FUSION SCIENCE AND TECHNOLOGY · VOLUME 75 · 835–840 · NOVEMBER 2019

DOI: https://doi.org/10.1080/15361055.2019.1658037

Development of Real-Time Software for Thomson Scattering Analysis at NSTX-U

Roman Rozenblat,[a]* Egemen Kolemen,[b] Florian M. Laggner,[b] Christopher Freeman,[a] Greg Tchilinguirian,[a] Paul Sichta,[a] and Gretchen Zimmer[a]

[a]*Princeton Plasma Physics Laboratory, Princeton, New Jersey*
[b]*Princeton University, Princeton, New Jersey*

Received May 31, 2018
Accepted for Publication August 16, 2019

Abstract — *The Thomson scattering (TS) diagnostic on the National Spherical Tokamak eXperiment Upgrade (NSTX-U) has been an essential system for many operational campaigns due to its function of measuring plasma electron density and temperature. Constructive feedback to improve the next plasma discharge, however, has been limited because of in-between shots analysis. Plasma control, therefore, desires a diagnostic system that is real-time capable. This contribution presents the development of software that demonstrates the feasibility of a real-time TS diagnostic system for NSTX-U. The developed software is able to evaluate the electron temperature and density within 2.5 ms.*

The overall system requirement is specified by a 60-Hz timing cycle, which is driven by the TS laser pulse rate. The real-time software processes the peak amplitudes of the detected photons, evaluates the electron temperature and density, and then outputs them to an analog output card that is used to interface with the NSTX-U control. The real-time software is implemented in an object-oriented architecture using C++11. C++11 software components include Abstract class, Atomic data types for synchronization, and a Hash data structure. The software application makes use of multiple threads that run concurrently: a thread to acquire the photon peak amplitude and feed a circular buffer, threads to evaluate the electron density and temperatures, and a thread that supplies corresponding output voltages and feeds the output card.

In summary, the new real-time TS system has been proven to meet the 60-Hz system requirement. For this reason, the software implementation was deemed successful. In future NSTX-U campaigns, this diagnostic will be a great asset enabling real-time plasma density and temperature control.

Keywords — *Thomson scattering, real time, NSTX-U, multiple threads, object-oriented architecture.*

Note — *Some figures may be in color only in the electronic version.*

I. INTRODUCTION TO THE ANALYSIS WORKFLOW AND THE UTILIZED HARDWARE

I.A. Background and Motivation

Thomson scattering (TS) is a known and widely applied tool used to diagnose hot plasma. Its underlying fundamental principle is the interaction of a high-energy laser beam with free electrons of the plasma. Especially in applications for magnetic confinement fusion, it is one of the main profile diagnostic tools since it provides a noninvasive, localized measurement of electron density and temperature.[1] For this reason TS diagnostics are very common and installed on multiple fusion devices. Some of the experiments that use the TS diagnostics method include the Joint European Torus experiment,[2] TFTR (Ref. 3), DIII-D (Ref. 4), Alcator C-Mod tokamak,[5] ADEX Upgrade,[6] MAST (Ref. 7), Large Helical Device[8] (LHD), Helical Symmetry Experiment[9] (HSX), and Wendelstein 7-X (Ref. 10). The goal of the presented project is to build a real-time analysis prototype for the

*E-mail: rrozenbl@pppl.gov

TS diagnostic tool at National Spherical Tokamak eXperiment Upgrade (NSTX-U) that can calculate plasma parameters. These parameters can then be used by the NSTX-U plasma control system (PCS) to feedback the plasma density and temperature during a shot. This diagnostic tool consists of several lasers, which fire in a predefined sequence.[11–14] In addition, this project demonstrates the real-time performance that can be achieved using present-day commercial off-the-shelf hardware and software technologies.

I.B. System Setup and Utilized Hardware Components

The real-time hardware can acquire data of eight polychromators, i.e., eight spatial channels. Each polychromator is equipped with up to six spectral filters and associated avalanche photodiodes (APDs) to measure the spectrum of the scattered light. The signals of the APD detectors are digitized with an SIS3316-250–14 analog-to-digital converter (ADC) that operates at 250 Msamples s^{-1} with 14-bit resolution.[15] The ADC is connected to a server via a Solarflare[16] Ethernet card. There are four SIS3316 cards with 16 channels each, which allows for the digitization of the eight spatial channels and other required signals for the TS analysis. The cards continuously digitize to a circular buffer, whose readout is triggered by laser pulses. When the cards are triggered, the peak values of the detected pulse are extracted in the first step of the real-time Multi-Pulse Thomson Scattering (MPTS) analysis software. The real-time server that is used to process the TS calculation is a ServeDirect server with 32 Gbytes Memory, Intel Xeon 2.2 GHz with 20 cores. The output of the real-time MPTS server is done through an analog output (AO) 16AO64 PCIE card, which has up to 64 channels with 16-bit resolution.[17]

I.C. Layout of the Real-Time Analysis Software

The Real-Time Thomson software consists of (1) the Read-Card thread, (2) Buffer-Read threads, (3) an AO thread, (4) a thread that monitors the trigger event coming from the digitizer card, and (5) the Laser thread. The Read-Card thread reads an ADC channel that indicates the laser energy and four ADC channels that indicate which laser of the TS diagnostic was fired. It also reads the data from each ADC channel on the card. This thread stores the values in several circular buffers. There are seven Read-Buffer threads running in parallel that are assigned to read and process one specific spatial channel consisting of up to six spectral channels. The Read-Buffer thread retrieves the data from the circular buffer and finds the signal's peak, i.e., the maximum amplitude of the detected laser pulse. Furthermore, the Read-Buffer thread computes plasma electron temperature T_e and electron density n_e as well as their errors ($T_{e,error}$ and $n_{e,error}$). To calculate n_e, the energy of the fired laser is required. This is provided by the Laser thread. This thread reads the energy of the laser from a circular buffer. Furthermore, the Laser thread then finds which laser fired by reading from four circular buffers.

The AO thread outputs a two-dimensional vector consisting of eight values, according to the eight analyzed spatial channels, when all the threads finish computing the plasma parameters. For each spatial channel, it outputs four computed plasma parameters to the AO card. The analog card must be calibrated, and the above values need to be adjusted, with the preset value, which is stored in an MDSplus tree.[18] The program then ends after a preset number of samples. Afterward, all raw data, i.e., the peak values, as well as the computed plasma parameters are archived to the MDSplus tree.

II. DATA STRUCTURE/IMPLEMENTATION OF CIRCULAR BUFFERS

The C++11 unordered map structure maps a physical card and channel to a spatial channel and spectral channel. This map structure is filled during the program initialization by grabbing the required information from the MDSplus tree. The data structure used for storing the digitized data is a circular buffer. The advantage of this kind of data structure is that the elements are pushed and read from the top of the vector in constant time.

There are two pointers to access the circular vector: One points to the head, and the other points to the tail. As new data are pushed into the array, the tail pointer is incremented, and modulo operation occurs with a capacity variable. This ensures that the tail pointer will reset to the beginning of the vector after the vector capacity is reached. The tail and the head pointer are atomic C++11 types. The atomic type in C++11 enables multiple threads to access the variable without causing race conditions, and the resulting behavior is well defined. To enable more efficient use of atomic variables without the need for locking, memory order release is used to store the atomic head and atomic tail. This prevents the compiler from changing the order of read and write operation after the store operation. For the same reason as stated above, memory order acquire is used for load operation to prevent reordering of write and read instructions before the load

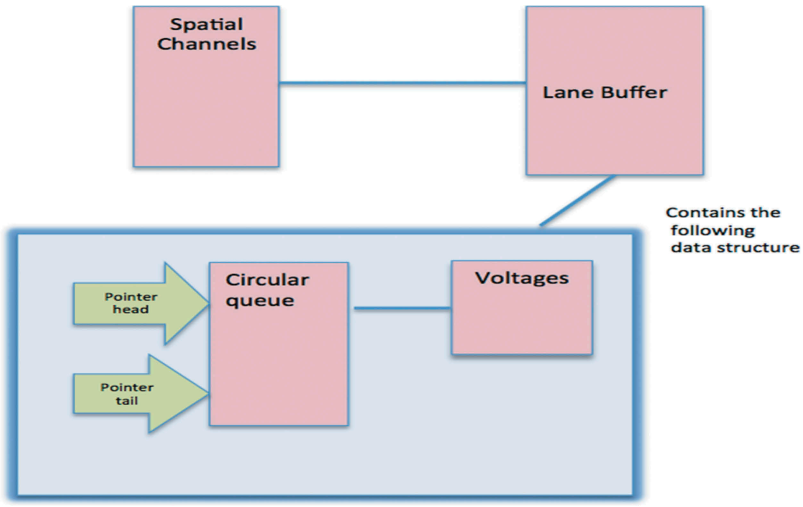

Fig. 1. The circular buffer consists of a circular queue that has a pointer that points to the head and a pointer that points to the tail of the queue. The circular queue contains multiple voltages.

operation. The vector is empty when the two pointers are equal. When the programs start, both pointers are set to the start of the vector. The memory for the vectors is allocated when the programs start. This is done to minimize memory allocation during the program run.

The data structure of a circular buffer is presented in Fig. 1. In Fig. 1, "Spatial Channels" is a C++11 two-dimensional vector, and the "Lane Buffer" is a C++11 class. In addition, the "Circular queue" is a two-dimensional C++11 vector that contains "Voltages," which is a C++11 vector of doubles.

III. DATA PROCESSING WORKFLOW

III.A. Initialization

The program reads the MDSplus tree to initialize the unordered map data structure that maps physical cards and channels to the spatial channels and polychromators. It also reads the calibration values for all the cards and the values used to initialize the cards. The program then reads 20 samples from digitizer cards and computes the average signal baseline and standard deviation. These values will be subtracted from the peak values that are read from the digitizer cards.

III.B. Main Program Execution

The real-time MPTS architecture is described in Fig. 2. The programs consist of ten parallel threads running in tandem. The Card-Reader thread waits for an atomic variable to be incremented by the Trigger thread to tell it that a laser incident occurred. After that, it first reads the energy of the laser and then reads four channels to determine which laser fired. It pushes this data information to an energy buffer and laser-fired buffers. Afterward, the thread reads all physical cards and channels and stores these peaks in circular buffers by using the unordered-mapped data structures that map the spatial channels and polychromator to the physical card and channel.

The Trigger thread waits for the SIS3316 digitizer acquisition control status register to indicate that the data acquisition is completed and then increments an atomic integer variable to indicate that the card was triggered. Each Buffer-Read thread is assigned a unique index number. The Buffer-Reader threads wait for the SIS3316 digitizer to be triggered by monitoring for an atomic integer that is constantly incremented by the Trigger thread. If this is greater than the last time the trigger happened, the thread continues to read the peaks from the circular buffer and reads only the spatial channel that was assigned to it. The thread loops through all the spectral channels for the corresponding spatial channels that are stored in the unordered map and then stores them in a vector of six doubles. After all six peaks are extracted, a function calculates the density and temperature of the plasmas.

The plasma parameter calculation function first determines if the peak can be used for calculation. For example, if the channel is saturated or the channel is inactive, the peak will be marked as bad. Each peak is then adjusted by subtracting the average baseline preshot voltage, and the adjusted peak values are passed to

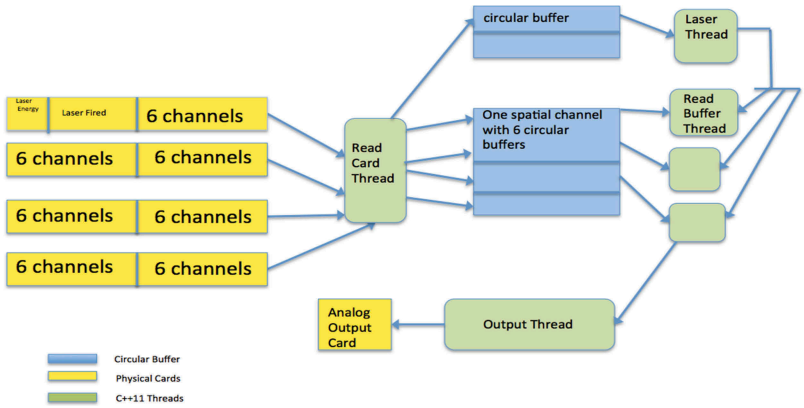

Fig. 2. The real-time MPTS Architecture contains a Reader Card thread that reads channels from the analog-to-digital cards and inserts the voltages into the spatial channels. In addition, it contains a Laser thread and multiple Read Buffer thread that read the voltages from the spatial channel. Furthermore, it contains one Output thread that converts temperature and density parameters produced by the Read Buffer threads to analog voltages that are outputted to the AO card.

a function, which fits the Thomson spectrum. This function attempts an iterative Thomson spectrum fitting on the inputted peaks. If it does not converge after a preset amount of time, it will exit the function.

The AO thread waits for all eight threads to complete or exit the iterative calculations by waiting for the atomic variable increment of each thread to reach 8. The thread then searches for the corresponding output voltage for each calculated plasma parameter in a preset lookup table. The thread uses a binary search[19] algorithm to find a corresponding voltage for each plasma parameter and then outputs the converted voltage of the calculated density, temperature, density error, and temperature error for each of the eight threads to the analog card. The real-time analysis function that produces the above density, temperature, temperature error, and density error was tested extensively and benchmarked. First, a previous shot was inputted into the postshot analysis code, and the output data were recorded. Then, the same shot was inputted into the real-time analysis code, and the output data were recorded as well. The postshot output data were compared to the real-time output data. The postshot and real-time analysis are in good agreement, and the small scatter might occur due to the different numerical implementations.[20] Nevertheless, these differences do not cause any systematic differences in the outputted T_e and n_e (Ref. 20).

III.C. Archiving

At the end of a plasma discharge, i.e., when the preset numbers of samples are fulfilled, the peak values from the archive vectors are stored in the MDSplus tree using segmented records. When the peaks are read from the digitizer card, a time stamp, read from the local server clock, is stored in a vector. In addition, the values of the calculated plasma parameters are stored in a vector during the program run. At the end of the plasma discharge, these values are also stored in the MDSplus tree together with the corresponding time stamps.

IV. SOFTWARE OPTIMIZATION

Vector references are passed into functions; this is done to avoid copying the entire vector structure into the function's working stack. All large array and vectors are allocated memory during the program initialization; therefore, each function and thread can access and store information in the vectors without allocating new memory. The unordered map data structure is used to access the corresponding values for each key in constant time. A binary search algorithm is used to efficiently search the corresponding voltage in a lookup vector. Since accessing the MDSplus tree is slow, all initialization parameters that are stored in the MDSplus tree are read and stored into vectors and unordered maps during program initialization. The program is compiled with the GNU Compiler Collection (GCC) compiler[21] option of o3. With this option, the compiler optimizes the code for speed of execution. The program spends most of its time fitting the Thomson spectrum, i.e., calculating n_e, T_e, n_e error, and T_e error. We were able to reliably reduce the function execution time to 2.5 ms.

The threads are synchronized using C++11 atomic types. This atomic type avoids the data race condition; therefore, the operating system does not need to perform

a context switch on a waiting thread that needs to run. Each thread is scheduled to run on its own core, and the threads run in first-in, first-out (FIFO) schedule.[22]

Only after the shot ends will the program archive the data to the MDSplus tree. During the run, the program stores the raw data and computed values to preallocated vectors in memory. To test a case scenario, we let the operating system schedule the threads and assigned them to different cores. The program was able to meet a 60-Hz deadline execution by using the above schedule technique. The optimization was not pushed to its maximum level. For this reason, it should be possible to optimize more and increase the execution rate.

V. SUMMARY

This project demonstrated the implementation and successful testing of an easily scalable prototype real-time TS analysis system. A big advantage of this system setup is that it works independently of the NSTX-U PCS. There is no direct interaction between the real-time MPTS server and the PCS servers, which would require clock synchronization.[20]

The Thomson spectrum fitting function was able to compute the plasma parameters, namely, T_e, n_e, T_e error, and n_e error, in 2.5 ms. Furthermore, this real-time framework (hardware and software) was able to run for 30 min reading from four ADC cards and evaluate and output eight spatial channels. The real-time system was able to keep a 60-Hz rate, read out the ADCs, process the laser data, and output the results to the AO card. All of this was accomplished with commercially available hardware. In the future, the AO card will output the computed plasma parameters to the NSTX-U PCS, which will use this input to control the plasma in NSTX-U in real time. To test this system prototype further, it will be installed at the TS diagnostic of the LHD stellarator.[23]

Acknowledgments

This work was supported by the U.S. Department of Energy under DC-AC02-09Ch11466, DE-SC0015878, and DE-SC0015480. NSTX-U is sponsored by the U.S. Department of Energy Office of Fusion Energy Sciences. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

ORCID

Roman Rozenblat http://orcid.org/0000-0002-4338-090X

References

1. I. H. HUTCHINSON, *Principles of Plasma Diagnostics*, 2nd ed., Cambridge University Press (2005).
2. R. PASQUALOTTO et al., "High Resolution Thomson Scattering for Joint European Torus (JET)," *Rev. Sci. Instrum.*, **75**, 3891 (2004); https://doi.org/10.1063/1.1787922.
3. D. JOHNSON et al., "TFTR Thomson Scattering System," *Rev. Sci. Instrum.*, **56**, 1015 (1985); https://doi.org/10.1063/1.1138255.
4. T. N. CARLSTROM et al., "Design and Operation of the Multipulse Thomson Scattering Diagnostic on DIII-D," *Rev. Sci. Instrum.*, **63**, 4901 (1992); https://doi.org/10.1063/1.1143545.
5. B. KURZAN et al., "Improvements in the Evaluation of Thomson Scattering Data on ASDEX Upgrade," *Rev. Sci. Instrum.*, **72**, 1111 (2001); https://doi.org/10.1063/1.1321747.
6. J. W. HUGHES et al., "High-Resolution Edge Thomson Scattering Measurements on the Alcator C-Mod Tokamak," *Rev. Sci. Instrum.*, **72**, 1107 (2001); https://doi.org/10.1063/1.1319367.
7. R. SCANNELL and D. T. GODDARD, "A Calibration Method for Lateral Forces for Use with Colloidal Probe Force Microscopy Cantilevers.," *Rev. Sci. Instrum.*, **79**, 10E730 (2008); https://doi.org/10.1063/1.2971971.
8. K. NARIHARA et al., "Design and Performance of the Thomson Scattering Diagnostic on LHD," *Rev. Sci. Instrum.*, **72**, 1122 (2001); https://doi.org/10.1063/1.1319368.
9. K. ZHAI et al., "Performance of the Thomson Scattering Diagnostic on Helical Symmetry Experiment," *Rev. Sci. Instrum.*, 75, 3900 (2004); https://doi.org/10.1063/1.1788835.
10. E. PASCH et al., "Systematic Effects from an Ambient-Temperature, Continuously Rotating Half-Wave Plate," *Rev. Sci. Instrum.*, **87**, 11E729 (2016); https://doi.org/10.1063/1.4962248.
11. D. W. JOHNSON et al., "APD Detector Electronics for the NSTX Thomson Scattering System," *Rev. Sci. Instrum.*, **72**, 1129 (2001); https://doi.org/10.1063/1.1321751.
12. A. DIALLO et al., "Prospects for the Thomson Scattering System on NSTX-Upgrade," *Rev. Sci. Instrum.*, **83**, 10D532 (2012); https://doi.org/10.1063/1.4740267.

13. B. P. LeBLANC et al., "Radial Resolution Enhancement of the NSTX Thomson Scattering Diagnostic," *Rev. Sci. Instrum.*, **83**, 10D527 (2012); https://doi.org/10.1063/1.4738655.

14. B. P. LeBLANC, "A Calibration Method for Lateral Forces for Use with Colloidal Probe Force Microscopy Cantilevers," *Rev. Sci. Instrum.*, **79**, 10E737 (2008); https://doi.org/10.1063/1.2956747.

15. "SIS3316 16 Channel VME Digitizer Family," Struck Innovative Systeme; https://www.struck.de/sis3316.html (current as of May 2018).

16. Solarflare website; https://www.solarflare.com/technologies-overview (current as of May 2018).

17. General Standards Corporation, High Performance Bus Interface Solutions website; http://www.generalstandards.com/view-products2.php?BD_family=16ao64c (current as **of** May 2018).

18. MDSplus website; http://www.mdsplus.org/index.php/Introduction (current as of May 2018).

19. M. JOSEPH and P. KESHWANI, "Comparison Between Linear Search and Binary Search Algorithms," KITE/NCISRDC/IJARIIT/2018/CSE/107, IJARITT; https://www.ijariit.com/conference-proceedings/15%20CSE%20107.pdf (current as of May 2018).

20. F. M. LAGGNER et al., "A Real Time Framework for the Multi Point Thomson Scattering Diagnostic at NSTX-U," *Rev. Sci. Instrum.*, **90**, *4* (2019); https://doi.org/10.1063/1.5088248.

21. GCC, The GNU Compiler Collection website; http://gcc.gnu.org/ (current as of May 2018).

22. N. ISHKOV, "A Complete Guide to Linux Process Scheduling," M.Sc. Thesis, University of Tampere; https://tampub.uta.fi/bitstream/handle/10024/96864/GRADU-1428493916.pdf (current as of May 2018).

23. K. NARIHARA et al., "Design and Performance of the Thomson Scattering Diagnostic on LHD," *Rev. Sci. Instrum.*, **72**, 1122 (2001); https://doi.org/10.1063/1.1319368.