applied sciences

MDPI

Article

Acceleration of an Algorithm Based on the Maximum Likelihood Bolometric Tomography for the Determination of Uncertainties in the Radiation Emission on JET Using Heterogeneous Platforms

Mariano Ruiz [1,*], Julián Nieto [1], Víctor Costa [1], Teddy Craciunescu [2], Emmanuele Peluso [3], Jesús Vega [4], Andrea Murari [5] and JET Contributors [†]

[1] Instrumentation and Applied Acoustic Research Group, Universidad Politécnica de Madrid, 28031 Madrid, Spain; julian.nieto.valhondo@upm.es (J.N.); victor.costa.perez@upm.es (V.C.)
[2] National Institute for Laser, Plasma and Radiation Physics, Magurele, 077125 Bucharest, Romania; c.teddy@ifa-mg.ro
[3] Department of Industrial Engineering, University of Rome 'Tor Vergata', Via del Politecnico 1, 00133 Rome, Italy; emmanuele.peluso@uniroma2.it
[4] Laboratorio Nacional de Fusión, CIEMAT, Av. Complutense 40, 28040 Madrid, Spain; jesus.vega@ciemat.es
[5] Consorzio RFX (CNR, ENEA, INFN, Universita' di Padova, Acciaierie Venete SpA), Corso Stati Uniti 4, 35127 Padova, Italy; andrea.murari@istp.cnr.it
* Correspondence: mariano.ruiz@upm.es
† JET Contributors are listed in the Acknowledgements.

Citation: Ruiz, M.; Nieto, J.; Costa, V.; Craciunescu, T.; Peluso, E.; Vega, J.; Murari, A.; JET Contributors. Acceleration of an Algorithm Based on the Maximum Likelihood Bolometric Tomography for the Determination of Uncertainties in the Radiation Emission on JET Using Heterogeneous Platforms. *Appl. Sci.* **2022**, *12*, 6798. https://doi.org/10.3390/app12136798

Academic Editor: Giuseppe Marco Tina

Received: 5 June 2022
Accepted: 30 June 2022
Published: 5 July 2022

Abstract: In recent years, a new tomographic inversion method based on the Maximum Likelihood (ML) approach has been adapted to JET bolometry. Apart from its accuracy and reliability, the key advantage is its ability to provide reliable estimates of the uncertainties in the reconstructions. The original algorithm was implemented and validated using the MATLAB software tool. This work presents the accelerated version of the algorithm implemented using a compatible ITER fast controller platform with the Ubuntu 18.04 or the ITER Codac Core System distributions (6.1.2). The algorithm has been implemented in C++ using the open-source libraries: ArrayFire, ALGLIB, and MATIO. These libraries simplify the management of specific hardware accelerators such as GPUs and increase performance. The speed-up factor obtained is approximately 10 times. The work presents the methodology followed, the results obtained, and the advantages and drawbacks of implementation.

Keywords: heterogeneous applications; MATLAB; ArrayFire; GPUs; C++ maximum likelihood; bolometry

1. Introduction

In Tokamaks, the measurement of the total emission of radiation and the reconstruction of the internal and local emissivity profiles is very important for the interpretation of Tokamak performance and for the design of experiments [1]. The reconstruction of the plasma radiation offers important information about the power exhaust and divertor detachment. In the case of a metallic first wall, the transport of heavy impurities, which may have a strong impact on the machine performance, can be identified by means of the radiation patterns. Important information related to instabilities that could potentially trigger disruptions can also be derived.

Bolometry is the diagnostic typically used for measuring radiation emission. On JET, this diagnostic is based on metal foil absorbers, which integrate the radiation emitted along a set of lines-of-sight (LOS) [2,3]. They allow a horizontal and a vertical view across the poloidal cross-section of the plasma, with an increased spatial resolution in the divertor area (Figure 1). This geometry allows tomographic reconstruction, but the mathematical

problem is an ill-posed one due to the limited number of views. Various tomographic methods have been proposed. Ingesson et al. [4] introduced a constrained optimization (ICO) method. ICO uses anisotropic smoothness on flux surfaces as an objective function and measurements as constraints. The reconstruction problem is discretized using a grid of pyramid local basis functions. ICO has been used in JET for more than two decades for various studies. It represents a landmark in what concerns the quality of reconstructions. The minimization of the Fisher information of the unknown 2D distribution has also been used to derive a bolometry reconstruction algorithm [5]. The Maximum likelihood (ML) bolometry implements a nonlinear iterative algorithm for estimating the emissivity distribution that is most consistent with the measured data, in the sense that, among the variety of all possible emissivity distributions, the algorithm chooses the one which maximizes the probability of obtaining the given experimental data set [6]. The ML algorithm also includes an iteratively smoothing procedure, based on the magnetic surfaces, either on closed or on open ones, given by the plasma equilibrium code used at JET. The ML method's key feature is its ability to routinely provide the confidence intervals. Comprehensive studies regarding the uncertainties in bolometric tomography on JET have been reported [7,8]. The proper assessment of the uncertainties is an important aspect of investigating high radiative discharges on JET with the ITER Like Wall [9]. Based on deep neural networks (DNN), a different conceptual approach has been proposed [10,11]. The network was trained using a database of reconstructions provided by the ICO method. Therefore, DNN generalizes the knowledge accumulated by using the ICO. The method can provide very fast reconstructions.

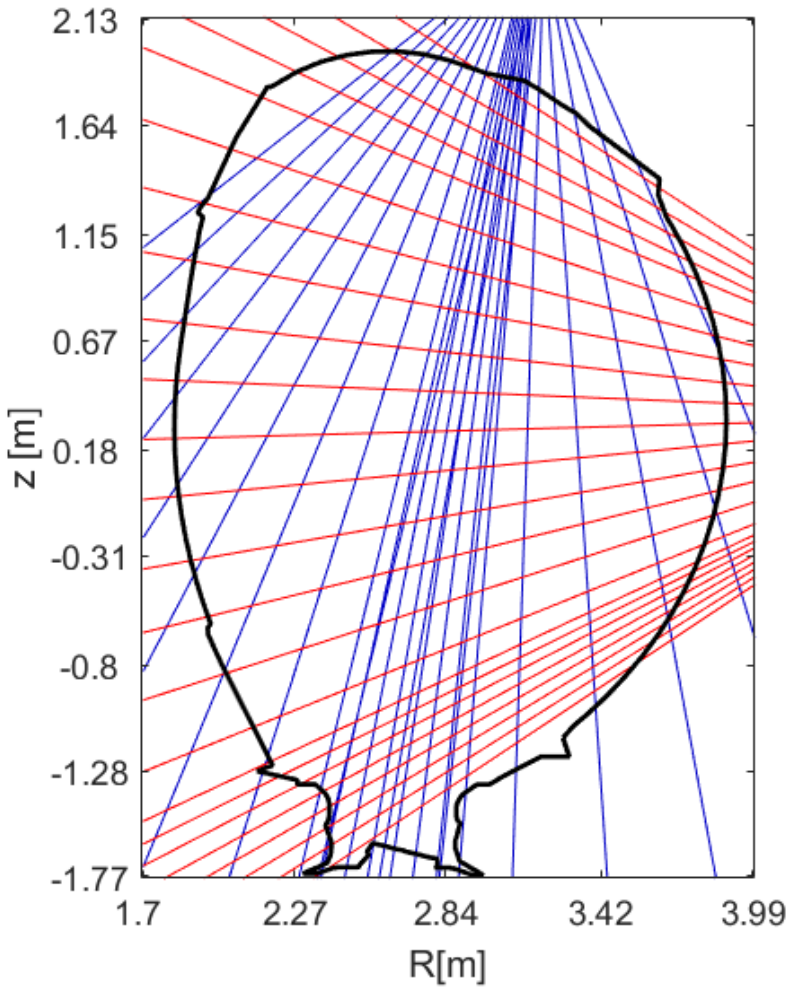

Figure 1. Schematic view of JET bolometric diagnostic layout.

In ASDEX, the routine radiated power computation relies on the code first developed in 1993 [12]. This code is based on a Tikhonov regularization scheme, which includes an anisotropic diffusion term that imposes smoothness on the flux surfaces and allows steeper gradients in the normal direction. Recently, the method has been improved by using a refined evaluation of the tomographic projection matrix which takes into account the full 3D field of view of each channel [13]. A new algorithm to improve the computation of total and local radiated power was recently developed [14].

A version of the constraint optimization approach, based on the error between the measured tomographic projections and backprojection and the smoothing function, is presented in this work [15].

Significant effort has been spent to achieve good-quality reconstruction with scarce tomographic geometries determined by the limited availability of location to install detectors (usually confined to ports of the vacuum vessel). The development of various techniques incorporating a priori information (mainly related to the magnetic configuration) played an essential role in ensuring a good spatial resolution. The temporal resolution that can be achieved in bolometry is suitable for studying rapid emissivity evolution. Bolometry became an increasingly requested diagnostic during experimentation, and, therefore, an increased amount of data should be processed. Therefore, the demand for high-speed computational methods is increasing quickly. The method based on neural networks represents a good solution for machines where a solid reconstruction database has been accumulated (by using a traditional approach, e.g., ICO on JET). However, for new machines, the implementation of very fast or even real-time algorithms is still at the beginning.

The code used in [6] is implemented in MATLAB, taking input data from the JET database formatted and organized accordingly. The execution time needed by MATLAB in the hardware platform used for the evaluation requires around 63 s, but the target execution time to obtain a real-time performance is 25 ms. Reducing the execution time implies that acceleration techniques are essential to approaching real-time performance.

2. Materials and Methods

The data used for the reconstruction were obtained from the bolometer diagnostic configuration and specific pulses of the JET database. These are:

- The magnetic surfaces, the first wall, and the last closed surface (or separatrix) cartesian coordinates.
- The time traces of a few related quantities such as the radial position of the magnetic axis and the vertical position of the so-called X-point.
- The data array obtained from JET bolometers that are averaged depending on the phenomenon to analyze, typically with a time window of 25 ms around the time instance of interest.

Once the algorithm is executed, among the outputs obtained, the back-calculated projections and the tomographic images can be listed.

2.1. MATLAB Algorithm

The implemented MATLAB algorithm can be split into three different steps. Firstly, the data are obtained and organized in different matrices. Secondly, several iterations configured by the user allow different quantities to be obtained, mainly the image of the emissivity distribution, the uncertainty one, the back-calculated projections, and the estimates of the radiated power in different locations of the main vessel. Figures 2–4 show the evolution of the reconstructed emissivity distribution, the related uncertainty, and the comparison between back-calculated and measured projections in subsequent iterations for the exemplificative JET pulse #85423 at 58.875 s.

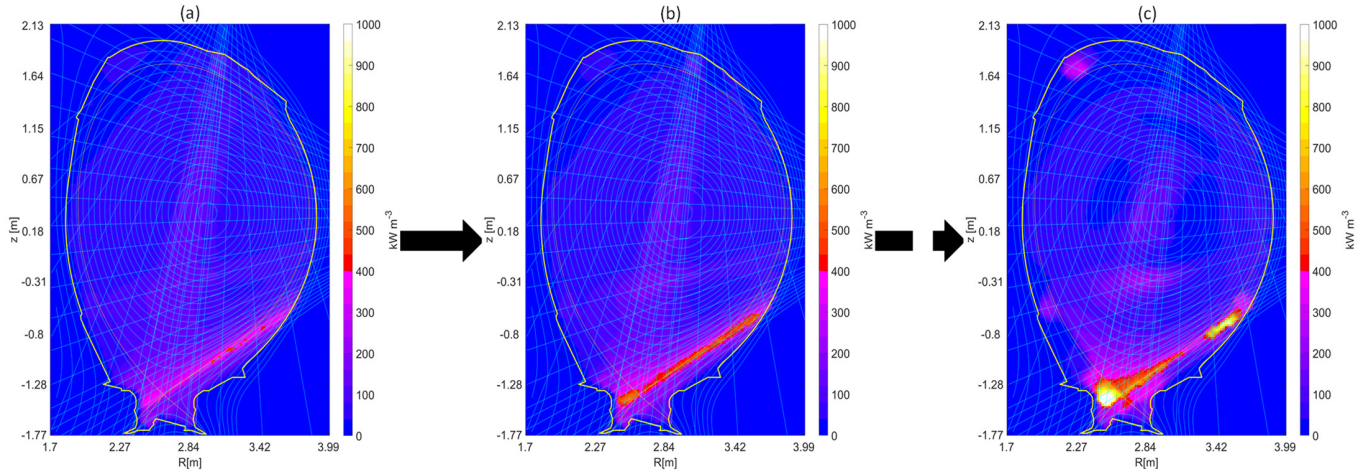

Figure 2. Evolution of the reconstruction ML (MATLAB) for iterations 1 (**a**), 2 (**b**), and 14 (**c**).

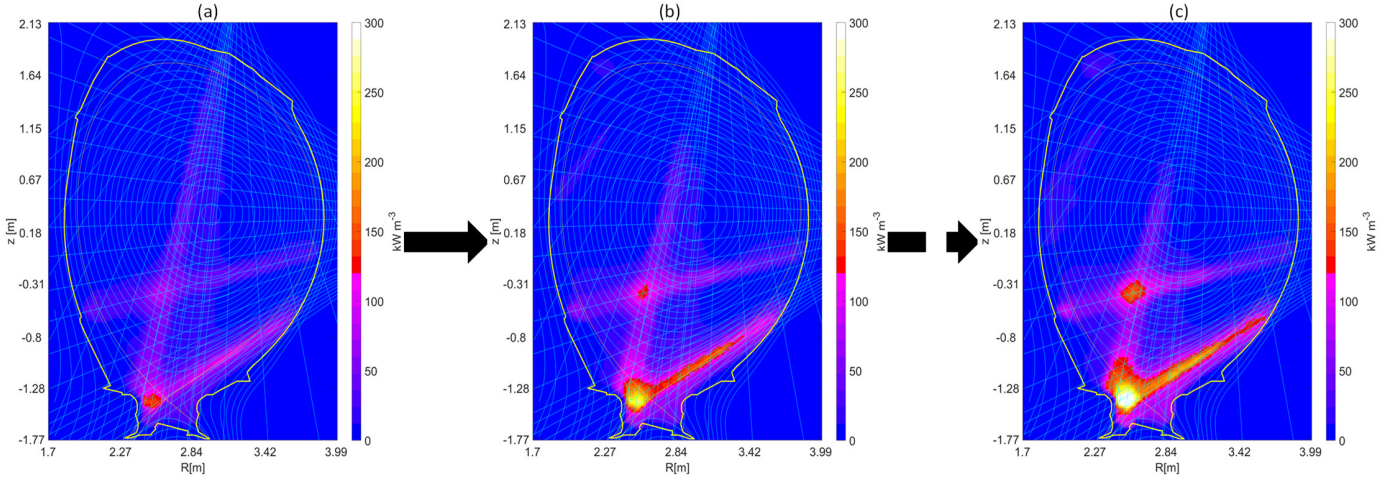

Figure 3. Evolution of the reconstruction uncertainty estimation process (MATLAB) for iterations 2 (**a**), 3 (**b**), and 4 (**c**).

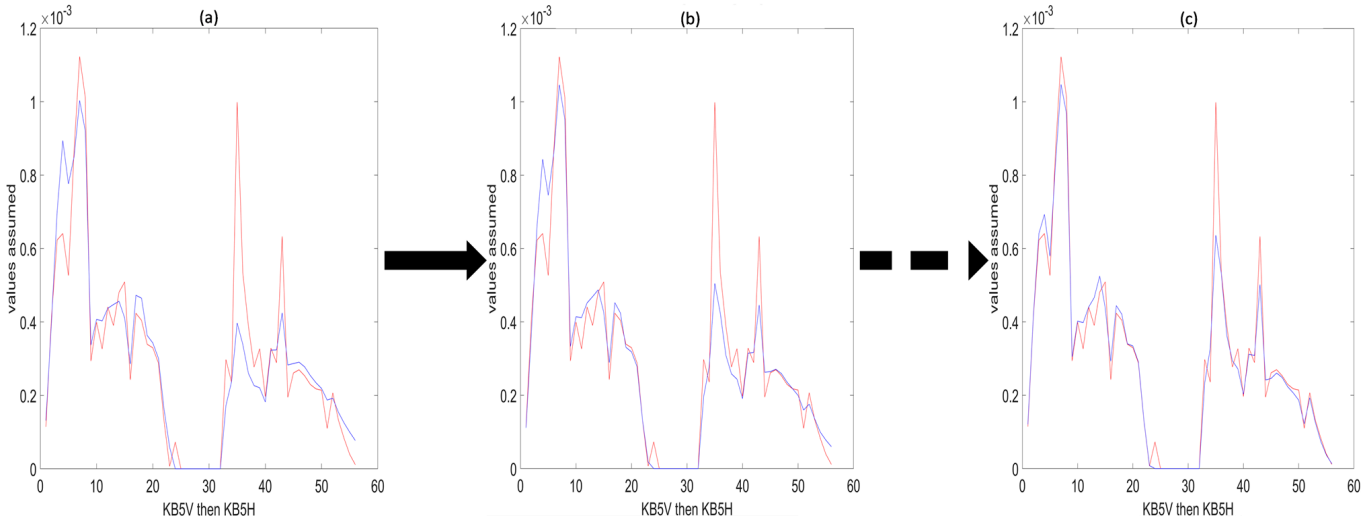

Figure 4. Measured (in red) and back-calculated projections (in blue) for iterations 1 (**a**), 2 (**b**), and 15 (**c**).

Finally, the main results are gathered and prepared to display the main information results after the number of iterations are executed. These are the final emissivity distribution

(Figure 5), its associated uncertainty (Figure 6), the radiation profile (Figure 7), and the reconstructed projections, as well as the estimates of the radiated powers in a different location outside of the main chamber, such as the total radiated power, the core one, i.e., the radiated power inside the Last Closed Surface (LCFS), the one below the Xpoint position, and the radiated power in the Scrape-Off Layer (SOL).

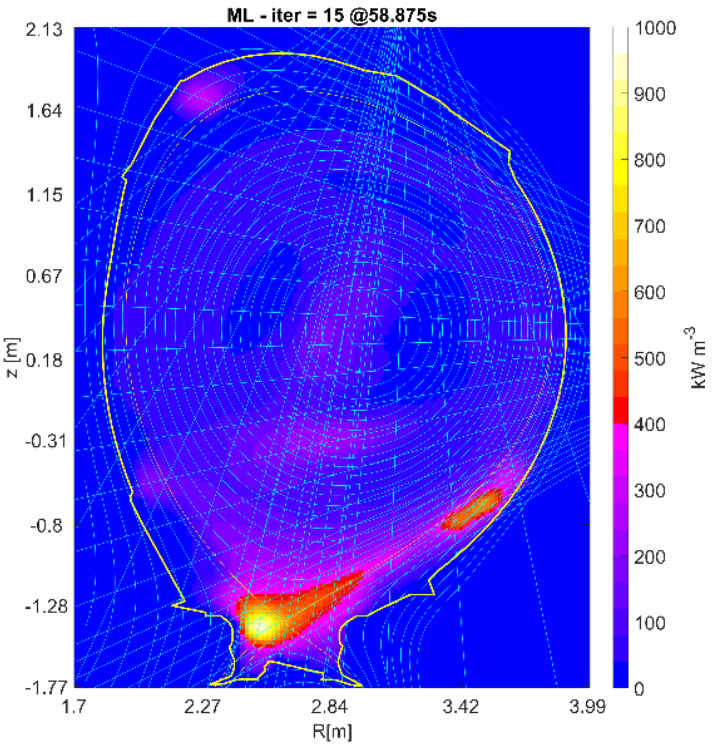

Figure 5. Final reconstruction.

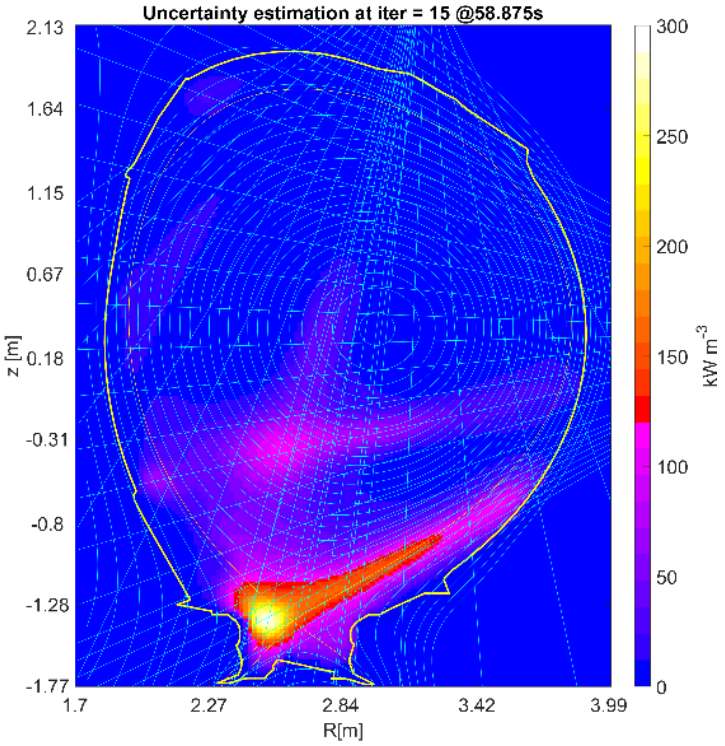

Figure 6. Final reconstruction uncertainty estimation.

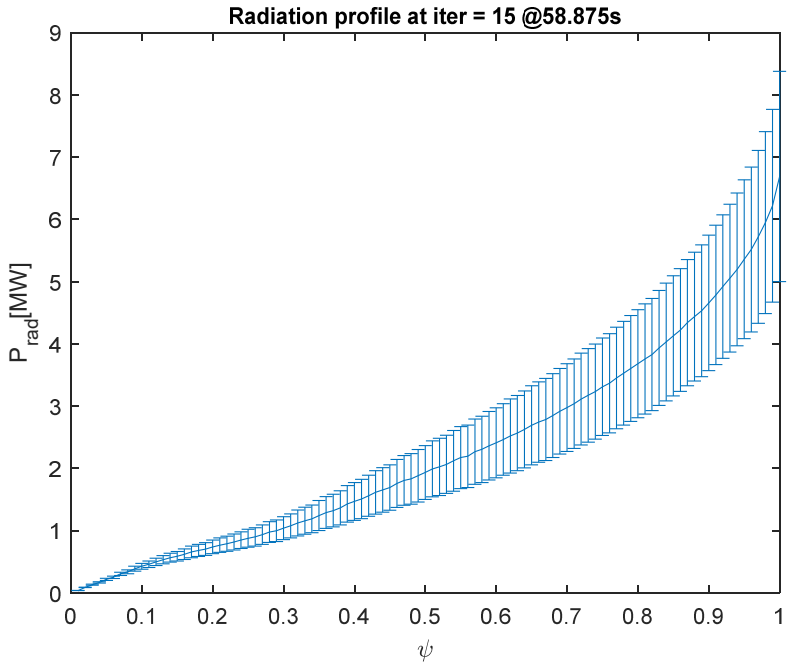

Figure 7. Final radiation profile as a function of the flux function $\psi \in [0, 1]$.

2.2. Analysis of the Different Alternatives

Off-line analysis of data using software tools such as MATLAB and programming languages such as Python are widespread in the scientific community [16]. Nevertheless, an alternative programming and deployment alternative must be used when a reduced execution time is needed to implement the algorithm in a real-time experiment (for example, in a nuclear fusion-related diagnostic) [17]. In addition, the use of graphical processing units (GPUs) has emerged as a powerful solution for massive data processing, especially in image processing with 1D or 2D detectors and cameras [18,19].

There are two different actions intended to reduce algorithm execution time. Firstly, the algorithm's steps must be identified, and secondly, the optimization options need to be evaluated. The methodology used to accelerate the algorithm execution follows these steps:

- MATLAB code analysis, which has an extension of approximately 2500 lines. This code uses high-level functions from MATLAB's toolboxes that solve curve fitting, 1D and 2D interpolations, and median and Gaussian image filtering. Unfortunately, from most parts of these functions, MathWorks does not provide the source code, complicating the implementation of the final solution.
- The code profiling analysis with the specific MATLAB tool to identify the time needed to execute the different algorithm phases. The algorithm is based on executing several configurable iterations, and the execution time depends on this.
- The analysis of the different options to shorten the execution time of the different code sections according to two different scenarios:
  a. Use of MATLAB:
     - Use of MATLAB Executable Files (MEX);
     - Use of parallel computing toolbox;
     - Use of MATLAB-optimized libraries;
     - Using GPUs in MATLAB (not all functions support GPU acceleration, and using GPU arrays in MATLAB is not efficient if the arrays are small).
  b. Use of other programming languages, rewriting the code in C++ with the help of optimized libraries for heterogeneous platforms:
     - CPU: ArrayFire, ALGLIB, libigl, armadillo, Intel MKL, IPP;
     - GPU: ArrayFire, CUDA (NVCC compiler, cuBLAS libraries, etc.);

    - FPGA: Use of IntelFPGA/XILINX acceleration techniques.
- Identification and selection of the most suitable hardware platform to accelerate a specific portion of the algorithm. Due to some functions' complexity, the most suitable hardware is considered to be a heterogeneous solution using an Intel multicore CPU and an NVIDIA GPU.
- Selection of the software to be used for the implementation considered criteria such as being Open Source, the best performance for CPU or GPU, and the possibility of using different programming languages.

Upon all these criteria, after a deep analysis, the solution was implemented by building a C++ application using the ArrayFire [20], ALGLIB [21], and MATIO [22] libraries, running in a commercial off-the-shelf (COTS) industrial computer with an Intel® Core™ i9 10980XE CPU @ 3 GHz, 128 GiB of RAM, and an NVIDIA RTX2080 SUPER GPU. The operating systems used for the development and testing were: Ubuntu 18.04 LTS and a REDHAT 7.4 with the ITER CODAC Core System version 6.3 installed.

The basic description of the libraries selected is explained in the following paragraphs.

2.2.1. ArrayFire

ArrayFire is an open-source library available for different programming languages such a C++, Python, and Rust. It has been developed to develop general-purpose solutions using GPGPU, widely used in high-performance computing, HPC [23,24]. ArrayFire has an API developed to simplify the development of applications that can be executed indistinctly in a CPU and/or one or multiple GPUs. GPU programming is hidden from the user, and, therefore, the learning curve is smoothed. ArrayFire follows the idea of "Code once, run anywhere," allowing code reuse. GPU programming can be deployed using CUDA (when supported) or OpenCL. ArrayFire focuses on developing specific array elements called ArrayFire arrays that are implemented in a different way than traditional arrays in C++ or Python. The ArrayFire API manages these array objects, which can be defined up to 4 dimensions of different types: single and double floating, booleans and integers, and complex numbers.

2.2.2. ALGLIB—C++/C# Numerical Analysis Library

ALGLIB is a numeric library focused on solving general numerical problems. It can be used with different programming languages such as C++, C#, and Delphi. It offers a great variety of functions for different science fields. In this specific application, it is required to interpolate the 2D data arrays that can or cannot be equally spaced (non-uniformly distributed). The development of the function implementing the equivalent to griddata requires the use of ALGLIB 2D interpolation functions for sparse/non-uniform data. For the fitting part, the least square solver function is used, for which two options are available: BlockLLS or FastDDDM. The FastDDDM option was chosen to achieve the best possible performance. However, the configuration parameters used do not correspond to the equivalent MATLAB "griddata" function, so there is a slight deviation from the original code.

2.2.3. MATIO

This library allows MATLAB files to be read and written. In addition, this library loads the source data (taken in the bolometry diagnostic) obtained from the JET database in the C++ application.

2.3. Profiling of MATLAB Code

The execution time of the original MATLAB-implemented algorithm was measured for the main functions and sections of the code using the MATLAB profiler. The most relevant execution times are displayed in Table 1. These times are obtained with 15 iterations. The total execution time for these iterations is around 63 s, which is fine for off-line processing but is very far from a real-time approach.

Table 1. Execution times (in ms) obtained with the MATLAB profiler.

Function	Time (ms)	Time for 15 Iterations
Initialization for smoothing on the open magnetic curves	3183.0	n/a
Load projections	3.0	n/a
Load geometry	165.0	n/a
Backprojections	10.0	n/a
Uncertainty preparations	20.0	n/a
Projection lines approach loop	2560.0	38,400.0
Smooth the closed and open surfaces	160.0	2400.0
Smooth LCF inside and outside	326.8	4902.0
Evaluation of emissivity	1.8	27.0
Uncertainty estimation	1363.5	5454.0
Evaluation of the reconstruction projection at iteration i^{th}	42.0	630.0
Final projections	1.2	n/a
Compute profile radiation and noise	7624.0	n/a
TOTAL	15,460.3	62,819.2

2.4. Development of the C++ Application and ArrayFire Optimization Techniques

ArrayFire provides a complete API that solves the most common functionalities implemented with MATLAB language. Therefore, it can be considered that porting MATLAB to C++ using ArrayFire API is relatively straightforward, and some parts of the code are even equivalent line by line. Nevertheless, ArrayFire does not include some powerful functions available in MATLAB. For example, the function "griddata" allows different types of interpolations using uniform and not-uniform input data distribution. This function in MATLAB has some parts of the internal code visible to the user, but other parts are not available, making it impossible to reproduce its calculations. While ArrayFire version 3.8.0 includes a function for interpolation, it expects that input data will be uniformly organized. To solve this problem, we chose the open-source library ALGLIB, which provides a set of functions for 2D interpolation that can be used to circumvent the problem. The translation from MATLAB to C++ followed the rule of performing precisely the same calculations in the different algorithm steps. Nonetheless, due to the use of functions such as griddata, which does not have a straightforward translation, the results are not precisely the same.

Once the conversion is completed, the code is optimized to improve execution time, maximizing parallelism and code execution in the GPU. The techniques applied are explained in the following paragraphs.

(a) Vectorization

The main technique to improve the execution time is the use of vectors. The objective is to execute the same operations in the different elements in a matrix. Of course, dependency between elements must not exist for vectorization to be a viable solution. One example of the application of this technique is the implementation using ArrayFire arrays of the following piece of code (see function projection_bolo) extracted from the "original" MATLAB code. In this case, pix(i,j,k) is the Projection matrix elements, which describes the geometrical contribution of each measurement to a certain pixel in the image. Rec1 is the reconstruction at the current iteration. The final result Prj represents the backprojection. In MATLAB this is implemented using loops and an if statement.

The exact equivalent code implemented in C++, with the help of ArrayFire, is presented below (see af:.array method projection_bolo). The first step is to obtain the portion of "ppix" with the relevant data to be processed, "ppixSlices". Then, the "rec1" 2D matrix is converted to a 3D matrix (repeating the data in the third dimension), using the function tile. Next, the two 3D matrices, "ppixSlices" and "rec1Sclices" are multiplied, which is the equivalent of the "pix(i,j,k)*rec1(i,j)" operation, for all the iterations, in the MATLAB code. The resulting 3D array is summed using ArrayFire's function af::sum. This operation

results in a 1D matrix which needs to have the dimensions rearranged, to match the results from the original MATLAB code.

```
function [prj] = projections_bolo(no_rays, nrpct_o, nrpct_v, pix, rec1)
        for k = 1:no_rays
        if (k <= 24) || (k >= 33)
        mysum = 0;
        for i = 1:nrpct_v
        for j = 1:nrpct_o
        mysum = mysum + pix(i,j,k)*rec1(i,j);
        end
        end
        prj(k) = mysum;
        else
        prj(k) = 0;
        end
        end
end
```

The code obtained is vectorized and ready to be executed in parallel in the multiple processing units available in the GPU, a process which is managed by ArrayFire with minimal user configuration. This notably improves the execution time of the function.

```
af::array projections_bolo(const std::int32_t &no_rays, const af::array &pix, const
                           af::array &rec1)
{
        af::array prj = af::constant(0, no_rays, f64);

        af::array kSlices = af::join(0, seq(24 - 1), seq(33 - 1, no_rays - 1));
        af::array ppixSlices = pix(span, span, kSlices);
        af::array rec1Slices = af::tile(rec1, 1, 1, kSlices.elements());
        af::array aux = ppixSlices * rec1Slices;
        aux = af::sum(af::sum(aux));
        prj(kSlices) = af::reorder(aux, 2, 0, 1);

        return prj;
}
```

(b) Just-In-Time execution

The Just-In-Time (JIT) execution [25] is a built-in compilation engine that performs a run-time analysis for the most computing demanding arithmetic functions. This process could be hidden from the user or it could be triggered by a specific function call that allows the data to be evaluated in order to be optimized by the JIT-supported functions. The most significant number of operations on arrays are grouped in the smallest possible number of kernels (kernel = minimum processing unit on a GPU). In this way, the execution times of the algorithm are significantly reduced, as the number of kernels is minimized to reduce the overhead that arises when small data sets are processed, when an element is reused multiple times (better cache performance), or the data kept in the GPU memory avoid unnecessary transfers.

(c) Data transfer

An attempt has been made to reduce the number of memory transfers between GPU and CPU to as few as possible to avoid memory transfers and, consequently, time consumption. However, this has not been possible for all code sections and functions. As aforementioned, ArrayFire does not currently provide an equivalent MATLAB griddata function supporting not-uniformly scattered data, so this solution uses the ALGLIB interpolation library. This library performs the operations on the CPU, so it is necessary to transfer the necessary matrices from the GPU memory to the CPU memory and then, after

performing the necessary calculations, transfer the processed data back to the GPU for later use. In addition, another part where data transfer from GPU to CPU is inevitable is in the last stage of the algorithm where the data are transferred to write the necessary output data.

3. Results

Once the C++ application is implemented, the code is optimized after different iterations, checking to ensure that the results of the different algorithm's steps are the same. The applied techniques to vectorize the code and use a heterogeneous platform with a CPU–GPU allow accelerating the original MATLAB by around 10 times (~×10). Table 2 shows the execution times for each step when 15 iterations are used as an input parameter.

Table 2. Execution times (in ms) obtained MATLAB vs. C++ application.

Function	MATLAB for 15 Iterations (ms)	C++ for 15 Iterations (ms)	Gain (ms)
Initialization for smoothing on the open magnetic curves	3183.0	125.6	3057.4
Load projections	3.0	5.0	−2.0
Load geometry	165.0	103.3	61.7
Backprojections	10.0	0.6	9.5
Uncertainty preparations	20.0	0.5	19.5
Projection lines approach loop	38,400.0	19.5	38,380.5
Smooth the closed and open surfaces	2400.0	54.5	2345.6
Smooth LCF inside and outside	4902.0	2703.5	2198.6
Evaluation of emissivity	27.0	3.3	23.7
Uncertainty estimation	5454.0	2953.1	2500.9
Evaluation of the reconstruction projection at iteration i^{th}	630.0	12.8	617.3
Final projections	1.2	1.2	0.0
Compute profile radiation and noise	7624.0	264	7360.0
TOTAL	62,819.2	6246.7	56,572.5

While most of the code and functions of the algorithm in MATLAB were translated into C++ and optimized, others could not be translated directly. The reason is that there is no information about the internal calculations of some of the functions in MATLAB. This implies that the results obtained in both implementations are slightly different. These differences are mainly evident in the implementation of the griddata function. For this application, it has been used with the "bicubic splines" interpolation method, incorporated in the FastDDM solver belonging to the ALGLIB library. Therefore, the differences between the results with the MATLAB version are minor, but not negligible.

Figures 8 and 9 show the results when comparing the tomographic reconstructions obtained in both versions of the algorithm and the absolute relative error of the projection reconstruction. In the first, it can be seen that the error between the algorithms for the projection reconstruction is less than 5%.

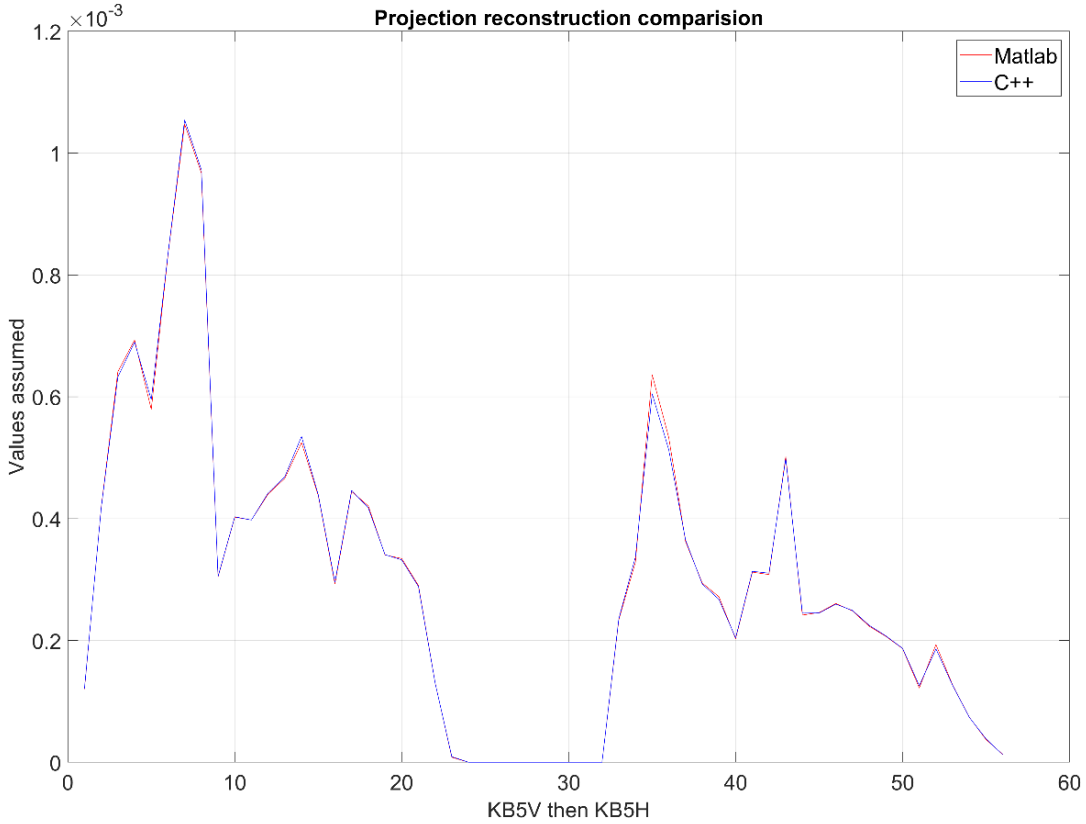

Figure 8. Comparison of the projection reconstruction for 15 iterations between the original MATLAB code and the C++ implementation.

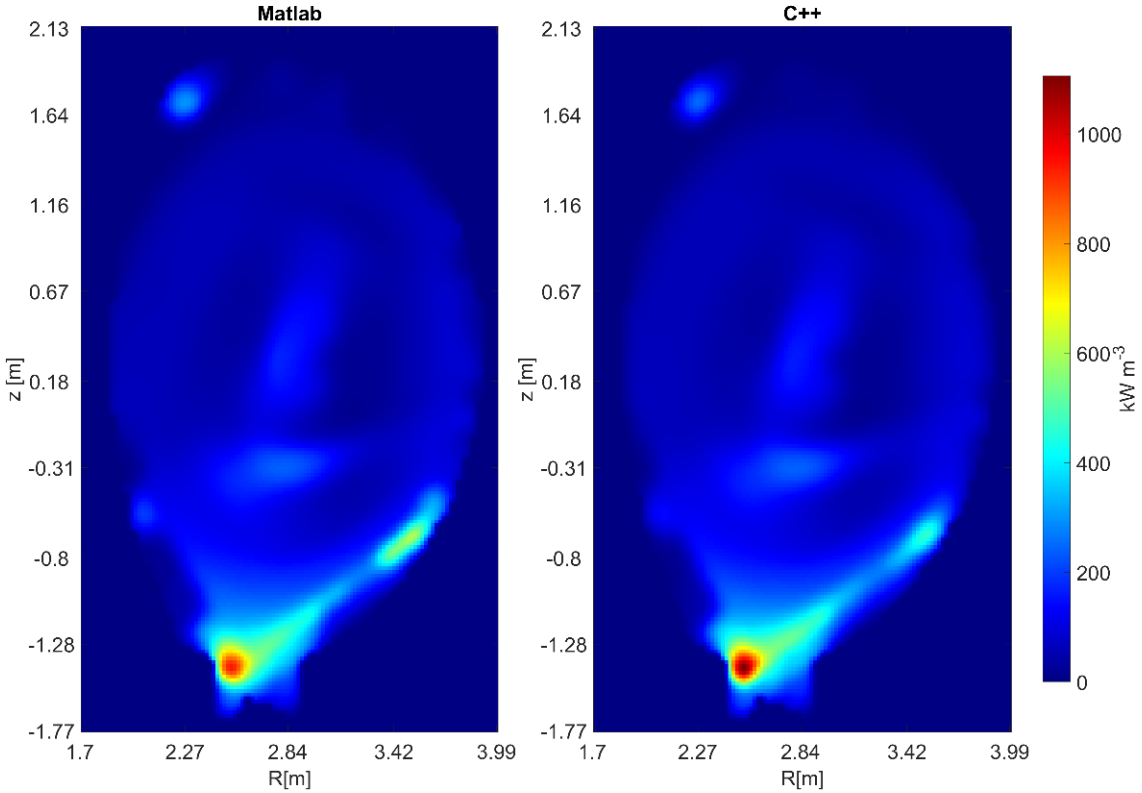

Figure 9. Comparison between the final reconstruction (15 iterations) obtained using the original MATLAB code and the C++ implementation, respectively.

4. Discussion

The initial goal was achieved by speeding up the algorithm's execution by a factor of ten (×10), porting a MATLAB application of ~6000 lines of code to an optimized standalone C++ application of ~4000 lines of code. As can be expected, the execution times obtained do not allow a real-time operation, but open the possibility of implementing other strategies or solutions that could achieve it. For example, the possibility of examining which parts of the code could be ported to other hardware platforms based on FPGA could also be analyzed to improve the execution time of specific parts. This is a good approach providing that software platforms are used that do not involve much effort in redoing all the code, as would be the case with OpenCL [26]. Although the desired speed had not been achieved, the speed-up obtained thanks to this first version is already approximately a factor of 10. While it is not yet fast enough for a real-time application of the ITER experiment, this work brings improved performance for the use of the code in inter-shot analysis on JET. Usually, discharges follow at 30 min intervals, the first ~5–7 min comprises the discharge itself and the data storage, the remaining interval of ~15 min for certain calculations, and ~5 min for decisions, based on these calculations. Considering then that the main bottlenecks slowing down the implementation have been identified, and that efforts are currently ongoing to improve the main algorithm implementing the ML methodology, it is expected to reach the objective of 25 ms per iteration in a future release.

All of the source code was generated with open-source libraries (no proprietary code was used), allowing better traceability, updating, building from scratch, and reuse by the scientific community. The testing platform is based on ITER CODAC Core System (RHEL) or Ubuntu 18.04. The former is quite interesting because the solution can be executed in an ITER Fast Controller platform. The selection of the ArrayFire library as the core of the application is an excellent choice because the MATLAB data structures and functions can be ported in a fairly direct way, requiring less effort than building the same application with other programming environments.

Additionally, ArrayFire simplifies enormously the use of the GPU. Another outstanding feature is the possibility of using a heterogeneous platform based on CPU and GPU, which allows balancing or choosing the most suitable for the nature of a given algorithm. Finally, it should be noted that the methodology followed can be applied to the development of applications into more complex frameworks used in large-scale science facilities, such as those corresponding to nuclear fusion.

Author Contributions: Methodology, validation and writing—review M.R., formal analysis and writing—review J.N., software, validation and writing—review V.C., conceptualization T.C., conceptualization E.P., investigation J.V., investigation A.M. All authors have read and agreed to the published version of the manuscript.

Funding: Grants PID2019-108377RB-C33 and PID2019-108377RB-C31 funded by MCIN/AEI/10.13039/501100011033; Comunidad de Madrid grant number PEJ-2019-AI/TIC-14507. Euratom Research and Training Programme (Grant Agreement No. 101052200—EUROfusion).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgments: JET Contributors: See the author list of "Overview of JET results for optimising ITER operation" by J. Mailloux et al. to be published in the Nuclear Fusion Special Issue: Overview and Summary Papers from the 28th Fusion Energy Conference (Nice, France, 10–15 May 2021). This work was carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No. 101052200—EUROfusion). Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wesson, J. *Tokamaks*, 4th ed.; Oxford University Press: Oxford, UK, 2011; ISBN 9780199592234.
2. Huber, A.; McCormick, K.; Andrew, P.; de Baar, M.R.; Beaumont, P.; Dalley, S.; Fink, J.; Fuchs, J.C.; Fullard, K.; Fundamenski, W.; et al. Improved radiation measurements on JET–First results from an upgraded bolometer system. *J. Nucl. Mater.* **2007**, *363–365*, 365–370. [CrossRef]
3. Huber, A.; McCormick, K.; Andrew, P.; Beaumont, P.; Dalley, S.; Fink, J.; Fuchs, J.C.; Fullard, K.; Fundamenski, W.; Ingesson, L.C.; et al. Upgraded bolometer system on JET for improved radiation measurements. *Fusion Eng. Des.* **2007**, *82*, 1327–1334. [CrossRef]
4. Ingesson, L.C.; Alper, W.B.; Chen, H.; Edwards, A.W.; Fehmers, G.C.; Fuchs, J.C.; Giannella, R.; Gill, R.D.; Lauro-Taroni, L.; Romanelli, M. Soft X ray tomography during ELMs and impurity injection in JET. *Nucl. Fusion* **1998**, *38*, 1675. [CrossRef]
5. Mlynar, J.; Craciunescu, T.; Ferreira, D.R.; Carvalho, P.; Ficker, O.; Grover, O.; Imrisek, M.; Svoboda, J. Current Research into Applications of Tomography for Fusion Diagnostics. *J. Fusion Energy* **2019**, *38*, 458–466. [CrossRef]
6. Craciunescu, T.; Peluso, E.; Murari, A.; Gelfusa, M.; JET Contributors. Maximum likelihood bolometric tomography for the determination of the uncertainties in the radiation emission on JET TOKAMAK. *Rev. Sci. Instrum.* **2018**, *89*, 053504. [CrossRef] [PubMed]
7. Peluso, E.; Craciunescu, T.; Murari, A.; Carvalho, P.; Gelfusa, M. A comprehensive study of the uncertainties in bolometric tomography on JET using the maximum likelihood method. *Rev. Sci. Instrum.* **2019**, *90*, 123502. [CrossRef] [PubMed]
8. Peluso, E.; Craciunescu, T.; Gelfusa, M.; Murari, A.; Carvalho, P.J.; Gaudio, P. On the effects of missing chords and systematic errors on a new tomographic method for JET bolometry. *Fusion Eng. Des.* **2019**, *146*, 2124–2129. [CrossRef]
9. Murari, A.; Peluso, E.; Craciunescu, T.; Lowry, C.; Aleiferis, S.; Carvalho, P.; Gelfusa, M. Investigating the thermal stability of highly radiative discharges on JET with a new tomographic method. *Nucl. Fusion* **2020**, *60*, 46030. [CrossRef]
10. Matos, F.A.; Ferreiraa, D.R.; Carvalho, P.J. Deep learning for plasma tomography using the bolometer system at JET. *Fusion Eng. Des.* **2017**, *114*, 18–25. [CrossRef]
11. Ferreira, D.R.; Carvalhob, P.J.; Carvalho, I.S.; Stuart, C.; Lomas, P.J. Monitoring the plasma radiation profile with real-time bolometer tomography at JET. *Fusion Eng. Des.* **2021**, *164*, 112179. [CrossRef]
12. Fuchs, J.; Neu, R.; Dux, R.; Fuchs, J.C.; Junker, W.; Kallenbach, A.; Mertens, V.; de PefiarHempel, S.; Schonmann, K.; ASDEX Upgrade Team. Influence of the Carbon and the Oxygen Concentration on the Density Limit in ASDEX Upgrade ECA. In Proceedings of the 21st EPS Conference Contribution, Montpellier, France, 27 June–1 July 1994; Volume 18B.
13. Carr, M.; Meakins, A.; Bernert, M.; David, P.; Giroud, C.; Harrison, J.; Henderson, S.; Lipschultz, B.; Reimold, F. Description of complex viewing geometries of fusion tomography diagnostics by ray-tracing. *Rev. Sci. Instrum.* **2018**, *89*, 083506. [CrossRef] [PubMed]
14. David, P.; Bernert, M.; Pütterich, T.; Fuchs, C.; Glöggler, S.; Eich, T. Optimization of the computation of total and local radiated power at ASDEX Upgrade. *Nucl. Fusion* **2021**, *61*, 066025. [CrossRef]
15. Konoshima, S.; Leonard, A.W.; Ishijima, T.; Shimizu, K.; Kamata, I.; Meyer, W.H.; Sakurai, S.; Kubo, H.; Hosogane, N.; Tamai, H. Tomographic reconstruction of bolometry for JT-60U diverted tokamak characterization. *Plasma Phys. Control. Fusion* **2001**, *43*, 959. [CrossRef]
16. Blas, J.G.; Dolz, M.F.; Garcia, J.D.; Carretero, J.; Daducci, A.; Aleman, Y.; Canales-Rodriguez, E.J. Porting Matlab Applications to High-Performance C++ Codes: CPU/GPU-Accelerated Spherical Deconvolution of Diffusion MRI Data. *Lect. Notes Comput. Sci.* **2016**, *10048*, 630–643.
17. Castro, R.; Romero, J.A.; Vega, J.; Nieto, J.; Ruiz, M.; Sanz, D.; Barrera, E.; de Arcas, G. Soft real-time EPICS extensions for fast control: A case study applied to a TCV equilibrium algorithm. *Fusion Eng. Des.* **2014**, *89*, 638–643. [CrossRef]
18. Esquembri, S.; Nieto, J.; Carpeño, A.; Ruiz, M.; Astrain, M.; Costa, V.; de Garcia, A. Application of Heterogeneous Computing Techniques for the Development of an Image-Based Hot Spot Detection System Using MTCA. *IEEE Trans. Nucl. Sci.* **2021**, *68*, 2151–2158. [CrossRef]
19. Esquembri, S.; Nieto, J.; Ruiz, M.; de Gracia, A.; de Arcas, G. Methodology for the implementation of real-time image processing systems using FPGAs and GPUs and their integration in EPICS using Nominal Device. *Fusion Eng. Des.* **2018**, *130*, 26–31. [CrossRef]
20. ArrayFire. Available online: https://arrayfire.com/ (accessed on 1 May 2022).
21. ALGLIB. Available online: https://www.alglib.net/ (accessed on 1 May 2022).
22. MATIO. Available online: https://github.com/tbeu/matio (accessed on 1 May 2022).
23. Aguilar-Rivera, A. A GPU fully vectorized approach to accelerate performance of NSGA-2 based on stochastic non-domination sorting and grid-crowding. *Appl. Soft Comput.* **2020**, *88*, 106047. [CrossRef]
24. HajiRassouliha, A.; Taberner, A.J.; Nash, M.P.; Nielsen, P.M.F. Suitability of recent hardware accelerators (DSPs, FPGAs, and GPUs) for computer vision and image processing algorithms. *Signal Processing Image Commun.* **2018**, *68*, 101–119. [CrossRef]

25. Performance of ArrayFire JIT Code Generation. Available online: https://arrayfire.com/blog/performance-of-arrayfire-jit-code-generation/ (accessed on 1 May 2022).
26. Astrain, M.; Ruiz, M.; Carpeño, A.; Esquembri, S.; Barrera, E.; Vega, J. A methodology to standardize the development of FPGA-based high-performance DAQ and processing systems using OpenCL. *Fusion Eng. Des.* **2020**, *155*, 111561. [CrossRef]