Performance Improvement in Real-Time Mapping of Thomson Scattering Data to Flux Coordinates in LHD[*)]

Masahiko EMOTO, Masanobu YOSHIDA, Chihiro SUZUKI, Yasuhiro SUZUKI, Katsumi IDA, Yoshio NAGAYAMA, Tsuyoshi AKIYAMA, Kazuo KAWAHATA, Kazumichi NARIHARA, Tokihiko TOKUZAWA and Ichihiro YAMADA

National Institute for Fusion Science, Toki, Gifu 509-5292, Japan

(Received 30 December 2011 / Accepted 15 March 2012)

More than 100 diagnostic devices are attached to the vacuum vessel of the Large Helical Device (LHD); they measure various aspects of the plasma physics. Because the shape of the LHD plasma is not symmetric, each diagnostic obtains the physical values in a different cross section. For example, the Thomson scattering system measures the electron temperature profile in the horizontally elongated cross section, and the laser interferometer measures the line-integrated electron density profile in the vertically elongated cross section. To analyze the data obtained by different diagnostics, their measurement positions must be mapped to a unified coordinate system, the flux coordinate system. Therefore, the authors have been building a database to map the physical coordinates to the flux coordinates. A system for mapping the electron temperature profile to the flux coordinates, TSMAP, has been developed using the database. The profiles calculated by TSMAP are fundamental data for analyzing the plasma physics during an experiment. Therefore, they must be obtained as soon as possible. However, the execution of TSMAP requires computational power, and the performance of a typical personal computer is not high enough to keep up with the 3-min plasma discharge cycle. To increase the performance, the authors use a parallel computing approach. Because the fitting calculation for each time is independent, the calculations for different times can be executed simultaneously. Using this approach, the authors increased the performance by 25 times, achieving a 25-s execution time.

Keywords: PC cluster, real-time, Python

DOI: 10.1585/pfr.7.2405058

1. Introduction

Flux coordinates are more useful than Cartesian coordinates for analyzing the plasma behavior in the Large Helical Device (LHD). For example, the YAG laser Thomson scattering system measures the electric temperature profile horizontally, whereas the far infrared (FIR) laser array measure the line-integrated electron density profile vertically. To obtain the electron density profile, these two results must be calibrated. For this purpose, they have to mapped to unique coordinates, or flux coordinates. Therefore, to analyze the data obtained by the different diagnostics, their measurement positions must be mapped to the flux coordinates. Therefore, the authors have developed a system called TSMAP (Thomson Scattering MAPping) to map the measured position from the Cartesian coordinates to the flux coordinates [1]. An example is shown in Fig. 1. The profiles are plotted for the effective minor radius r_{eff}. This is an averaged minor radius that is constant on the same flux surface; thus, it is a flux coordinate. By using the effective minor radius, one can compare the Thomson scattering data and the FIR laser interferometry data.

The principle of TSMAP is as follows. Because an electron moves along a field line with a speed comparable to the light velocity, the electron temperature on a flux surface is constant. TSMAP searches for an LHD equilibrium that fits the electron temperature profile, which is measured using the Thomson scattering system. The Thomson scattering measurement is executed 10 times per second in the LHD. A typical LHD plasma shot lasts for about 4 s, and the repetition time of the LHD plasma is 3 min. Thus, the mapping calculation should be done 40 times within 3 min. The equilibrium in a helical confinement system for typical experimental parameters is generally calculated in advance using the VMEC code [2]. TSMAP looks for the best fitted data from the database.

It currently takes more than 600 s for TSMAP to complete the calculation. To enhance the performance, the authors use two servers and six clients to calculate the profiles of different plasma shots simultaneously; this approach can keep up with the experimental cycle (Fig. 2). In this system, the authors use virtual machines as clients instead of physical machines. Using virtual machines makes it easy to enhance the performance simply by copying the virtual machine images. However, this method does not reduce

author's e-mail: emo@nifs.ac.jp
[*)] This article is based on the presentation at the 21st International Toki Conference (ITC21).

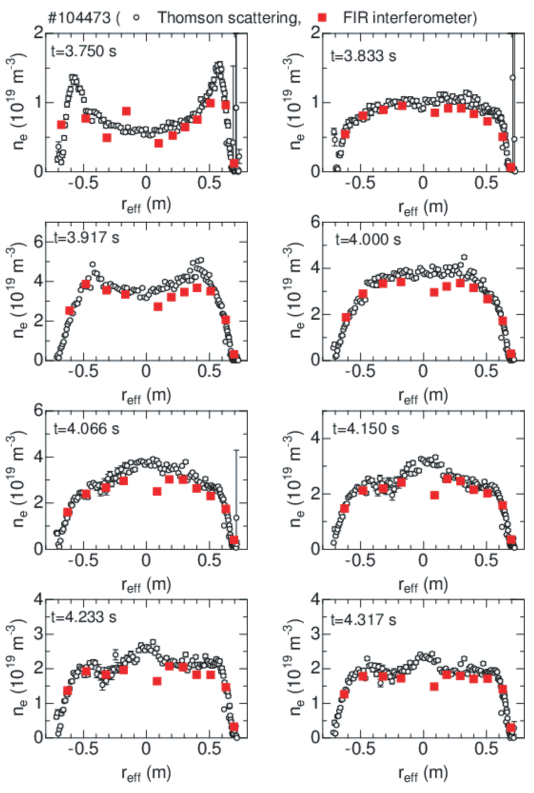


Fig. 1 Electron density profiles in flux coordinates measured by Thomson scattering and FIR interferometer.

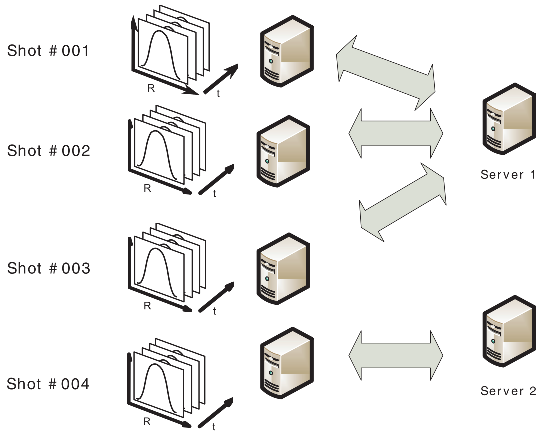


Fig. 2 Each computer calculates mapping data for a given shot number.

the calculation time for single-shot data. Therefore, it is necessary to enhance the calculation speed for a single shot. Because the calculation of a certain time is independent of the calculation of other times, these calculations can be executed simultaneously.

2. System Overview

Figure 3 shows an overview of the prototype system. It differs from the previous system in that the client program runs on one actual PC. The original program was

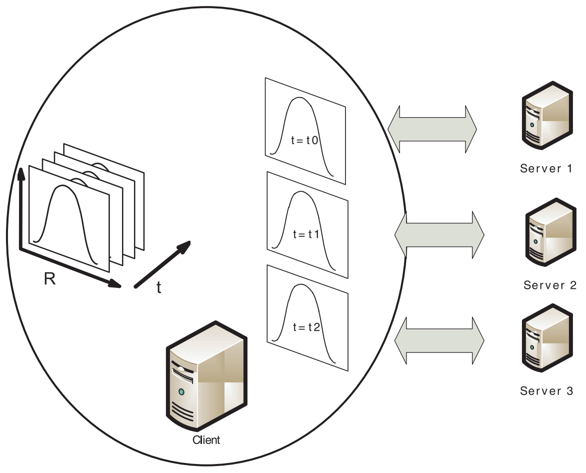
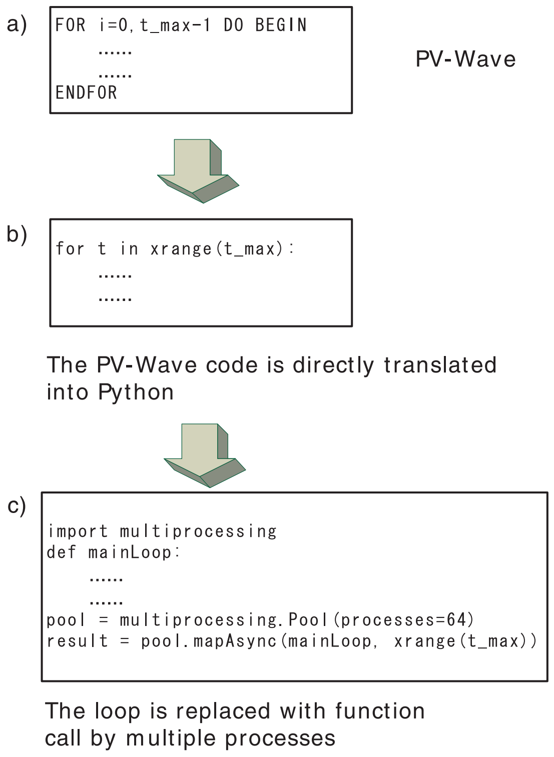


Fig. 3 Each process calculates the mapping data for a given time frame of the same shot number.

written in PV-Wave, but the new program was rewritten in Python. The reason is that it is difficult to implement parallel computing using PV-Wave. Many computer languages can be used to write parallel programs. One of the reasons to choose Python is that there are many scientific packages for handling experimental data. For example, NumPy [3] provides an array handling interface similar to that of PV-Wave, and it is relatively easy to migrate from PV-Wave to Python. To write a parallel program in Python, there are two useful standard packages, multithreading and multiprocessing. The former is used for multithread programming in Python. The latter was developed later to write multiprocess programs using interfaces similar to those of the multithreading package. Using multiple threads is generally faster than using multiple processes because less overhead is required to create a new thread than to create a new process. However, the most popular Python implementation, CPython, does not use a native thread library, and the Python interpreter has to switch the program context of threads by itself [4]. Even when the Python interpreter executes multiple threads, only one thread is running from the viewpoint of the operating system. Therefore, it cannot use multiple CPU cores at the same time even when the multithreading package is used, and it cannot take advantage of a multiple-CPU core architecture to reduce the calculation time. On the other hand, when the multiprocessing package is used, the context switch is managed by the operating system, and full use can be made of the multiple CPU cores. Other Python implementations exist, such as Jython, which is written in Java, or PyPy, which is written in Python itself. The Jython implementation uses the Java thread library, and it can use multiple CPU cores at the same time. However, it cannot use existing external libraries, such as NumPy, to handle LHD experiment data. Therefore, the authors chose CPython and a multiprocessing package instead of a multithreading package.

Figure 4 shows the essential parts of this program.



a)

```
FOR i=0,t_max-1 DO BEGIN
    ......
    ......
ENDFOR
```

b)

```
for t in xrange(t_max):
    ......
    ......
```

The PV-Wave code is directly translated
into Python

c)

```
import multiprocessing
def mainLoop:
    ......
    ......
pool = multiprocessing.Pool(processes=64)
result = pool.mapAsync(mainLoop, xrange(t_max))
```

The loop is replaced with function
call by multiple processes

Fig. 4 PV-Wave is rewritten in Python to be calculated by multiple processes.

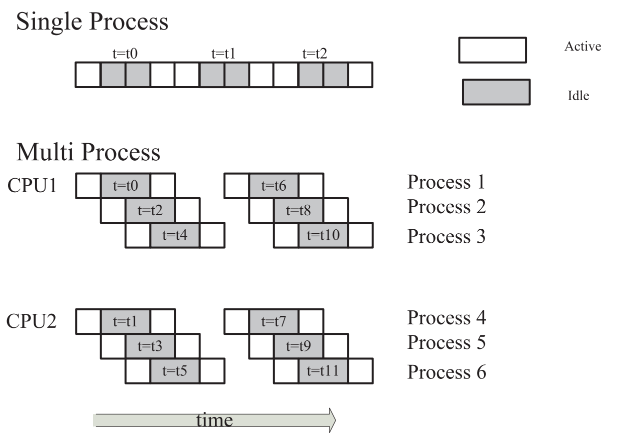


Fig. 5 Execution scheduling; the CPU can be used by one client process while another waits for the server results.

Table 1 Specifications: The client is the PC that runs TSMAP, and the server is the PC that provides the coordinate data for the client.

	Client	Servers
CPU	Core i7 920 (2.66 GHz)	Core i7 860, X5650, X5570, X5690 (2.67GHz-3.47 GHz)
Number of CPU cores	4 (8) Physical (Logical)	16—48 (32—96) Physical (Logical)

Figure 4-b) is the Python code, which is directly translated from the original PV-Wave code in Fig. 4-a). In the original code, the calculation is executed over time slices indexed by *t*. Because the calculation when $t = t_i$ is independent of the calculation when $t = t_j$, the code inside the loop can be executed in parallel. Therefore, Fig. 4-b) can be rewritten as shown in Fig. 4-c). The code inside the loop is calculated by multiple processes, in this case 64 processes, simultaneously. Each process calculates the best fitted value for a given time. TSMAP searches for the best fitting value in the flux coordinates for the changing plasma parameters.

TSMAP adopts a client-server architecture; it asks the severs for the flux coordinates corresponding to the actual coordinates, and TSMAP calculates the best fitted values in flux coordinates. By using the multiprocess scheme, TSMAP can use the client CPU power more efficiently; while one process is waiting for the server response, another can use the client CPU. Figure 5 shows an example. When only one process is running, 50% of the CPU time is idle because it must wait for the server response. However, when six processes are running at the same time in a two-CPU PC, the total calculation speed is 3 times faster than that of single process even though it has only two CPUs.

Because TSMAP is a client-server program, the performance depends on the number of servers as well as the number of processes. TSMAP currently uses two servers. To study the dependencies, the authors measured the calculation speed while increasing the number of processes and the number of servers. The hardware specifications used in these measurements are listed in Table 1.

3. Result

Figure 6 shows the results. The *x* axis indicates the number of processes, and the *y* axis is the relative calculation speed. The relative calculation speed is the reciprocal of the time that TSMAP spends on calculation. It is set to one when the number of processes is one. The calculation speed grows linearly when the number of processes is small, but the growth ratio declines as the number of processes increases. Moreover, as the number of servers (CPU cores) increases, the linearly growing part becomes large, and the calculation speed increases. When 4 servers or 40 CPU cores are used, the relative calculation speed reaches 25 when 64 processes are running. This means that the calculation is completed in 25 s.

When the number of server CPU cores is 40, the calculation speed seems to saturate when the number of processes is 30. If we can find the cause of the saturation, the calculation speed can be enhanced. The network is a possible bottleneck. However, during this measurement, the network traffic from the client to each server is at most 3 Mbps, and the client and the servers are connected via a 1 Gbps network. Therefore, it is difficult to believe that the network is the bottleneck. Another possibility is that the number of processes exceeds the client CPU capability.

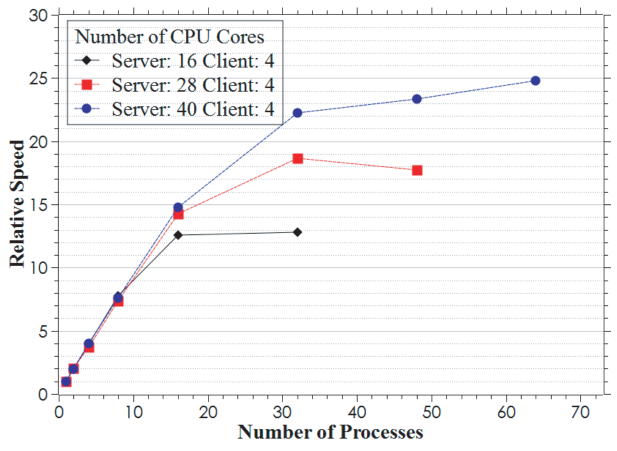


Fig. 6 Relative calculation speed vs. number of processes.

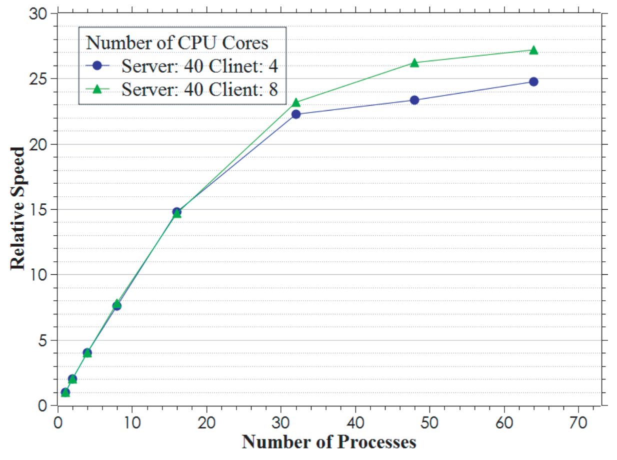


Fig. 7 Comparison of four-core client and eight-core client.

To verify this assumption, the calculation speed was measured using an eight-core PC instead of a four-core PC as the client PC. The difference between four and eight cores is shown in Fig. 7. The difference in the calculation speed is small, and the bottleneck does not seem to be the number of client CPU cores. Furthermore, because the CPU load of the server is about 30% at most, the number of server CPU cores is not the bottleneck, either. Therefore, the authors suspect that the bottleneck is the disk I/O, such as looking for data in the database or reading files from the disks. If this assumption is true, the performance can be enhanced by replacing the existing disk with a faster one, such as a solid-state disk. However, further analysis is required to verify this.

4. Conclusion

The authors modified the current TSMAP program to develop a prototype system for executing the calculations in parallel. With this system, the calculation speed becomes 25 times faster than that of the current system, or 25 s. This is not short enough to keep up with the 3-min plasma discharge cycle considering that TSMAP requires Thomson scattering and FIR data. However, using this method with other analysis, the authors think it is possible to realize real-time analysis during an experiment.

Acknowledgment

The authors thank the LHD experiment group and the technical staff of LHD for their support of this work. This work was supported by the National Institute for Fusion Science (budget no.: NIFS10ULHH014).

[1] M. Emoto *et al.*, 8th IAEA Technical Meeting, San Francisco, 2011.
[2] S.P. Hirshman and O. Betancourt, J. Comput. Phys. **96**, 99 (1991).
[3] http://www.numpy.org/
[4] D. Beazely, Python Concurrency Workshop, Chicago, 2009.