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, Julian Nieto, Victor Costa, Sergio Esquembri (Universidad Politécnica de Madrid)
Teddy Craciunescu (National Institute for Laser, Plasma and Radiation Physics)
Emmanuele Peluso (University of Rome Tor Vergata)
Jesús Vega (CIEMAT)
Andrea Murari (Consorzio RFX Padova) and JET contributors*

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 and 2019-2020 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

*See the author list of "Overview of JET results for optimising ITER operation" by J. Mailloux et al. to be published in Nuclear Fusion Special issue: Overview and Summary Papers from the 28th Fusion Energy Conference (Nice, France, 10-15 May 2021)

Outline

- Scope of the problem to be solved
- Hardware and software platform used
- Methodology used for the development
- Results
- Conclusions

Problem to be solved

- Acceleration of the maximum likelihood offline algorithm implemented for JET bolometry diagnostic to measure:
  - Plasma radiation emission
  - Internal and local emissivity profiles
  - Estimate the uncertainties
  - New dataset every 25ms
- Algorithm implemented in MATLAB:
  - Around 2500 lines of code using some MATLAB toolboxes
    - Use of fitting and interpolation functions (fit, griddata)
    - Use of median and gaussian filters
- Execution time in desktop computer: 25s

- T. Craciunescu et al., Rev. Sci. Instrum. 89, 053504 (2018)
- E. Peluso et al., Rev. Sci. Instrum. 90, 123502 (2019)

Algorithm: steps

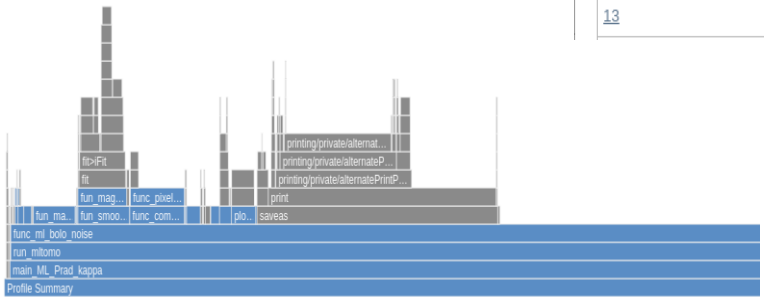
- Input data: mat and csv files from JET bolometer
- Data preparation:
  - Initialization for smoothing on the open magnetic curves (360x100, 133x453, 48x453)
  - Load noise projection (1x56, 56x1)
  - Load geometry (98x59x56, 98x59)
  - Backprojections
- Iterations:
  - ijk loop (98x59)
  - Smooth the closed and open surfaces (360x100)
  - Smooth LCF inside and outside (98x59)
  - Evaluation of emissivity (98x59)
  - Noise estimation (98x59)
  - Evaluation of the reconstruction projection at iteration i-th (56x1)
  - Uncertainty estimation (scalar)
- Final data:
  - Final projections (98x59)
  - Compute profile radiation and noise (scalars)

Profiling Kappa code with MATLAB

- Identify most time consuming functions:

Profile Summary (Total time: 92.632 s)

▾ Flame Graph



Generated 24-Mar-2021 15:43:36 using performance time.

Function Name	Calls	Total Time (s)	Self Time* (s)	Total Time Plot (dark band = self time)
main_ML_Prad_kappa	1	91.886	0.084	
run_mitomo	1	91.704	0.066	
func_ml_bolo_noise	1	91.621	32.346	

▾ Lines that take the most time

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
13	contur102 = fun_magnetic_parfor_bolo_unclosed_mini(contur1…	15	5.914	94.6%	

fun_smooth_diverto3 (Calls: 15, Time: 6.251 s)

▾ Flame Graph

Function Name	Function Type	Calls
fun_smooth_diverto3	Function	15

▾ Lines that take the most time

Line Number	Code	Calls	Total Time (s)	% Time	Time Plot
14	Z = fit(aaa',vect', 'smoothingspline', 'SmoothingParam', s…	720	5.681	96.1%	

fun_magnetic_parfor_bolo_unclosed_mini (Calls: 15, Time: 5.911 s)

Alternatives to improve the performance of MATLAB code

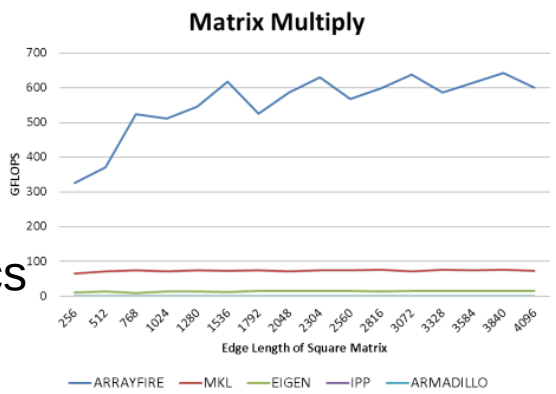
1. Use of Matlab Executable Files (MEX)
2. Use of parallel computing toolbox
3. Use of optimized libraries from MATLAB
4. Using GPUs in MATLAB
   - Not all functions support GPU acceleration. Using GPU arrays in MATLAB is not efficient if the arrays are small
5. Rewrite the code in C++ using optimized libraries for heterogeneous platforms:
   - CPU: arrayFire, alglib, libigl, armadillo, Intel MKL, IPP
   - GPU: arrayFire, CUDA (NVCC compiler, libraries cuBLAS, etc)
   - FPGA: Use of IntelFPGA/XILINX acceleration techniques

Hardware and Software solution

- Hardware:
  - CPU: Intel i7-9700k 3.6GHz, 128GBytes RAM)
  - GPU: NVIDIA (RTX2080 super)
  - FPGA: IntelFPGA Stratix 10
- Software:
  - Operating System: Ubuntu 18.04 or ITER CODAC Core System 6.3
  - Development of a C++ application develop using gnu tools (g++ 7.5)
- Software libraries:
  - arrayfire (https://arrayfire.com/). Library oriented to accelerate algorithms using CPU, GPU and other acceleration devices (OpenCl devices). Last version 3.8 (open source)
  - alglib (https://www.alglib.net/): Free edition
    - 2d interpolation of scattered data
  - libigl (https://libigl.github.io/) : Open Source
    - Point in polygon
  - Matio (https://github.com/tbeu/matio)
    - Mat files I/O

ArrayFire: General-purpose GPU library

- Built around a flexible data structure named "array"
- Lightweight wrapper around the data on the compute device (CPU multicore, GPU)
- Manages the data and basic metadata such as size, type and dimensions
- Support for multiple languages and OS: C/C++, Fortran, Java and R (Linux, Windows, Mac OS X)
- JIT Compiler, Combine multiple operations into one “kernel”
- GFOR - data parallel loop
- Allows concurrent execution over multiple data sets (for example images)
- Hundreds of parallel functions for multi-disciplinary work
  - Image processing
  - Machine learning
  - Graphics
  - Sets
- OpenGL based graphics




```
#include <arrayfire.h>
#include <af/utils.h>
void af_example()
{
        float f[8] = {1, 2, 4, 8, 16, 32, 64, 128};
        array a(2, 4, f); // 2 rows x 4 col array
        initialized with f values
        array sumSecondCol = sum(a(span, 1)); //
        reduce-sum over the second column
        print(sumSecondCol); // 12
}
```

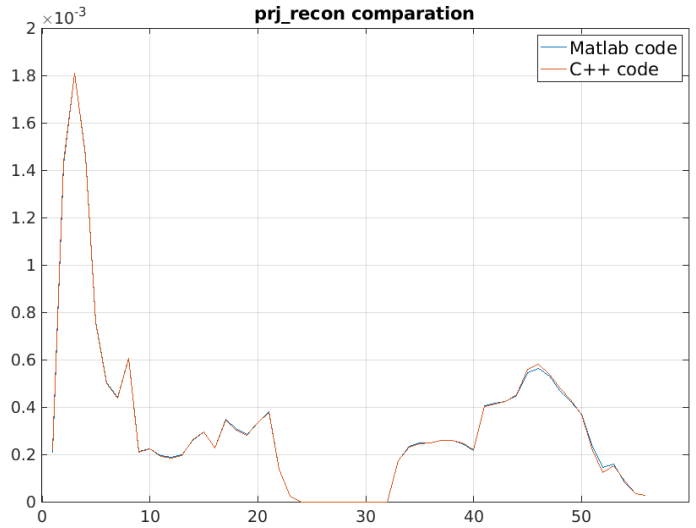
Optimization techniques used with arrayFire

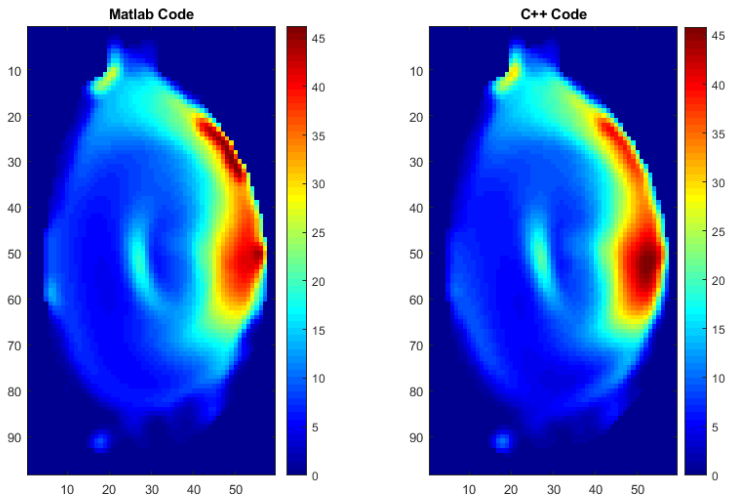
- Pre-allocation of main arrays
- Maximize the use of arrayFire vectors/matrixes (vectorization)
- Maximise use of arrayFire functions to avoid memory transfer to host for another operations
- Use of parallel “for loop” to launch parallel iterations in the GPU (Not always possible)
  - Avoid conditional statements inside loops
  - “parallel for loop” cannot be nested
  - Avoid as much as possible dependencies with previous loop (Use of auxiliary arrays)

Results: Execution time comparative

	Step	Matlab [ms]	C++ [ms]	Improvement [ms]
Preparation	Initialization for smoothing on the open magnetic curves	3065.0	111.0	2954.0
	Load projection noise	3.0	0.1	2.9
	Load geometry	303.0	10.6	292.4
	Backprojections	50.0	1.0	49.0
	Total Prep Time	3421.0	122.7	3298.3
Iterations	ijk loop	2050.0	230.0	1820.0
	Smooth the closed and open surfaces	360.0	3.0	357.0
	Smooth LCF inside and outside	350.0	180.0	170.0
	Evaluation of emissivity	0.3	0.1	0.2
	Noise estimation	1050.0	250.0	800.0
	Evaluation of a reconstruction projection at iteration i-th	17.0	0.1	16.9
	Uncertainty estimation	0.3	0.5	-0.2
	Total Iteration Time	3828	664	3164
Final Data	Final projections	0.1	0.1	0.0
	Compute profile radiation and noise	7000.0	35.0	6965.0

Results: Data accuracy





- Differences:
  - Implementation of “griddata”. This is a black-box in MATLAB
- Pulse number: 92398 Time: 7.5s

Conclusions and Future Work

- arrayFire simplifies enormously the implementation in C++
  - Very simple use of GPU
- Execution time gain ->12x (real-time target is 25ms)
- Use of Open Source Libraries + CUDA to support GPU
- Tested in Ubuntu 18.04 and ITER CODAC Core System
- Future work:
  - Improvement of specific parts using an FPGA-based accelerator
    - Bittware 520H with an Stratix 10. Hardware implementation using OpenCL for specific functions
  - Development of an ITER RTF application for “evaluation purposes”