INTERCATHEDRA

3(44), 2020, 157–164

http://www.up.poznan.pl/intercathedra/
pISSN 1640-3622
eISSN 2545-2045

http://dx.doi.org/10.17306/J.INTERCATHEDRA.2020.00104
Accepted for print: 21.01.2021

Ajit Singh[1]✉

[1]**Patna Women's College, India**

COMMUNICATION COROUTINES FOR PARALLEL PROGRAM USING SW26010 MANY CORE PROCESSOR

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 of a coroutine scheduling system on the SW26010 processor to improve its concurrency, and it is very important and necessary that communication between coroutines for the coroutine scheduling system is achieved in advance. Therefore, this paper proposes a communication system for data and information exchange between coroutines on the SW26010 processor which contains the following parts. The designing and implementation of a producer-consumer mode channel communication based on ring buffer, and designing a synchronization mechanism for the condition of multi-producer and multi-consumer based on different atomic operations 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 time of the program for communication. The testing and analysis of the performance of a channel with different numbers of producers and consumers, leads to the conclusion that when the number of producers and consumers increases, performance of the channel will decrease.

Keywords: coroutine, SW26010, many-core, parallel communication, synchronisation

INTRODUCTION

The performance processer has been developed by Wuxi Jiangnan Institute of Computing Technology. It belongs to the Sunway series. It has good performance in supercomputers and in the field of high-performance computing. And it is the main building-block of the current world's third-fastest supercomputer: Sunway Taihu Light(Fu et al., 2016). This processor has been used in many fields of high-performance computing, such as computational mechanics (Duan et al., 2012), bioinformatics (Wang et al., 2018) deep learning (Fang et al., 2017) etc. But for a long time, application development on the processor presents several difficulties, such as high learning costs, being highly associated with hardware, hard to migrate etc. The SW26010 processor consists of 4 management processing elements (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 does not support blocking and switching, which limits its parallel ability. Therefore, our team uses

✉M. Phil. Computer Science Ajit Singh, Department of Computer Science, Patna 800001, Bihar India, e-mail: ajit_singh24@yahoo.com

the idea of coroutine, and designs a coroutine running framework on the SW26010 processor to replace the direct use of threads on CPEs, which breaks through the parallel restriction of the Sunway manycore processor and enables the upper applications to be run more efficiently.

The communication between coroutines needs to be discussed as an indispensable part of the coroutine running framework. Since the communication between threads on a Sunway manycore 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 that can exchange messages between coroutines on either MPE or CPE of a Sunway processor, and provides a guarantee for the cooperation of parallel coroutines.

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

BACKGROUND AND RELATED WORK

The 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 is adopted combining on-chip computing array cluster and distributed shared storage. The Sunway multi-core processor is commonly used in the execution of high-performance computing programs. Its hardware architecture is shown in Figure 1.

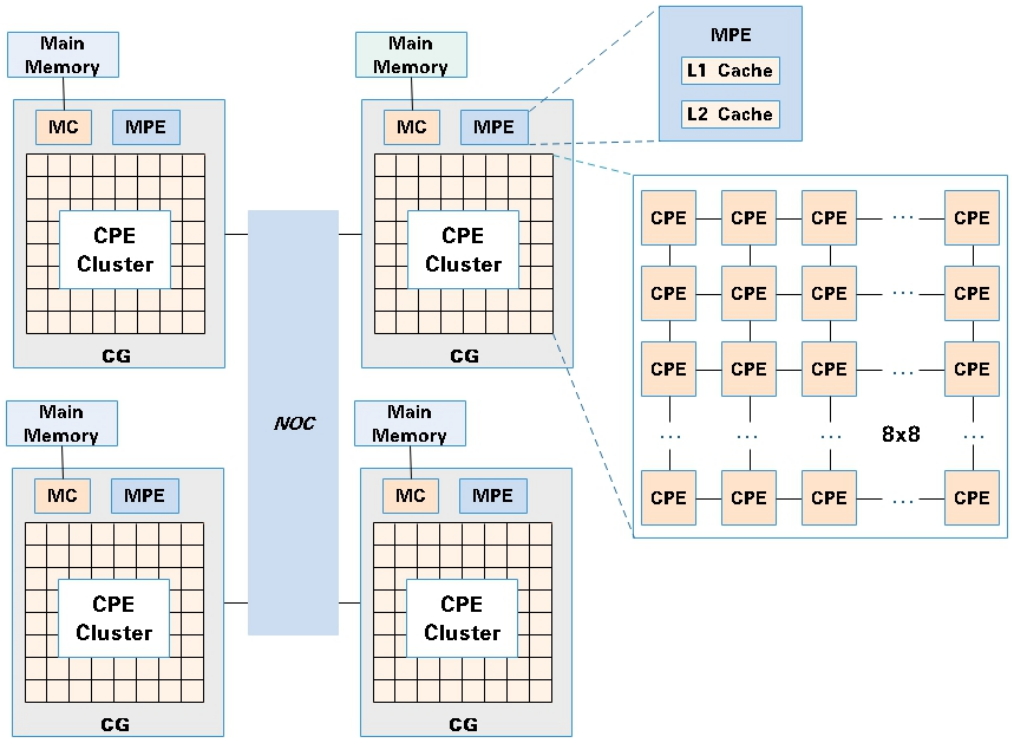

Fig. 1. SW26010 processor

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 64 subordinate 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 through a memory controller (MC), and the four core groups are connected through a 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 a 32KB L1 data cache and a 256 KB L2 instruction/data cache to ensure fast read and write operations 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 the LDM of other slave cores, a 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 flavor running on the MPE, C/C++ and FORTRAN programs are supported on MPE and C, as well as 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 using mixed compilation, finally submitted to a work queue for execution. It can be seen from the calculation structure of the 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 an 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 MPE and CPE. Computing tasks are divided into small blocks and assigned to CPEs to be executed, 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 is only responsible for the management part.

Implementation of coroutine on an SW26010 processor

A 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 coroutine implementations (Bailes, 1985). However, good implementation of a coroutine requires a more detailed design in terms of scheduling and communication (Pauli and Soffa, 1980). Owing to less system resource costs compared to threads, a coroutine is often used in high-concurrency scenarios such as ib crawlers, distributed system (Hui--Ba et al., 2008), simulation mechanism (Xu and Li, 2012) etc. For the SW26010 processor, only one thread runs on a CPE. This scheme does not support blocking and switching and it limits its parallel ability. The use of coroutine can break through the concurrency restriction of 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 the SW26010 processor. Based on the master-slave parallel structure of the SW26010 processor, i design and implement a coroutine library (Basiles, 1985) which combines dispatch, execution, communication and other modules. The coroutine framework based on the athread 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 Figure 2.

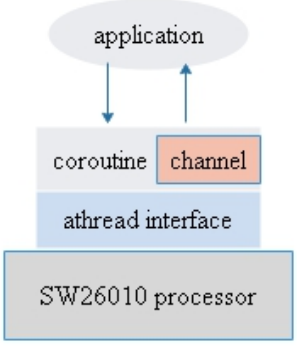

Fig. 2. Coroutines on SW26010 processor

The implementation of coroutine on the SW26010 Processor involves the following stages:

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, and executors can execute specific programs. Each executor contains two queues, of which one is a runnable queue, and the other is a wait queue. The runnable queue contains coroutines that can be executed, and the wait queue contains coroutines blocked because of communication or for other reasons.
3. Communication module: If coroutines need to cooperate with each other, they need to communicate and exchange data. The communication module of a coroutine is called a channel, and a coroutine can send messages to other coroutines with the use of a channel. This paper is mainly to introduce the communication module.

DESIGN OF COMMUNICATION BETWEEN COROUTINES

Data structure of a channel

A channel's data structure is based on a ring buffer. A ring buffer (Zhangdun et al., 2012) is a first-in-first-out data structure that reduces duplicate address operations and increases stability relative to queues (Feldman and Dechev, 2015). Ring buffers are widely used in various fields (Bergauer et al., 1996). It is easy to separate data writing from reading with the use of a ring buffer, which avoids competition between reading threads and writing threads, and reduces the use of locks. I have used ring buffer as channel's infrastructure. The working principle of the ring buffer is shown in Figure 3.

As shown in Figure 3, in a fixed-size buffer, there are two pointers: read and write. When some data is written to the buffer, write increases. When some data is read from the buffer, read increases. Using a ring buffer, the producer-consumer mode can be simply realized. 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, there is no need to lock the buffer, which increases the efficiency of communication. The channel structure including a ring buffer is as follows:

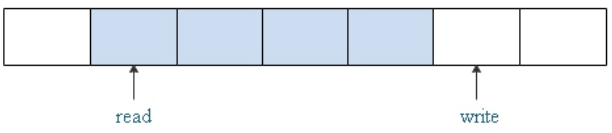

Fig. 3. Ring buffer structure

```
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, 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 that 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 detail in the next section. 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 the channel is stored in the main memory so that both MPE and CPE can access it.

Design of parallel synchronization mechanism

With ring buffer, messages can be delivered safely without a synchronization mechanism in the case of a single producer and a 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, certain measures need to be taken to ensure that the data in the channel is correct (He, 2012). In x86 instruction computers, CAS (compare and swap) atomic operation is often used to deal with multithreading competition (Michael, 2003). In the SW26010 processor, MPE and CPE have different levels of instruction support. CAS operation is supported on MPE, but

not on CPE. First, CAS operation is used to deal with multithreading competition on MPE.

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 it is 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, it is first determined 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 lets 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 the 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 the write location in the buffer has been determined.

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 modify data. Its interface is as follows.

updt(_n_, _addr_)

This operation represents adding _n_ to the data pointed by _addr_. Parallel synchronization of a 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:

```
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' saved in the variable 'temp', and compared with the value of 'to_write'. If they are not equal, it means that another producer has modified the value of 'write'. At this point, the 'write' value should be read again. If they are equal, it means that another producer has finished sending to the channel, then this producer will modify the value of 'write'. Since comparison and update cannot be performed 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 compared with the value saved by the local variable 'temp'. If the current value of variable 'write' equals 'temp'+1, which indicates that only one atomic operation has been performed, the message can be send to the channel in the 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 point, it means that other threads have also made atomic updates, this producer should reduce the value of the variable 'write' by an atomic update, let the 'write' value revert to the state in which it was before this producer accessed. This send can be considered a failure, and it is done again from the beginning. In this way, it is ensured that when multiple producers send data to the channel, at most one producer can find that after an atomic operation, the value of variable 'write' is 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 performance loss is higher than that of MPE III.

Different modes of channels

Although using the synchronization mechanism can ensure the correctness of the messages in the channel, it will also result in decreasing communication efficiency. Therefore, in order to maximize communication efficiency, this paper designs different communication modes for different numbers 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 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 the reading and writing of the buffer need to be synchronized, which is also the default mode of the channel.

Blocking and wakeup mechanism of channel

In the process of communication, sometimes the program wants to communicate but cannot communicate normally, for example, the producer cannot send a message when the channel is full. In such a case, 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, thread switching consumes many system resources, which will lead to 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 the running queue, other coroutines continue running, which reduces the time cost for waiting. When a producer sends messages to the 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 directly 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 is awakened by the 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 returns to the running queue to continue running, and has a high probability to send messages successfully. Similarly, when the channel is empty, the consumer coroutine will also be blocked and awakened by a producer coroutine. This kind of mutual wake-up mechanism allows a program to directly give up the execution in the case of inability to communicate and let other coroutines run instead of waiting in a loop. It also does not consume a lot of system resources as in the case of thread switching, and effectively uses the operation ability of the processor.

EXPERIMENTAL RESULTS AND ANALYSIS

Performance test of channel

In order to understand the specific performance of channel communication, it is necessary to test the operation performance of the channel under different conditions. Since a channel has different modes, which will affect the competition between producers and consumers, the situation was tested with and without competition, on MPE and on CPE. The used message data is an integer, with an average of multiple send times as a result.

Table 1. Channel communication performance with different numbers of producers and consumers

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 loir than that of the main core, and the performance degradation is more serious when the multi-core competes. From the results i 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 loir than that of MPE. First is that the speed of accessing the main memory from CPE is loir 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.

CONCLUSIONS

In this paper, i have designed 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 have designed 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, i 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.

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.

REFERENCES

Bailes, P.A. (1985). A low-cost implementation of coroutines for c. Softw. Pract. Exp., 15(4), 379–395.

Bergauer, H., Jeitler, M., Kulka, Z., Mikulec, I., Neuhofer, G., …, Taurok, A. (1996). A 1-ghz flash-adc module for the tagging system of the cp-violation experiment na48. Nucl. Instrum. Methods Phys. Res., 373(2), 213–222. https://doi.org/10.1016/0168-9002(95)01521-3

Duan, X., Xu, K., Chan, Y., Hundt, C., Schmidt, B., …, Liu, W. (2017). S-Aligner: Ultrascalable Read Mapping on Sunway Taihu Light. IEEE International Conference on Cluster Computing. IEEE.

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) (pp. 615–624). Orlando: IEEE. https://doi.org/10.1109/IPDPS.2017.20

Feldman, S., Dechev, D. (2015). A wait-free multi-producer multi-consumer ring buffer. Acm Sigapp Applied Computing Review, 15(3), 59–71.

Fu, H., Liao, J., Yang, J., Wang, L., Song, Z., …, Yang, G. (2016). The sunway taihulight supercomputer: system and applications. Sci. China Inf. Sci., 59(7), 072001 (2016). https://doi.org/10.1007/s11432-016-5588-7

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

Hui-Ba, L.I., Yu-Xing, P., Xi-Cheng, L.U. (2008). A programming pattern for distributed systems. Computer Engineering & Science.

Michael, M.M. (2003). CAS-Based Lock-Free Algorithm for Shared Deques. Euro-par Parallel Processing, International Euro-par Conference, Klagenfurt, Austria, August. DBLP, 651–660.

Pauli, W., Soffa, M.L. (1980). Coroutine behaviour and implementation. Softw. Practi. Exp., 10(3), 189–204.

Wang, X., Liu, W., Xue, W., Wu, L. (2018). swSpTRSV: a fast sparse triangular solve with sparse level tile layout on sunway architectures. In: PPoPP '18: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming Acm Sigplan Symposium (pp. 338–353). New York: Association for Computing Machinery. https://doi.org/10.1145/3178487.3178513

Xu, X., Li, G. (2012). Research on coroutine-based process interaction simulation mechanism in c++. Asia Sim., 3, 178–187.

Zhangdun, T., Shuyu, C., Yao, L. (2012). Research and implementation of high-performance ring buffer. Computer Engineering, 38(8), 228–231.

KOMUNIKACJA MIĘDZY KORUTYNAMI DLA PROGRAMU RÓWNOLEGŁEGO Z WYKORZYSTANIEM WIELORDZENIOWEGO PROCESORA SW26010

Abstrakt. Komunikacja między programami równoległymi jest nieodzowną częścią obliczeń równoległych. SW26010 to heterogeniczny, wielordzeniowy procesor używany do budowy superkomputera Sunway Taihu Light, który jest dobrze przystosowany do obliczeń równoległych. Aby zaprojektować i wdrożyć korutyny na procesorze SW26010 i poprawić jego współbieżność, bardzo ważne i konieczne jest osiągnięcie komunikacji między korutynami. Dlatego w niniejszej pracy zaproponowano system komunikacyjny do wymiany danych i informacji pomiędzy korutynami na procesorze SW26010, który składa się z następujących części: projektu i implementacji komunikacji kanałowej w trybie producent–konsument na podstawie buforu pierścieniowego oraz mechanizmu synchronizacji dla stanu multiproducent i multikonsument, bazującego na różnych operacjach atomowych na MPE (*management processing element*) i CPE (*computing processing element*) procesora SW26010. Zaprojektowano również mechanizm budzenia pomiędzy producentem i konsumentem, który redukuje czas oczekiwania programu na komunikację. W wyniku przeprowadzonych testów i analizy wydajności kanału przy różnej liczbie producentów i konsumentów, wyciągnięto wniosek, że przy zwiększeniu liczby producentów i konsumentów wydajność kanału będzie spadać.

Słowa kluczowe: korutyny, SW26010, wielordzeniowy, komunikacja równoległa, synchronizacja