the data needed for program running on the core. Each slave core can read and write its own LDM quickly, but cannot access LDM of other slave core, the slave core can copy data from the main memory to LDM or write it back in batches by Direct Memory Access (DMA). The whole chip can provide computing peak performance over 3TFlops.

The operating system is a customized Linux flavour running on the MPE, C/C++ and FORTRAN programs are supported on MPE and C, FORTRAN programs are supported on CPE. The MPE and CPE on Sunway 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 the SW26010 processor that the computing pair of CPEs accounts for more than 98% of the computing pair of the whole chip, so the development of application on SW26010 processor needs to give full play to the computing pair of CPEs. In general, application development on SW26010 is based on the parallel execution of 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 CPEs cannot run. This way, the core computing part of the program can be executed by CPEs, and MPE only responsible for management part.

### *B. 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 ib crawlers, distributed system [7], simulation mechanism [8] and so on. For the SW26010 processor, there are only one thread runs on a CPE. This scheme doesn't support blocking and switching, and limits its parallel ability.

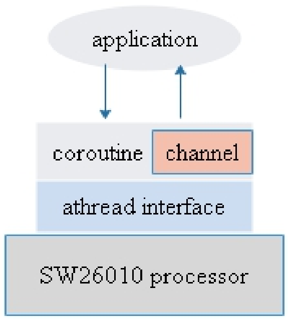

Fig. 2. Coroutines on SW26010 processor

The use of coroutine can break through the concurrency restriction of CPE, and can achieve multiple concurrencies on a CPE only with one single thread. So, our team decided to develop a framework of coroutine on the SW26010 processor. Based on the master-slave parallel structure of the SW26010 processor, We design and implement a coroutine library which combines dispatch, execution, communication and other modules. The coroutine framework based on the a thread interface provided by SW26010, using threads on CPEs 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 Fig. 2. The implementation of coroutine on the SW26010 Processor consists of the following parts:

1. Scheduler: The scheduler is run on MPE, which creates a coroutine, initializes the coroutine, and assigns the coroutine to the execution queues of different executors on CPEs, waiting for the executor to execute.
2. Executors: Executors are run on CPEs, 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.

## III. Design of Communication between coroutines

### A. 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. I use ring buffer as channel's infrastructure. The working principle of ring buffer is shown in the Fig. 3:

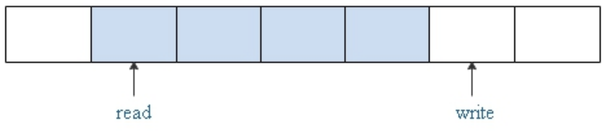

Fig. 3. Ring buffer structure

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, i 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, I do not need to lock the buffer, which increases the efficiency of communication. The channel structure including ring buffer is as follows:

```
typedef struct {
char *buffer;
int capacity;
int elem_size;
int read;
int write;
int to_read;
int to_write;
list read_queue;
list write_queue;
}channel;
```

In the channel structure, buffer refers to the buffer where data is stored, 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 MPE and CPE can access.

### B. 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, i 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, MPE and CPE have different degrees of instruction support. CAS operation is supported on MPE, but not on CPE. i first use CAS operation to deal with multithreading competition on MPE.

#### 1) Parallel synchronization mechanism on 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 i 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:

```
do {
    if (full(chan)){
        co_swap_out();
    }
    temp = chan->write;
    next = temp+1;
    ok = CAS(&chan->write, temp, next);
} while (!ok);
//copy data here
```

When sending data to the channel, i first determine whether the channel is full. If it is full, it is unable to send data to the channel, which gives up the control right and let other coroutines run. If the channel is not full, it first reads the write pointer and then updates the write value with CAS operation. If it succeeds, it means that no other coroutines successfully change the write value, data 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 i have determined the write location in the buffer.

*2) Parallel synchronization mechanism on CPE*

While CAS can be used in MPE to realize the synchronization of parallel programs and ensure the correctness of communication, it is not supported on CPE. There is only one atomic operation supported on 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 CPE because the atomic operation changes data directly without comparison. This paper uses the mechanism shown below to synchronize channels, as shown in the listing program.

```
while (1){
    if (full(chan)){
        co_swap_out();
    }
    temp = chan->write;
    if (temp == chan->to_write)}
        updt_addw(1,&(chan->write));
    } else {
        continue;
    }
    if (chan->write == temp+1){
        //copy data here
        updt_addw (1,&chan->to_write);
        return 0;
    } else {
        updt_addw (-1,&chan->to_write);
        continue;
    }
}
```

In order to synchronize the writing of buffer using atomic operations, to write, a comparison of the write pointer is introduced. When to write and write are equal, 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, i 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 i 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 writes 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 i 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 CPE can also ensure that data will not be overwritten or be read repeatedly, it is more complex than the CAS operation on MPE, and has the possibility of invalid operation, so the performance loss is higher than that of MPE. III

*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.

## *C. 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, i 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.

## IV. RESULTS AND ANALYSIS

## *A. 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. Since channel has different modes, which will affect the competition between producers and consumers, we test the situation with and without competition, on MPE and on CPE. The used message data is an integer, with the average of multiple send times as a result.

TABLE I. CHANNEL COMMUNICATION PERFORMANCE WITH DIFFERENT PRODUCER AND CONSUMER NUMBER.

| **Time (µs)** | **1(µs)** | **10(µs)** | **32(µs)** |
|---|---|---|---|
| MPE send | 0.13 | 0.22 | 0.22 |
| MPE receive | 0.12 | 0.23 | 0.22 |
| CPE send | 1.37 | 30 | 457 |
| CPE receive | 1.95 | 54 | 791 |

It can be seen that the communication efficiency of the slave core is lower than that of the main core, and the performance degradation is more serious when the multi-core competes. From the results we can draw a conclusion that the more producers or consumers compete, the more serious the communication efficiency degradation is.

There are two reasons why the communication efficiency of CPE is lower than that of MPE. First is that the speed of accessing the main memory from CPE is lower than that of MPE. Second, the synchronization mechanism on CPEs can 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.

## V. CONCLUSIONS

In this paper, we design the producer-consumer mode inter-core communication based on the coroutine implementation on SW26010 processor. For the channel mode communication that suitable for both MPE and CPE, i design the data structure based on ring buffer, the synchronization mechanism based on different atomic operations of 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 analyze 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. 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.

**CONFLICT OF INTEREST:** There is no conflict of interest in the paper.

## REFERENCES

[1] Fu, H. , Liao, J. , Yang, J. , Wang, L. , Song, Z. , & Huang, X. , et al. (2016). The sunway taihulight supercomputer: system and applications. *Science China Information Sciences*, 59(7).

[2] Duan, X. , Xu, K. , Chan, Y. , Hundt, C. , Schmidt, B. , & Balaji, P. , et al. (2017). S-Aligner: Ultrascalable Read

Mapping on Sunway Taihu Light. *IEEE International Conference on Cluster Computing.* IEEE.

[3] Wang, X. , Liu, W. , Xue, W. , & Wu, L. . (2018). swSpTRSV: a fast sparse triangular solve with sparse level tile layout on sunway architectures. *Acm Sigplan Symposium*. ACM.

[4] Fang, J. , Fu, H. , Zhao, W. , Chen, B. , & Yang, G. . (2017). swDNN: A Library for Accelerating Deep Learning Applications on Sunway TaihuLight. *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE.

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

[6] Pauli, W. , & Soffa, M. L. . (1980). Coroutine behaviour and implementation. *Software Practice & 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). Research on coroutine-based process interaction simulation mechanism in c++.

[9] Zhangdun, T. , Shuyu, C. , & Yao, L. . (2012). Research and implementation of high-performance ring buffer. *computer engineering*, 38(8), 228-231.

[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. , et al. (1996). A 1-ghz flash-adc module for the tagging system of the cp-violation experiment na48. *Nuclear Instruments & Methods in Physics Research*, 373(2), 213-222.

[12] He, Z. . (2012). On algorithm design and programming model for multi-threaded computing. *Dissertations & Theses Gradworks*.

[13] Michael, M. M. . (2003). CAS-Based Lock-Free Algorithm for Shared Deques. *Euro-par Parallel Processing, International Euro-par Conference, Klagenfurt, Austria, August.* DBLP.