Fusion Engineering and Design 157 (2020) 111720

Contents lists available at ScienceDirect

Fusion Engineering and Design

journal homepage: www.elsevier.com/locate/fusengdes

Development of a real-time framework for parallel data stream processing★☆

Giil Kwon*, Jaesic Hong

Control Team, National Fusion Research Institute, Daejeon, South Korea

ARTICLE INFO

Keywords:
Real-time system
Software framework
Real-time network
Data stream processing

ABSTRACT

This paper presents the Korean Superconducting Tokamak Advanced Research (KSTAR) real-time framework for the parallel data stream processing framework (RT-ParaPro). RT-ParaPro is a framework used to develop a program that simultaneously processes data streamed over a real-time network, sends the data over a network, and archives the data in real-time. In most fusion experimental devices, each device processes the data needed for real-time control and transmits them to the other real-time systems in real-time via the network. By using RT-ParaPro, it is possible to simplify the configuration of a program that performs a series of processes and shorten the development time. Unlike other real-time frameworks that focus on real-time control, RT-ParaPro is specialized for the parallel data stream processing and transmission of data over a real-time network. By using this framework, the L-H transition detection system using machine learning (LHML), which determines whether plasma is in low-confinement mode (L-mode) or high-confinement mode (H-mode) in real-time by using machine learning, and the reflective memory (RFM) archiving system, which stores various RFM channel data to MDSplus, has been developed and operated in KSTAR. To evaluate the real-time performance of this framework, we tested the consistency of the thread period by varying the period of the thread. The test results show that the thread control period is consistent. The period of the thread has a jitter of about 8 μs not only at a low control cycle rate (1 kHz) but also at a control cycle rate of 100 kHz.

1. Introduction

To control plasma, a real-time data processing system is necessary, because plasma control system (PCS) requires real-time processed data to predict the state of the plasma in real-time. Recently, many fusion devices have been developed for a real-time diagnostic system and real-time data processing system and applied to infer the plasma state in real-time. Most plasma physics algorithms that are used to analyze and simulate the plasma physics phenomenon have not been applied to plasma real-time control due to their high computational complexity. However, the algorithms can now be applied to plasma control due to increasing computational power and by decreasing the computational complexity of algorithms using machine learning approximation techniques. Korean Superconducting Tokamak Advanced Research (KSTAR), like other fusion devices, is increasingly demanding real-time data stream processing. In KSTAR, 50 real-time systems are connected to the real-time network (such as International Thermonuclear Experimental Reactor (ITER), Synchronous Databus Network (SDN), Reflective Memory (RFM)) to process data in real-time. These real-time systems process the data transmitted from multiple servers in parallel and send the processed data to other systems in real-time. In KSTAR, the RFM archiving system that has the highest number of data channels must simultaneously process data stream from 158 different channels in parallel in a 1 kHz period [1]. To address this requirement and to standardize the real-time application at KSTAR, KSTAR has developed RT-ParaPro, which is specialized for the parallel data stream processing, archiving and transmission of data over a real-time network. The real-time framework for parallel data stream processing framework (RT-ParaPro) is designed to have the following attribute:

1 Good performance: Most KSTAR real-time applications, except the KSTAR PCS application, operate at a 1 kHz control loop frequency. To operate in synchronization with the KSTAR real-time applications, a real-time system operating in KSTAR must have a control cycle of 1 kHz or more with low jitter.
2 Fault-tolerance: KSTAR real-time applications must run without faults and automatically recover in the event of a fault, effectively protecting the equipment. Real-time monitoring or control is

☆ This work was and performed within the cooperation defined in the Memorandum of Understanding (MoU) between ITER International Organization (IO) and NFRI and was partially supported by the Korean Ministry of Science ICT & Future Planning under the KSTAR project.
* Corresponding author.

https://doi.org/10.1016/j.fusengdes.2020.111720
Received 8 August 2019; Received in revised form 3 February 2020; Accepted 17 April 2020

necessary to protect the equipment during the KSTAR experiment. Finite state machine (FSM) of RT-ParaPro manages the thread life cycle. Every experiment thread (including fault threads) are terminated and generated at the initial stage of the experiment according to commands from the thread life cycle FSM. Thus, this makes the application fault-tolerant.

3 Administration: All the parameters of the real-time applications must be set and get via an Experimental Physics and Industrial Control System (EPICS) Channel Access (CA). Because all KSTAR control devices are controlled by the CA of EPICS. The user can control the system, which uses RT-ParaPro remotely by using EPICS CA. The information of the system is provided in the EPICS process variable (PV) form, so the user can monitor the status of the system by using the GUI-based Operator Interface (OPI).

4 Automation: All of the KSTAR applications should be operated automatically according to the shot sequence of KSTAR. The system using RT-ParaPro synchronizes with the KSTAR Central Control System (CCS) and operates automatically according to the shot sequence.

2. Related works

There are real-time frameworks used for fusion experiments (such as ITER Real-Time Framework (RTF) [2] and Multi-threaded Application Real-Time executor (MARTe) [3]). RT-ParaPro is similar to those real-time frameworks. RT-ParaPro has a modular structure like ITER RTF and MARTe. Due to this structure, only the parts that are dependent on the environment can be changed according to the environment, and the parts that are not dependent on the environment can be reused. RT-ParaPro is designed to develop applications that process data for the KSTAR environment. The framework support RFM and ITER SDN, which KSTAR uses as a real-time network. It also supports interfaces for interworking with other KSTAR control systems. The framework also supports the EPICS and MDSplus modules used by KSTAR. There are also data stream processing frameworks (such as Apache Spark [4], Apache Flink [5], and Apache Heron [6]). These data stream processing frameworks are real-time distributed data stream processing system. They are specialized for distributed processing data stream. The main difference between these data stream processing frameworks and RT-ParaPro is a real-time data processing capability. These data stream processing frameworks take several milliseconds (ms) to process each data stream. RT-ParaPro supports sub-microsecond (μs) latency.

3. Real time parallel processing (RT-ParaPro)

3.1. Software architecture

Fig. 1 presents the software architecture of RT-ParaPro. The RT-ParaPro framework is based on the CentOS 7 operating system. A Messaging Real-time Grid (MRG)-Realtime kernel was used to achieve real-time performance. The server using RT-ParaPro is synchronized with the KSTAR Timing system using the ITER Time Communication Network (TCN) [2], which is the dedicated network for absolute synchronization. EPICS was used to implement an interface that can control the RT-ParaPro system. The data generated by RT-ParaPro can be saved to the MDSPlus server. For this purpose, RT-ParaPro provides the MDSplus interface. RT-ParaPro provides the real-time thread configuration API. By using the API, the user can easily set the real-time thread attributes (such as thread period, thread policy, and thread CPU affinity). KSTAR uses RFM and the ITER SDN [2] as a real-time network. The ITER SDN is the real-time network based on a UDP multicast over a 10-GbE cut-through packet-switching infrastructure. Two real-time network interfaces are provided that the program developed using RT-ParaPro can send and receive data in real-time using this interface.

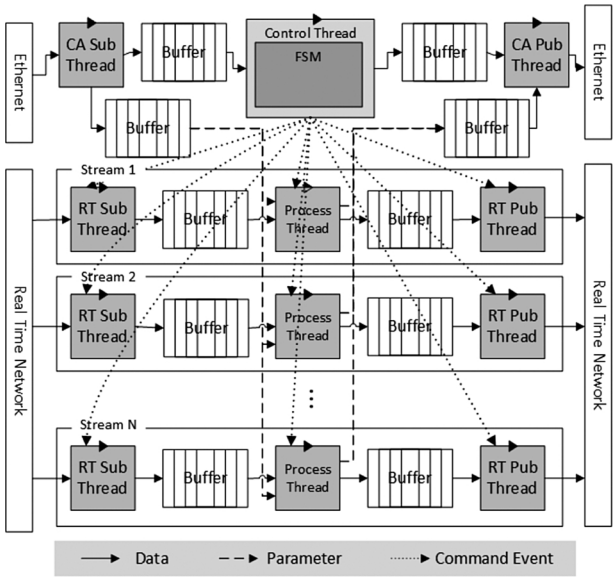

Fig. 1. Software architecture of RT-ParaPro. RT-ParaPro uses MRG-R Kernel to achieve real-time processability. ITER SDN, ITER TCN, MDSPlus, EPICS library are used.

3.2. Real time data stream processor unit

This framework consists of pairs of threads and buffers, single threads and many other components. Among them, the thread and buffer pairs that implement parallel producer/ consumer design pattern are the core components of the framework. These pairs are real-time data stream processing units. Fig. 2 shows multiple real-time data stream processing units. Each unit receives data stream from real-time networks. The unit processes one topic data stream, and sends the result through a real-time network. Each real-time data stream unit can have a different thread period. Often, the system that sends data has different data sending rates. By using this framework, we can adopt various re-sampling techniques to match the data rates of two different systems. Resampling is a method of taking a value representing a part of a data section according to a target low rate at a high rate data. There are various ways to take this representative value, such as taking the first value of some intervals, taking the average value of the interval data as a representative value, taking the intermediate value and many other

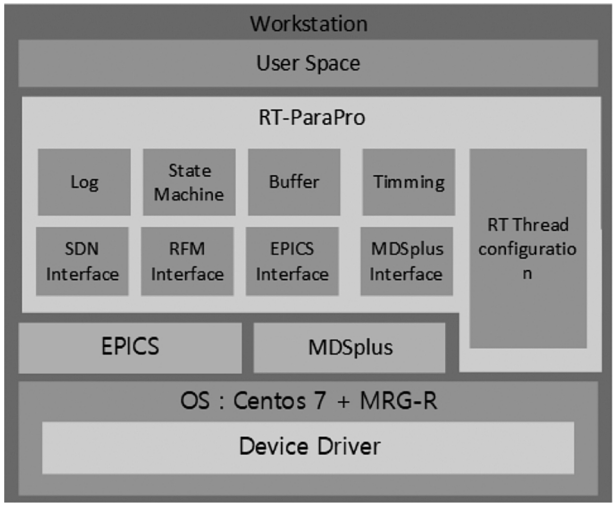

Fig. 2. Structure of RT-ParaPro. RT-ParaPro is consists of a pair of real-time threads and thread-safe buffers. A control thread controls each thread using events. Users can control the 'control thread' via EPICS Channel Access.

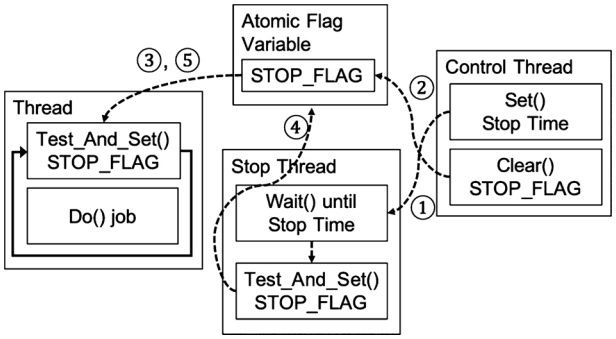

Fig. 3. Thread synchronization procedure: (1) The control thread sets stop time. (2) The control thread clears STOP_FLAG to start the thread. (3) The thread checks the STOP_FLAG every time. If STOP_FLAG is set before the call, the loop runs; otherwise the loop exits. (4) The stop thread waits until the previously sets time and set the STOP_FLAG. (5) The loop ends because the stop flag is set.

methods are used for resampling data in this framework. Among them, we applied the LHML system to take the first value as a representative value of the data of a certain section and to duplicate it at a low data rate.

3.3. Thread synchronization

Threads in the framework are implemented to operate in synchronization with the commands from the control thread. To minimize performance degradation, we implement the lock-free thread synchronization code using atomic operations. When threads share the resource, they use locks (such as mutex, and spinlock) or use critical sections to avoid the race condition. However, this approach causes performance degradation. Atomic operations reduce the impact of performance degradation caused by locks. An atomic operation is an operation that ensures that it is not interrupted by other threads during the operation. This feature allows avoiding the race conditions without using a lock. Events used in the control thread were developed using this atomic operation to control other threads. Fig. 3 presents the thread synchronization procedure. In STOP_FLAG is the std::atomic_flag type variable, which is guaranteed to be lock-free. The control thread starts the RT-ParaPro's thread at the start time by clearing the STOP_FLAG using the atomic flag function (clear()). The thread in RT-ParaPro checks the STOP_FLAG every time by using the atomic flag function (test_and_set()). If STOP_FLAG is set before the call, the loop runs otherwise the loop exits. The stop thread waits until the previously set time and stops the RT-ParaPro's thread by set STOP_FLAG. The buffer used in the consumer, producer model includes spinlock implemented with atomic instructions (_sync_val_compare_and_swap) to prevent multiple threads from concurrently accessing the buffer.

3.4. Thread life cycle management

Fig. 4 shows the FSM of the control thread that manages the thread lifecycle. As Fig. 1 shows, the control thread controls each thread, and it operates according to the FSM. The core FSM defines the default task of the thread. Users can use additional FSMs to define other operations, such as post-processing jobs, other than the default operations. These additional FSMs are called overlay FSMs. By using an overlay FSM in an overlapped manner, a user can disjoint additional tasks from a default task. It is possible to create an additional tasks by overlaying them on existing tasks regardless of the existing task. This allows complex tasks to be implemented by simply nesting simple FSMs on top of the core FSM. At each point in time, FSMs send an event to each thread to start and stop the thread. The core FSM has 5 states (IDLE, INIT, CONFIG, RUNNING, and OFF) and 5 events (INIT, CONFIG, START, STOP, and EXIT). For an overlay FSM, users can define the dedicated state and

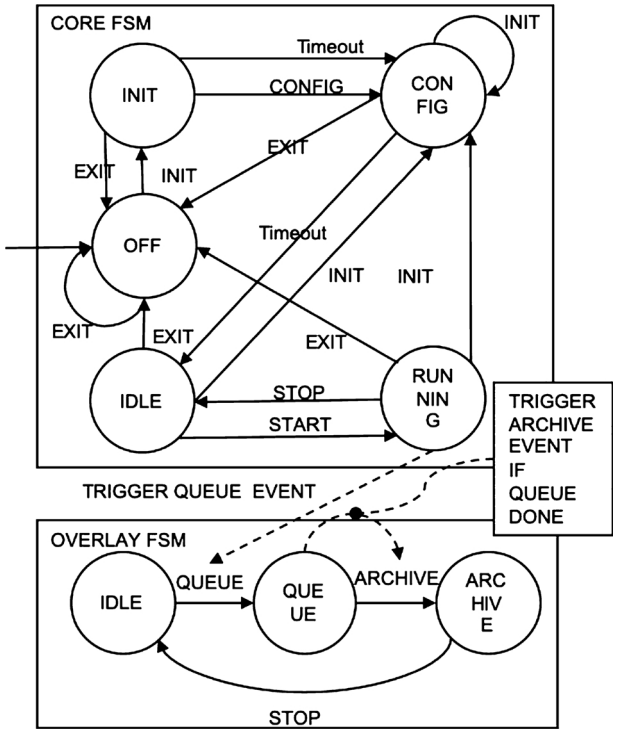

Fig. 4. Finite state machine (FSM) of the control thread which manages the thread lifecycle. The FSM of RT-ParaPro is consists of the core FSM, which defines the default task of the thread, and the overlay FSM, which defines the additional operation.

events. Fig. 4 presents the FSM of the RFM archiving system [1]. In the RFM archiving system case, the default task is to get the data from RFM, and the additional task is to store the data in the MDSplus server. The system starts in the 'OFF' state. When the core FSM receives an 'INIT' event, the state of the core FSM goes into the 'INIT' state. In the 'INIT' state, the FSM sets the initial parameters. Actions performed in the 'INIT' state must be performed only once after starting the program. When the FSM receives a 'CONFIG' event, the state goes into the 'CONFIG' state. In the 'CONFIG' state, the FSM sets parameters that must be set for each experiment. The FSM deletes all the threads (including fault threads) and buffers and creates new threads and buffers. Thus even if the thread seems to be malfunctioning, all the threads will be recreated in the 'CONFIG' state and the system will run robustly in the next experiment without any problems. After a predetermined time in the 'CONFIG' state, the FSM is transferred to the 'IDLE' state by a timeout event. When the 'START' event is received in the 'IDLE' state, the state changes to the 'RUNNING' state. At this moment, the core FSM sends 'RUN' events to the RFM read thread to acquire and process the data in parallel. When a 'STOP' event is received in the 'RUNNING' state, it returns to the 'IDLE' state. After an 'INIT' event is received in 'IDLE' state, the core FSM goes to the 'CONFIG' state and sets the system parameter value. After a certain time, it returns to the 'IDLE' state. When the state of the core FSM is changed to the 'IDLE' state from the 'RUNNING' state by a 'STOP' event, the state of the overlay FSM transitions from the 'IDLE' state to the 'QUEUE' state. At this moment, data parser threads receive 'RUN' events from the overlay FSM. The data parser thread gets the data from data buffers and parses the data according to the MDSplus tag and sends it to parser unit's buffers. Each time a task is finished, the thread sends an event to the overlay FSM indicating that the task is finished. The overlay FSM receives it, and when all the threads are done, it sends an 'ARCHIVING' event to itself and goes to the 'ARCHIVE' state. In the 'ARCHIVING' state, the overlay FSM sends 'STOP' events to the data parser threads and send 'RUN' events to the 'MDSplus write thread' to get the data from the buffer and

send data into MDSplus server. Since the data does not need to be processed quickly after the experiment, the data is taken out of the buffers sequentially in batch style and stored in the MDSplus server. When the 'STOP' event occurs on overlay FSM, the state goes back to the 'IDLE' state.

3.5. System control using EPICS Channel Access

To reduce the dependency on the application by the EPICS IOC and simplify the application, the framework does not include the EPICS Input/Output Controller (IOC) in the framework. Instead of running IOC inside the framework, the RT-ParaPro application has the interface for EPICS CA to communicate with the EPICS IOC. All parameters in the RT-ParaPro application are mapped to EPICS PVs. Therefore, all parameters can be obtained or set through the EPICS CA. As seen in Fig. 1, RT-ParaPro has a dedicated thread that gets/puts the information from the CA and buffers it to get/put the data asynchronously through the CA without compromising the performance of the real-time thread. The user can control the system remotely by using EPICS CA. The information of the system is provided in an EPICS PV form, so the user can monitor the status of the system by using a GUI-based operator interface (OPI). The OPI was implemented by using Control System Studio (CSS).RT-ParaPro is automatically executed following the KSTAR shot sequence from KSTAR CCS with EPICS CA.

4. Applications

By using RT-ParaPro, two real-time applications are developed in KSTAR.

4.1. RFM archiving system (RFMARC)

Fig. 5 shows an overview of the RFM archiver system [1]. This application, which stores 158 RFM channel data to MDSplus, was developed and operated in KSTAR (this application runs at 1 kHz). The RFM archiver was developed in 2017 in KTSAR and has been modified and redesigned to achieve improved performance in 2019. Fig. 6 represents the structure of the RFM archiver system. This system consists of 318 data stream processing units (there are 158 topics, 2 connected units used for 1 topic, and 2 units used for EPICS CA). The RFM read unit, which takes data from the RFM and puts it in the buffer, and the parser unit, which takes data from the RFM read unit buffer, parses the data according to the MDSplus tag, and puts the data in the buffer of each parser unit, are connected in sequence. The MDSplus write unit writes thread reads from the parser unit's buffer and sends data sequentially to the MDSplus server.

4.2. L-H mode detection system using machine learning (LHML)

Fig. 7 presents an overview of the L-H mode detection system. This application determines whether plasma is in a low-confinement mode

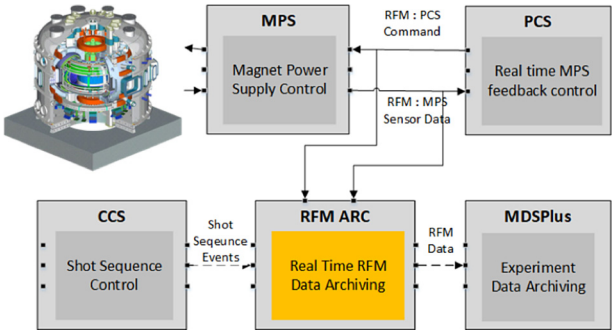

Fig. 5. Overview of the RFM archiver system.

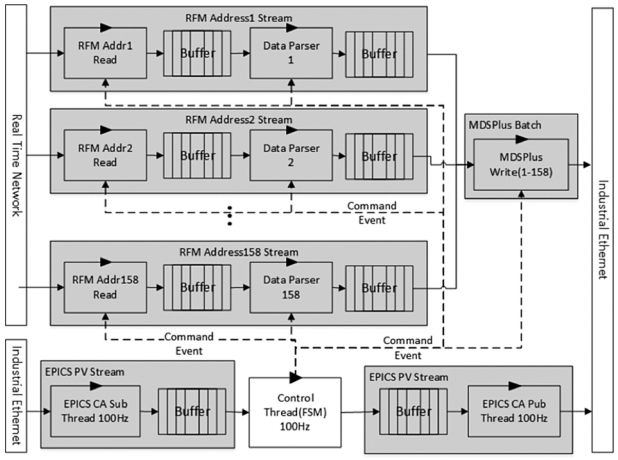

Fig. 6. The software structure of the RFM archiver system. RFM archiver system consists of 318 data stream processing units. (There are 158 topics, 2 connected units used for 1 topic and 2 units used for EPICS CA.)

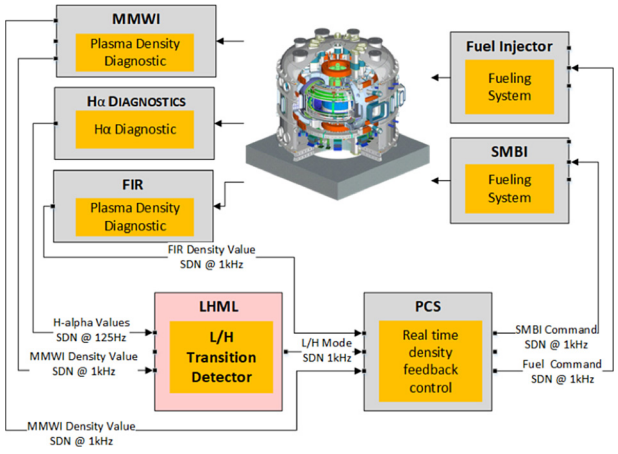

Fig. 7. Overview of the L-M mode detection system.

(L-mode) or high-confinement mode (H-mode) in real-time by using machine learning (Long Short-Term Memory, LSTM). The estimated L-H mode from the LHML system is sent to the PCS. The PCS determines the control algorithm of the fuel injector system according to the L-H mode from the LHML system. Fig. 8 shows the L-H mode detection system consists of 6 real-time data stream processing units (2 connected units used for H-alpha topic, 1 unit for MMWI topic, 1 unit for L-H mode

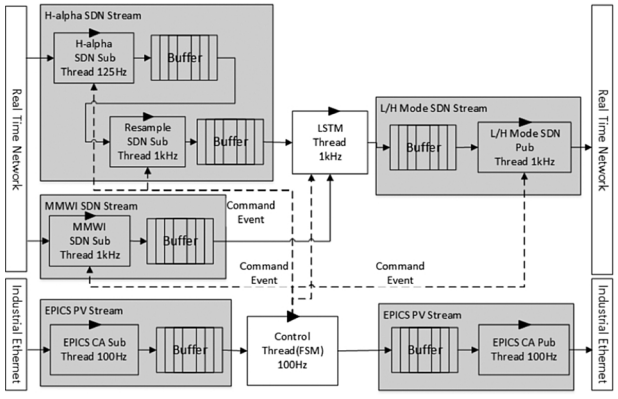

Fig. 8. Software structure of the L-M mode detection system (2 connected units used for H-alpha topic, 1 unit for MMWI topic, 1 unit for L-H Mode topic and 2 units for EPICS CA).

topic and 2 units for EPICS CA). To estimate the L-H mode, the system receives a $H_\alpha$signal and Milli-Meter Wave Interferometer (MMWI) signal from other diagnostic systems. Here, the H_α SDN real-time data stream processing unit, MMWI SDN real-time data stream processing unit, and L-H Mode SDN real-time data stream processing unit are send/receive SDN topic data in a fixed cycle. H_α SDN real-time data stream processing unit is running at 125 Hz. The MMWI SDN real-time data stream processing unit is running at 1 kHz. The system that sends H_α diagnostic data has different data sending rates. The framework uses resampling techniques to match the data rates of two different systems. In this case, the resampling thread duplicates the first elements of buffer 8 times to match the data rate with those of other real-time data stream processing units (1 kHz). It is better to match the system run cycle to the lowest system cycle, but this application has a target run cycle designed at 1 kHz. The LSTM thread runs a deep learning algorithm (LSTM) to detect L-H transition in the incoming data sequentially. LSTM module is implemented in C++ to run in real-time. LSTM module takes 56 μs to estimate the L-H mode.

5. Performance test

The real-time performance of a real-time thread can be determined by measuring whether the period of the thread is constant. If a thread's period is constant, it can be seen that the thread works deterministically. To calculate the thread's period, we store the time at which the thread starts, save time (nano-second order) at the next start, and calculate the difference between these two times. To evaluate the real-time performance of this framework, we tested the consistency of the thread period by varying the frequency of the thread (1 kHz, 2 kHz, 5 kHz, 10 kHz, and 100 kHz). The test results show that the thread control period is consistent. Fig. 9 shows a plot of the thread cycle measurement for 100 s. It shows that the measured thread period value according to all thread frequency setting values (1 kHz, 2 kHz, 5 kHz, 10 kHz, and 100 kHz) is constant without sudden change for 100 s. The majority of real-time data processing applications in KSTAR set up a 1 kHz thread frequency setting. Fig. 10 plots the consistency of the real-time thread cycle at a 1 kHz thread frequency in more detail. The period of a real-time thread with a frequency of 1 kHz is constant in the upper graph of Fig. 10 without any sudden change in spikes. The lower graph of Fig. 10 represents the histogram of the thread period and Gaussian curve fitting result. The thread period data has a Gaussian distribution. There are data of noise components, but the number is too small to plot on the graph. The data of this noise component also have a small Gaussian distribution. Table 1 shows the results of the thread of period consistency test results. The mean of the measured thread period is approximately equal to the set thread duration. The period of the thread has a jitter of about 8 μs not only at a the low control cycle rate (1 kHz) but also at a the control cycle rate of 100 kHz. The experiment results show that the real-time thread of the real-time application created with RT-ParaPro is constant.

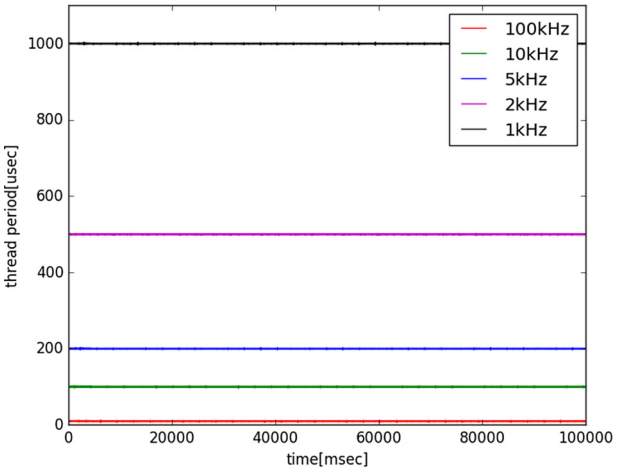

Fig. 9. Consistency of RT thread period at various threads frequency (1 kHz, 2 kHz, 5 kHz, 10 kHz, and 100 kHz).

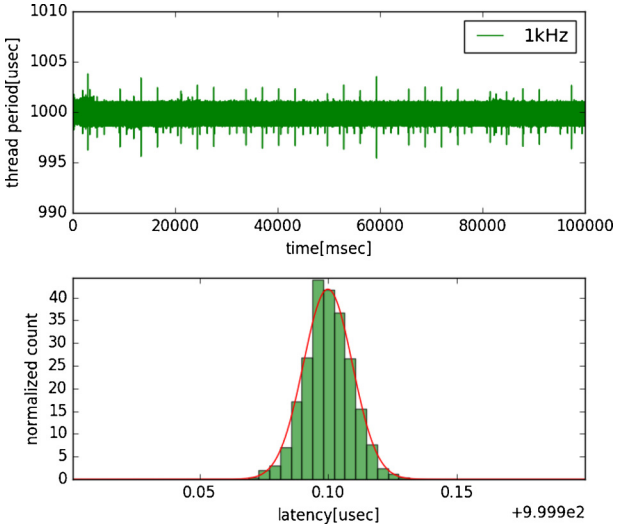

Fig. 10. Upper graph: Thread period graph, there is no abrupt latency spike exists in the thread period plot. Lower graph: The histogram of the thread period and Gaussian curve fitting results, thread period has a Gaussian distribution.

Table 1
Consistency of real-time thread period.

Thread (Hz)	Period (ns)	Mean (ns)	Std (ns)	Max-Min (ns)
1 kHz	1,000,000	999,999.99	176.94	8369.00
2 kHz	500,000	499,999.99	171.88	6654.00
5 kHz	200,000	199,999.99	169.11	8528.00
10 kHz	100,000	99,999.99	171.58	7620.00
100 kHz	10,000	9999.99	164.99	7325.00

6. Conclusion

In this paper, we presented a Real-Time framework for the Parallel data stream Processing framework (RT-ParaPro). RT-ParaPro is a framework used to develop a program that simultaneously processes data stream transmitted over a real-time network and sends data over the network in real-time. This framework is specialized for parallel data stream processing and transmission of data over a real-time network. In this framework, a processor for processing data stream is configured in parallel, and a program advantageous for parallel processing can be created. By using this framework, KSTAR developed two real-time data processing application which needs to have microsecond (μs) order latency. To support deep learning that requires more computation, there is a plan to develop General-Purpose computing on Graphics Processing Units (GPGPU) computing processing modules for this framework.

Author contributions

Giil Kwon: Conceptualization, methodology, software, validation, formal analysis, investigation, resources, data curation, writing – original draft, visualization, investigation, writing – reviewing and editing.

Jaesic Hong: Writing – reviewing and editing, supervision, project administration, funding acquisition.

Conflict of interest

The authors declare that there is no conflict of interest.

References

[1] G. Kwon, W. Lee, T. Lee, J. Hong, Development of a real-time data archive system for a KSTAR real-time network, Fusion Eng. Des. 127 (2018) 202–206.
[2] A. Winter, P. Makijarvi, S. Simrock, J. Snipes, A. Wallander, L. Zabeo, Towards the conceptual design of the ITER real-time plasma control system, Fusion Eng. Des. 89 (2014) 267–272.
[3] A.C. Neto, F. Sartori, F. Piccolo, R. Vitelli, G. De Tommasi, L. Zabeo, A. Barbalace, H. Fernandes, D.F. Valcárcel, A.J. Batista, MARTe: a multiplatform real-time framework, IEEE Trans. Nucl. Sci. 57 (2010) 479–486.
[4] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, et al., Apache spark: a unified engine for big data processing, Commun. ACM 59 (2016) 56–65.
[5] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, K. Tzoumas, Apache flink: stream and batch processing in a single engine, Bull. IEEE Comput. Soc. Tech. Comm. Data Eng. 36 (2015).
[6] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J.M. Patel, K. Ramasamy, S. Taneja, Twitter heron: Stream processing at scale, Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, SIGMOD'15, Association for Computing Machinery, New York, NY, USA, 2015, pp. 239–250.