PROCEEDINGS OF SCIENCE

Towards real-time density profile reconstruction with CUDA

D. R. Ferreira,[a] **P. J. Carvalho**[*],[b] **H. Fernandes,**[b] **L. Meneses**[b] **and JET contributors**[†‡]
EUROfusion Consortium, JET, Culham Science Centre, Abingdon, OX14 3DB, UK
[a]*Instituto Superior Técnico, Universidade de Lisboa*
Campus do Taguspark, Avenida Prof. Dr. Cavaco Silva, 2744-016 Porto Salvo, Portugal
[b]*Instituto de Plasmas e Fusão Nuclear, Instituto Superior Técnico, Universidade de Lisboa*
Av. Rovisco Pais, 1049-001 Lisboa, Portugal
E-mail: pedro.carvalho@ipfn.tecnico.ulisboa.pt

Some present-day fusion diagnostics, including the reconstruction of the density profile from reflectometry measurements, require a lot of processing power to achieve the desired results. This processing is thus usually postponed to after the pulse and can last well beyond the starting of the next pulse on standard tokamaks like JET or ASDEX Upgrade. With the parallel computing capabilities of modern GPUs, it is possible to significantly shorten the time it takes to compute the density profile, bringing it closer to the update interval of the JET real-time network. This paper presents the implementation of profile reconstruction from X-mode Frequency-Modulated Continuous-Wave (FMCW) reflectometry for JET's KG10 diagnostic on an NVIDIA® CUDA-enabled GPU. The system can process one profile in less than 5ms, a temporal resolution that opens the prospects of using FMCW reflectometry as a real-time diagnostic.

1st EPS conference on Plasma Diagnostics
14-17 April 2015
Frascati, Italy

[*]Speaker.

[†]See the Appendix of F. Romanelli et al., Proceedings of the 25th IAEA Fusion Energy Conference 2014, Saint Petersburg, Russia

[‡]This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. IST activities also received financial support from *Fundação para a Ciência e Tecnologia* through project Pest-OE/SADG/LA0010/2013. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

© Copyright owned by the author(s) under the terms of the Creative Commons Attribution-NonCommercial-ShareAlike Licence. http://pos.sissa.it/

1. Introduction

Frequency-modulated continuous-wave (FMCW) reflectometry is a plasma diagnostic which can be used to determine the density profile of a magnetically confined plasma along the major radius of a tokamak. In JET, the plasma is probed from the low-field side with a series of frequencies $\{f_0, f_1, ..., f_n\}$ in the microwave range, including the Q-band (33–50 GHz), V-band (50–75 GHz), W-band (75–110 GHz), and D-band (110–150 GHz). The probing frequency f_i is increased to f_{i+1} in steps of 0.01 to 0.02 GHz, and the system is sufficiently fast to sweep the plasma in 10μs and to provide a new sweep every 15μs, if necessary [1]. However, at such rate the generated data quickly exceeds the amount of available memory, so the highest acquisition rates are reserved for events of special interest. An acquisition rate of one sweep per millisecond is common.

The underlying principle of microwave reflectometry is that a probing wave of frequency f_i is reflected at a cutoff position x_i inside the plasma where there is some electron density $n_e(x_i)$. While the density value $n_e(x_i)$ that causes total reflection can usually be predicted from f_i, the actual cutoff position x_i where such reflection occurs is unknown and must be determined based on the roundtrip delay of the probing wave.

1.1 Finding the group delay

As the plasma is swept with a series of frequencies $\{f_0, f_1, ..., f_n\}$, the reflectometer generates an in-phase/quadrature (I/Q) signal [2] with time-varying amplitude and phase. With the increase in probing frequency, the phase varies in such a way that the I/Q signal appears to oscillate at a beat frequency which is related to the group delay τ by (see, e.g. [3]):

$$f_{\text{beat}}(f_i) = \tau(f_i)\frac{df}{dt} \tag{1.1}$$

where $\frac{df}{dt}$ is the step increase in probing frequency per unit time.

The beat frequency $f_{\text{beat}}(f_i)$ can be obtained using a sliding-window approach, which consists in applying a Short-Time Fourier Transform (STFT) to a segment of signal around f_i. For this purpose, we use a segment of 64 samples, we apply a weighting window to the segment, and we use zero-padding to improve the FFT resolution. We compute the FFT of the zero-padded segment and we find the point at which the FFT has its maximum. The procedure is illustrated in Figure 1.

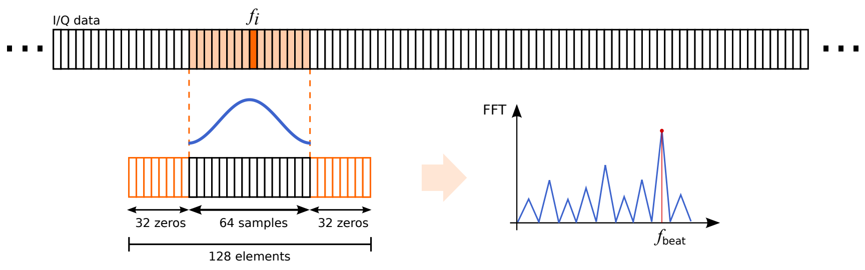

Figure 1: Computing the beat frequency for a probing frequency f_i

1.2 Group delay inside the plasma region

From the beat frequency $f_{\text{beat}}(f_i)$ it is possible to obtain the group delay $\tau(f_i)$ from Eq. (1.1). However, such group delay includes the total propagation time, both inside and outside the plasma.

Since we are interested in measuring the delay only within the plasma region, we need to subtract the propagation time outside the plasma. Such propagation time can be determined beforehand by probing the empty chamber with the same set of frequencies. In this case, the probing wave is reflected on the back wall. Let $f_{\text{wall}}(f_i)$ represent the beat frequency obtained when probing the empty chamber. Then the roundtrip delay inside the plasma region is given by:

$$\tau(f_i) = \frac{f_{\text{beat}}(f_i)}{df/dt} - \left(\frac{f_{\text{wall}}(f_i)}{df/dt} - \frac{2(R_0 - R_{\text{wall}})}{c} \right) \tag{1.2}$$

where R_0 is the first (outermost) position where there is usually some plasma density, R_{wall} is the position of the back wall,[1] and c is the speed of light in vacuum.

1.3 Linearization of f_{wall}

In practice, the measurement of both $f_{\text{beat}}(f_i)$ and $f_{\text{wall}}(f_i)$ is subject to noise. In $f_{\text{beat}}(f_i)$ it is hard to distinguish between noise and actual density fluctuations. However, in $f_{\text{wall}}(f_i)$ there should be no noise because the chamber is empty and the back wall is at a fixed position. Yet, Figure 2 shows that while $f_{\text{wall}}(f_i)$ follows an overall linear trend, there are some outliers that deviate significantly from that trend. Here, least squares regression (LS) provides a poor fit; the linear trend is best captured by a least median of squares (LMedS) [4]. For this purpose, we search for the two points that yield a line with the best fit. An inconvenience of this method is that its time complexity is $O(k^2)$ where k is the number of points. However, this linearization needs to be done only once per pulse, and applies to every sweep thereafter.

1.4 Computing the density profile

To find the cutoff positions $\{x_0, x_1, ..., x_n\}$ from the group delays $\{\tau(f_0), \tau(f_1), ..., \tau(f_n)\}$ we use Mazzucato's algorithm [5]. This is an iterative procedure which, at each iteration i, computes x_i based on the positions $\{x_0, x_1, ..., x_{i-1}\}$ from previous iterations. Due to the sequential nature of this algorithm, it can hardly be parallelized, but it can be somewhat optimized. Our main efforts are focused on parallelizing the computation of the beat frequencies and the linearization of f_{wall}. Figure 3 shows a plot of a density profile obtained from a sample sweep. For comparison, the plot also shows the density measurements obtained via Thomson scattering [6].

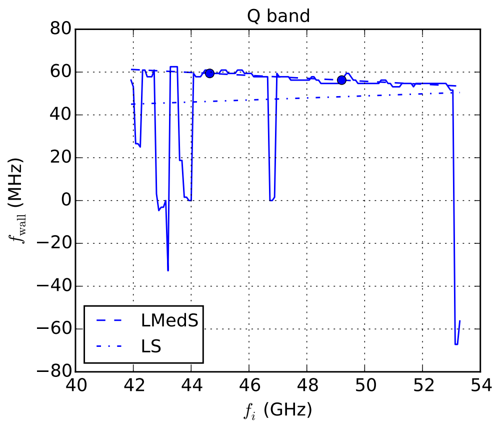

Figure 2: f_{wall} in the Q-band

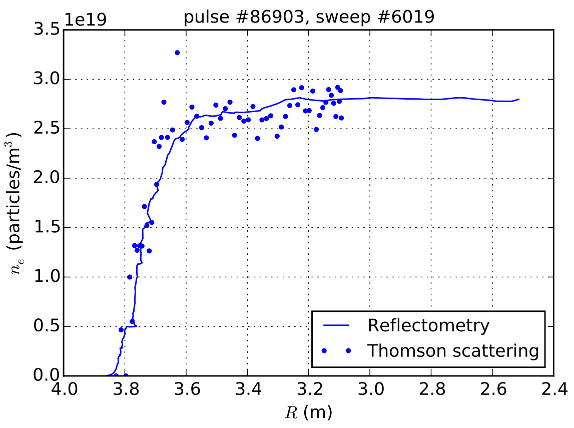

Figure 3: Density profile for a sample sweep

[1]In JET, R_0 is usually very close to the radial outer gap (3.86 m) and the back wall is positioned at 1.78 m.

2. Computing the density profile with CUDA

Modern GPUs have hundreds or even thousands of cores, so they can largely outperform the CPU in certain parallelizable tasks. CUDA [7] is a technology introduced by NVIDIA to make the parallel capabilities of GPUs accessible for general-purpose programming. CUDA revolves around the idea of dividing work into a large number of independent threads. The actual work to be carried out by each thread is programmed into a special function called a *kernel*. In essence, a kernel executes on the GPU and is replicated into as many threads as necessary. Each thread has a unique thread id which can be used to differentiate between the multiple executions of the same kernel. Typically, a CUDA kernel operates on one or more input arrays, and writes the results to one or more output arrays. These arrays are stored in GPU memory. The thread id is used to determine exactly on which elements of the input and output arrays each thread will operate.

2.1 Computing the beat frequencies

To compute the beat frequencies, we use three different kernels and also the cuFFT library [8]. Figure 4 illustrates the procedure. Kernel #1 builds all the input segments for the FFT. It receives the I/Q data for all bands in an input array and builds all the segments in an output array which is initialized with zeros (no need for an additional zero-padding step). Each value is copied to the output array by a separate thread, which also multiplies by the corresponding (Hanning) window point. The output of kernel #1 is given as input to cuFFT, which runs in batch mode to compute all FFTs on the GPU with a single call. Kernel #2 computes the magnitude of each FFT coefficient.

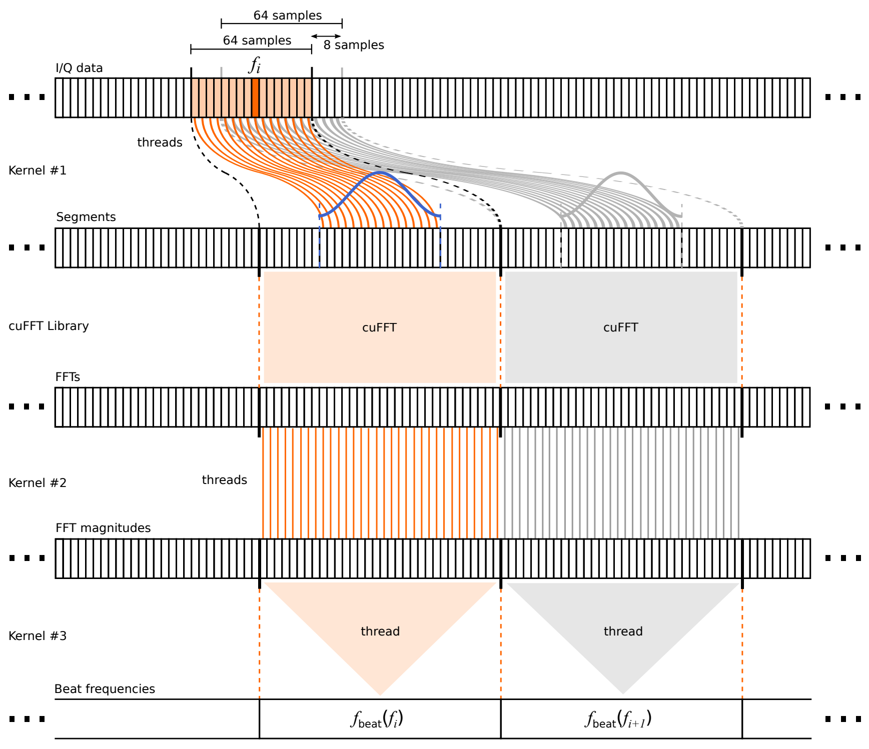

Figure 4: Computing the beat frequencies (CUDA version)

Kernel #3 finds the beat frequency for each segment; here, each thread operates on its own segment to find the position in that segment where the FFT has its highest peak.

2.2 Linearizing f_{wall}

The linearization of f_{wall} is done separately for each band, because each band has its own line that yields the best fit. To find such line, we need to consider every pair of points $(f_i, f_{\text{wall}}(f_i))$ and $(f_j, f_{\text{wall}}(f_j))$. We pass a line through those two points and compute the residues for every point. Then we sort the residues to find the median residue for that line. The line that yields the best fit is the one with the least median residue.

The procedure for obtaining f_{wall} from I/Q data is the same as described in Section 2.1. To linearize f_{wall} we use three additional kernels and two external libraries. Figure 5 illustrates this. Kernel #4 picks a pair of points and calculates the line parameters (slope and y-intercept) for the line that passes through those two points. Kernel #5 calculates the residues for each point and line. The residues for the same line are stored contiguously in a segment of the output array. Each of these segments must be sorted to find the median residue for the corresponding line. For this purpose, we use the Modern GPU library [9] to sort all segments at once on the GPU. Kernel #6 collects the median residue for each line and places it in an output array. Then we use the Thrust library [10] to find the position of the least median residue in that array and. from the same position in the array of line parameters, we retrieve the line that yields the best fit.

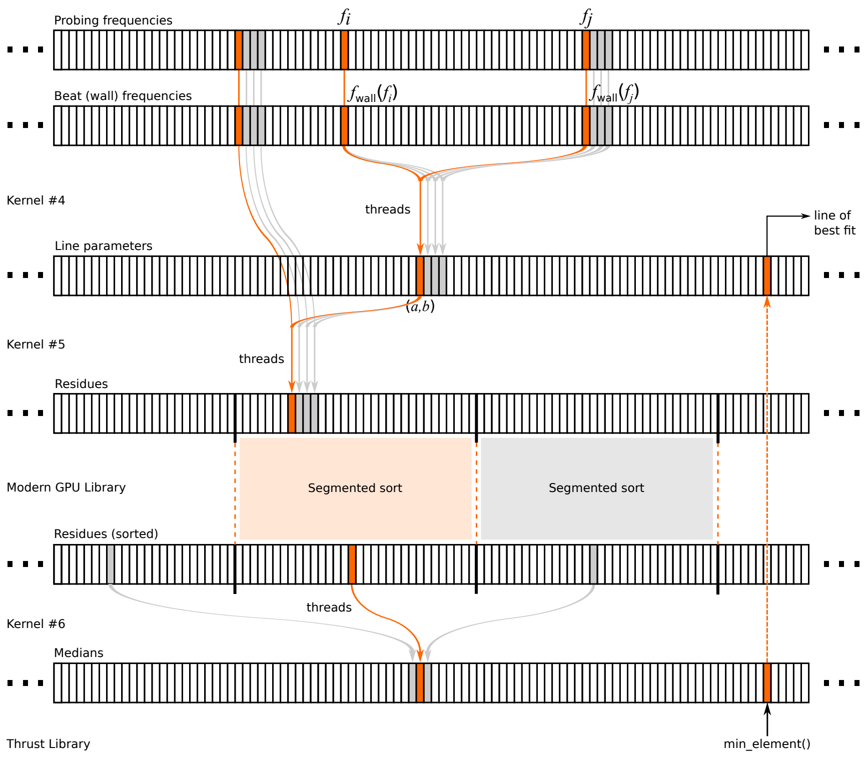

Figure 5: Finding the best linear fit for f_{wall} (CUDA version)

2.3 Results

To assess the performance of our CUDA approach, we compare it with a reference implementation in C that uses the FFTW library [11]. Table 1 shows the processing time of each sweep for both versions. The largest performance gain is observed in the linearization of f_{wall}. But more importantly, the reconstruction of the density profile is brought down to about 3.4ms, a time that is now much closer to the standard update interval of 2ms in the JET real-time ATM network [12]. The additional time required for data acquisition can be pipelined with the density profile reconstruction (i.e. the data for the next profile can be loaded while the current profile is being processed) to yield a total processing time of under 5ms.

CPU: Intel Core i5-4690 @ 3.5 GHz GPU: NVIDIA GeForce GTX 750 Ti SM5.0 with 640 CUDA cores @ 1137 MHz			
Run time (s)	C version	CUDA version	Performance gain
f_{wall} linearization	1.240397	0.130477	9.5x
sweep #6019	0.011204	0.003446	3.3x
sweep #10019	0.011178	0.003402	3.3x
sweep #73185	0.011247	0.003429	3.3x

Table 1: Processing time for three sample sweeps from pulse #86903

References

[1] A. Sirinelli, B. Alper, C. Bottereau, F. Clairet, L. Cupido, J. Fessey, C. Hogben, L. Meneses, G. Sandford, M. J. Walsh, and JET-EFDA Contributors. Multiband reflectometry system for density profile measurement with high temporal resolution on JET tokamak. *Rev. Sci. Instrum.*, 81(10), 2010.

[2] S. Hacquin, L. Meneses, L. Cupido, N. Cruz, L. Kokonchev, R. Prentice, and C. Gowers. Upgrade of the X-mode reflectometry diagnostic for radial correlation measurements in the Joint European Torus. *Rev. Sci. Instrum.*, 75(10):3834–3836, 2004.

[3] G. Cunningham. Use of the absolute phase in frequency modulated continuous wave plasma reflectometry. *Rev. Sci. Instrum.*, 79(8), 2008.

[4] D. L. Massart, L. Kaufman, P. J. Rousseeuw, and A. Leroy. Least median of squares: a robust method for outlier and model error detection in regression and calibration. *Anal. Chim. Acta*, 187:171–179, 1986.

[5] E. Mazzucato. Microwave reflectometry for magnetically confined plasmas. *Rev. Sci. Instrum.*, 69(6):2201–2217, June 1998.

[6] C. W. Gowers, B. W. Brown, H. Fajemirokun, P. Nielsen, Y. Nizienko, and B. Schunke. Recent developments in LIDAR Thomson scattering measurements on JET. *Rev. Sci. Instrum.*, 66(1):471–475, 1995.

[7] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, 2008.

[8] NVIDIA Corporation. cuFFT library user's guide, August 2014. http://docs.nvidia.com/cuda/cufft/.

[9] S. Baxter. Modern GPU, 2013. http://nvlabs.github.io/moderngpu/.

[10] NVIDIA Corporation. Thrust quick start guide, August 2014. http://docs.nvidia.com/cuda/thrust/.

[11] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proc. IEEE*, 93(2):216–231, February 2005.

[12] R. Felton, K. Blackler, S. Dorling, A. Goodyear, O. Hemming, P. Knight, M. Lennholm, F. Milani, F. Sartori, and I. Young. Real-time plasma control at JET using an ATM network. In *Proceedings of the 11th IEEE NPSS Real Time Conference*, pages 175–181, 1999.