The 2D Type-3 Non-Uniform FFT in CUDA

Amedeo Capozzoli, Claudio Curcio, Angelo Liseno, and Jonas Piccinotti

Università di Napoli Federico II
Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione
Via Claudio 21, I 80125 Napoli (Italy)
a.capozzoli@unina.it

Abstract — We present the parallel implementation on Graphics Processing Units (GPUs) of a type-3 Non-Uniform FFT (NUFFT) approach, namely, of a NUFFT for which data and results are located at irregular points. The performance of the algorithm is assessed against that of a parallel implementation of the same algorithm on multi-core CPUs using OpenMP directives.

Index Terms — CUDA, Non-Uniform FFT, OpenMP.

I. INTRODUCTION

In many areas of electromagnetics, the need arises of evaluating Non-Uniform Discrete Fourier Transforms (NUDFTs), namely DFTs with data and/or results on irregular grids. Imaging [1], solutions to differential and integral equations [2], fast array antenna analysis [3] and synthesis [4] and antenna diagnosis [5] are just few examples.

Unfortunately, the calculation of a NUDFT does not promptly benefit of the use of standard Fast Fourier Transforms (FFTs) ($O(NlogN)$ complexity) which on the contrary require Cartesian input and output grids. This solicited the development of Non-Uniform FFT (NUFFT) algorithms capable to perform accurate computations essentially with the same $O(NlogN)$ complexity. NUFFTs achieve such a complexity by exploiting fast and accurate pre- and/or post-interpolation stages, properly tailored to the problem at hand, from/to regular to/from irregular grids.

Apart from fast approaches, efficiency and effectiveness in the calculation of a NUDFT can be pursued also by adopting high performance, massively parallel computing (HPC) platforms as Graphics Processing Units (GPUs). The use of HPC is of course not disjoined from the numerical aspect since the appropriate exploitation of parallel hardware requires the choice of conveniently parallelizable algorithms.

The purpose of this paper is to present and discuss the parallel implementation on GPUs of a type-3 NUFFT approach (henceforth, NUFFT-3), namely, of a NUFFT for which data and results are located at irregular points.

NUFFT-3 finds important applications from the electromagnetic point of view. Indeed, it has been applied in [6] to effectively compute the aggregation and disaggregation stages of the Fast Multipole Method. Furthermore, it is of interest in aperiodic antenna analysis and synthesis when the far-field pattern is required into a non-uniform grid of the spectral plane [7].

NUFFT-3 has been originally dealt with using Gaussian interpolation windows [1, 8] or as a combination of type-1 and type-2 transforms [9, 10]. Most recently, we have improved [6] the choice of the Gaussian window parameters over that detailed [1, 8]. Despite type-1 and type-2 NUFFTs have been extensively researched also from the point of view of GPU approaches, it should be also noticed that only standard sequential CPU implementations for the NUFFT-3 have appeared throughout the literature, with neither parallel CPU nor GPU cases ever dealt with. Accordingly, in this paper, a NUFFT-3 GPU implementation is described for the first time.

Our approach is based on the recent scheme in [6]. Its timing performance is assessed against that of a parallel implementation of the same algorithm on multi-core CPUs, while its accuracy performance is pointed out thanks to a case of electromagnetic interest.

II. TYPE-3 NUFFT

Let $\{(x_i, y_i)\}_{i=0}^{N-1}$ be a set of N 2D non equispaced points, $\{f\}_{i=0}^{N-1}$ a set of corresponding coefficients and $\{(s_k, t_k)\}_{k=0}^{K-1}$ a set of K 2D non-equispaced spectral points. The transformation:

$$F_k = \sum_{i=0}^{N-1} f_i \, e^{-jx_i s_k} e^{-jy_i t_k}, \qquad (1)$$

is referred to as a 2D NUDFT-3 [1].

The problem of computing the F_k's amounts to the fact that Eq. (2) is not in the form of a standard Discrete Fourier Transform (DFT) since spatial and spectral points are irregularly located. Fortunately, reformulating the problem by interpolating non-uniformly sampled exponentials by uniformly sampled ones is in order. This can be achieved by the Poisson formula [11]:

$$e^{-j\xi x} = \sqrt{2\pi} \frac{\sum_{m\epsilon\mathbb{Z}} \mathcal{F}[\Phi(\xi) e^{-j\xi x}; m] e^{jm\xi}}{\sum_{m\epsilon\mathbb{Z}} \Phi(\xi + 2m\pi) e^{-j2m\pi x}}, \qquad (2)$$

where Φ is an appropriate interpolation window and

Submitted On: June 4, 2017
Accepted On: March 26, 2018

$\mathcal{F}$ denotes Fourier transformation. Accordingly, a computational scheme analogous to a Non-Uniform FFT (NUFFT) procedure of Type-3 [1, 6] can be set up. We briefly illustrate such a procedure by assuming the window functions Φ to be Gaussian [1, 6].

A. Step #1

The contributions from non-uniformly spaced input sampling points corresponding to $\mathbf{exp}[-\boldsymbol{j}(\boldsymbol{s_k x_i} + \boldsymbol{t_k t_i})]$ are "spread" by Gaussian windows $\mathbf{exp}[-\frac{x^2}{4\tau_x} - \frac{y^2}{(4\tau_y)}]$ with parameters $\boldsymbol{\tau_x}$ and $\boldsymbol{\tau_y}$, to a regular grid $(\boldsymbol{n\Delta x}, \boldsymbol{m\Delta y})$. Step #1 thus produces [6]:

$$f_\tau^{-\sigma}(n\Delta x, m\Delta y) = \frac{e^{[\sigma_x(n\Delta x)^2+\sigma_y(n\Delta y)^2]}}{\sqrt{4\sigma_x\sigma_y}} \sum_{i=0}^{N-1} f_i e^{-\left[\frac{(n\Delta x - x_i)^2}{4\tau_x}+\frac{(m\Delta y - y_i)^2}{4\tau_y}\right]}, \quad (3)$$

where the presence of the exponential function $\mathbf{exp}[\boldsymbol{\sigma_x x^2} + \boldsymbol{\sigma_y y^2}]$ is related to the pre-compensation of the Gaussian window used in Step #3. Due to the rapid decay of the exponential functions, $\boldsymbol{f_i}$ significantly contributes to only few samples of $\boldsymbol{f_\tau^{-\sigma}(n\Delta x, m\Delta y)}$. On defining $\boldsymbol{Int}[\boldsymbol{\alpha}]$ as the nearest integer to $\boldsymbol{\alpha}$, by letting $\boldsymbol{\xi_i} = \boldsymbol{Int}[\frac{x_i}{\Delta x}]$ and $\boldsymbol{\eta_i} = \boldsymbol{Int}[\frac{y_i}{\Delta y}]$, $\boldsymbol{i = 0, \dots, (N-1)}$, denote the nearest regular grid points to $\frac{x_i}{\Delta x}$ and $\frac{y_i}{\Delta y}$, respectively, and assigning $\boldsymbol{n' = n - \xi_i}$ and $\boldsymbol{m' = m - \eta_i}$, the contributions of each $\boldsymbol{f_i}$ to $\boldsymbol{f_\tau^{-\sigma}(n\Delta x, m\Delta y)}$ can be ignored when $|\boldsymbol{n'}| > \boldsymbol{m_{sp}}$ or $|\boldsymbol{m'}| > \boldsymbol{m_{sp}}$, where $\boldsymbol{m_{sp}}$ is a parameter properly selected according to the required accuracy. In other words, the summation in (3) provides a non-negligible contribution to only $(\boldsymbol{2m_{sp} + 1}) \times (\boldsymbol{2m_{sp} + 1})$ terms.

B. Step #2

The "spread" contributions are transformed to the spatial frequency domain via a standard FFT. In other words, the second step produces

$$F_\tau^{-\sigma}(p\Delta s, q\Delta t) \cong \frac{\Delta x \Delta y}{4\pi} \sum_{n=-\frac{M_{rx}}{2}}^{\frac{M_{rx}}{2}} \sum_{m=-\frac{M_{ry}}{2}}^{\frac{M_{ry}}{2}} f_\tau^{-\sigma}(n\Delta x, m\Delta y)\, e^{-jpn\Delta x\Delta s} e^{-jqm\Delta y\Delta t}. \quad (4)$$

C. Step #3

The "transformed" data are interpolated from the FFT output uniform grid to the non-uniform grid $\{(\boldsymbol{s_k}, \boldsymbol{t_k})\}_{\boldsymbol{k=0}}^{\boldsymbol{K-1}}$, again by Gaussian windows, $\mathbf{exp}[-\frac{s^2}{4\sigma_x} - \frac{t^2}{(4\sigma_y)}]$. The final output is thus:

$$F_k = \frac{\Delta s \Delta t}{4\pi\sqrt{\tau_x\tau_y}} e^{\tau_x s_k^2} e^{\tau_y t_k^2} \sum_{n=-\frac{M_{rx}}{2}}^{\frac{M_{rx}}{2}} \sum_{m=-\frac{M_{ry}}{2}}^{\frac{M_{ry}}{2}} F_\tau^{-\sigma}(n\Delta s, m\Delta t)\, e^{-\frac{(n\Delta s - s_k)^2}{4\sigma_x}} e^{-\frac{(m\Delta t - t_k)^2}{4y}}. \quad (5)$$

Similarly to Step #1, the presence of the Gaussian functions $\mathbf{exp}[\boldsymbol{\tau_x s^2} + \boldsymbol{\tau_y t^2}]$ is related to the post-compensation of the Gaussian windows used in Step #1. Again due to the rapid decay of the involved exponential functions, $\boldsymbol{F_\tau^{-\sigma}(n\Delta s, m\Delta t)}$ significantly contributes to only few samples of $\boldsymbol{F_k}$. In particular, on letting $\boldsymbol{\tilde{\xi}_k} = \boldsymbol{Int}[\frac{s_k}{\Delta s}]$ and $\boldsymbol{\tilde{\eta}_k} = \boldsymbol{Int}[\frac{t_k}{\Delta t}]$, $\boldsymbol{k = 0, \dots, K-1}$, and $\boldsymbol{p' = p - \tilde{\xi}_k}$ and $\boldsymbol{q' = q - \tilde{\eta}_k}$, the contributions of $\boldsymbol{F_\tau^{-\sigma}(n\Delta s, m\Delta t)}$ can be ignored when $|\boldsymbol{p'}| > \boldsymbol{m_{sp}}$ and $|\boldsymbol{q'}| > \boldsymbol{m_{sp}}$. In other words, the summation in (5) can be truncated to $(\boldsymbol{2m_{sp} + 1}) \times (\boldsymbol{2m_{sp} + 1})$ terms.

D. "Centering" and choice of the relevant parameters

Before applying the above procedure, a "centering" of the input and output sampling points is required, see [6]. Similarly, for the choices of $\boldsymbol{\Delta x}$, $\boldsymbol{\Delta y}$, $\boldsymbol{\tau_x}$, $\boldsymbol{\tau_y}$, $\boldsymbol{\sigma_x}$, $\boldsymbol{\sigma_y}$ and $\boldsymbol{m_{sp}}$, see [6] and Table 1. In such a table, $\boldsymbol{R}$ is chosen strictly larger than 2, $\boldsymbol{X} = \boldsymbol{max}\{|\boldsymbol{x_i'}|\}_{\boldsymbol{n=0}}^{\boldsymbol{N-1}}$, $\boldsymbol{Y} = \boldsymbol{max}\{|\boldsymbol{y_i'}|\}_{\boldsymbol{n=0}}^{\boldsymbol{N-1}}$, $\boldsymbol{S} = \boldsymbol{max}\{|\boldsymbol{s_k'}|\}_{\boldsymbol{k=0}}^{\boldsymbol{K-1}}$, $\boldsymbol{T} = \boldsymbol{max}\{|\boldsymbol{t_k'}|\}_{\boldsymbol{k=0}}^{\boldsymbol{K-1}}$ following the "centering" step, $\boldsymbol{m_{sp}} = \boldsymbol{2\pi b}$, b is chosen according to successive approximations of the following equation:

$$\boldsymbol{b} = \frac{\boldsymbol{1}}{\boldsymbol{\gamma}} \boldsymbol{log}\left(\frac{\boldsymbol{4\alpha}}{\boldsymbol{e}}\boldsymbol{b} + \frac{\boldsymbol{9\alpha}}{\boldsymbol{e}}\right), \quad (6)$$

with

$$\boldsymbol{\alpha} = \boldsymbol{2} + \frac{\boldsymbol{1}}{\sqrt{\boldsymbol{2\pi}}}, \quad \boldsymbol{\gamma} = \boldsymbol{\pi^2}\left(\boldsymbol{1} - \frac{\boldsymbol{2}}{\boldsymbol{R^2}}\right), \quad (7)$$

and e is the requested accuracy [6].

Table 1: Summary of the parameters choice

$\Delta x = \frac{\pi}{RS}$	$\Delta y = \frac{\pi}{RT}$
$\Delta s = \frac{2\pi}{\Delta x M_{rx}}$	$\Delta t = \frac{2\pi}{\Delta y M_{ry}}$
$M_{rx} \geq 2\left(\frac{XSR^2}{\pi} + 2\pi Rb\right)$	$M_{ry} \geq 2\left(\frac{YTR^2}{\pi} + 2\pi Rb\right)$
$\tau_x = b\Delta x^2$	$\tau_y = b\Delta y^2$
$\sigma_x = b\Delta s^2$	$\sigma_y = b\Delta t^2$

III. IMPLEMENTATIONS

The illustrated NUFFT-3 algorithm has been implemented in both GPU and CPU multithreaded codes. The latter has been developed in C++ parallelized by OpenMP directives. Such a choice matches with the use of the CUDA environment to develop the GPU counterpart. Both the codes are structured according to the above Steps and have been highly optimized. To perform a fair comparison, the CPU implementation has benefitted of most of the optimizations applied to the CUDA code. In the following, some implementation details are presented.

A. GPU multithreaded implementation

Step #1. The computation of $f_\tau^{-\sigma}(n\Delta x, m\Delta y)$ is the

most critical step of the three and requires some care since different approaches could be envisaged. The difficulty is due to the need of "pseudo-randomly" accessing the elements of $f_\tau^{-\sigma}(n\Delta x, m\Delta y)$ when selecting the $(2m_{sp}+1)\times(2m_{sp}+1)$ indices (n,m) to which each coefficient f_i contributes.

A first parallelization strategy would be to commit a thread to compute a single matrix element $f_\tau^{-\sigma}(n\Delta x, m\Delta y)$ using a 2D thread grid with each thread associated to a different (n,m) couple. However, in this way, the generic thread should perform, due to the "pseudo-random" filling, a time-consuming browsing of the input elements to establish whether they contribute to the committed element of $f_\tau^{-\sigma}(n\Delta x, m\Delta y)$ or not.

As an alternative, our code employs a 1D thread grid with each thread associated to a different input index i. In this, way, the browsing is avoided since each thread is assigned to a different f_i and updates the $(2m_{sp}+1)\times(2m_{sp}+1)$ corresponding elements of $f_\tau^{-\sigma}(n\Delta x, m\Delta y)$. However, notice that, by this solution, more than one thread may need to simultaneously update (namely, read, compute and store a new value) the same $f_\tau^{-\sigma}(n\Delta x, m\Delta y)$. When this happens, a "race condition" occurs. To preserve data integrity, atomic operations have been exploited ensuring the semantic correctness of the algorithm. Although serializing the updating operations, they have become very fast in the recent CUDA architectures.

Step #2. This step is implemented using cuFFT and a customized CUDA kernel executing the FFT shift operation.

Step #3. Parallelizing the calculation of Eq. (5) is easier than that of eq. (3), as it does not suffer from race condition hazards. The implemented code employs a 1D thread grid where each thread is associated to a different output index k.

B. CPU multithreaded implementation

Step #1 has been implemented in an analogous to what done for the CUDA case. More in detail, the parallelization has been applied according to the input index i. Accordingly, the `#pragma omp atomic` has been used to prevent race conditions.

Concerning Step #2, the FFT step required by Eq. (4) has been achieved by the multithreaded version of the FFTW routine contained in the Intel Math Kernel Library (MKL).

Finally, Step #3 has been implemented analogously to that done for the CUDA case, by applying the parallelization strategy to the output index k.

IV. NUMERICAL RESULTS

The performance of GPU and CPU implementations has been assessed with random spatial and spectral location vectors (x_i, y_i) and (s_k, t_k) and random complex coefficients f_i. Two cases have been considered: the case when $N = K,\ K = 2^p,\ p = 8,\ldots,20$ and the case when $N = K^2,\ K = 2^p,\ p = 8,9,10$. The former case is of interest for scattering by impenetrable objects, i.e., when only the scatterer's surface must be discretized, where discretization is essentially 1D and $N = K$. The latter case, instead, is of interest for the scattering by penetrable objects [12], i.e., when the scatterer's interior must be discretized, where discretization is essentially 2D and $N = K^2$. The computational speeds have been measured by averaging a number of 10 realizations for each individual test. The codes have been run on an Intel Core i7-6700K, 4GHz, 4 cores (8 logical processors), equipped with an NVIDIA GTX 960 card, compute capability 5.2.

Figure 1 (upper panel) displays, for the case $N = K$, the partial timings of the three mentioned calculation steps for the CUDA implementation. As it can be seen, the most computationally demanding operations are the spatial and spectral interpolations, namely, Step #1 and Step #3. Despite employing atomic operations, the spatial interpolation step is only slightly more demanding than the spectral implementation. This is due to two reasons. First, the implementation of Step #1 has been highly optimized. Second, atomic operations are extremely fast for the Maxwell architecture.

Table 2: Partial timings (in [ms]) for the CUDA implementation and for the case $N = K^2$

K	Step #1	Step #2	Step #3
256	38	0.8	0.9
512	150	0.8	0.9
1024	600	1.0	0.9

The partial timings of the three steps for the case $N = K^2$ and for the CUDA implementation are reported in Table 2. Due to the larger number of input points as compared to the output ones, now Step #1 is the most time consuming part of the computation.

Figure 1 (lower panel) displays the speedup obtained by the parallel GPU implementation, against the OpenMP one, for the two cases of $N = K$ and $N = K^2$. The GPU timings do not comprise CPU-GPU memory movements, as the use of the NUFFT-3 CUDA code is understood to be exploited within a fully GPU-based computation. As it can be seen, speedups of up to 8 are obtained for the case $N = K$. Larger speedups are achieved for the case $N = K^2$ since, in this case, Step #1 is the most time consuming one and more significantly benefits of the GPU acceleration. Notably, according to Amdahl's law [13], the amount of achievable speedup depends on the portion of the code that can be parallelized. A speedup of 5/6 can be already considered a satisfactory result.

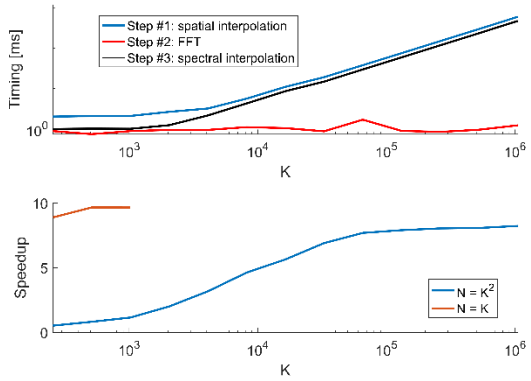

Fig. 1. Upper panel: Partial timings of the CUDA implementation: case $N = K$. Lower panel: Speedup of the CUDA implementation against the OpenMP one. Red line: case $N = K$. Blue line: case $N = K^2$.

We now show a test case of electromagnetic interest. As already mentioned, aggregation and disaggregation in the FMM [14, 15] can be effectively performed by a NUFFT-3 [6]. We consider a 2D Electric Field Integral Equation (EFIE) applied to the scattering of a perfectly conducting circular cylinder of radius a=2.5λ under TM (Transverse Magnetic) plane wave illumination. The cylinder's surface has been discretized in 1536 segments, grouped in 32 clusters [6]. More in detail, we compare the cases when aggregation and disaggregation are evaluated in an exact way and by a NUFFT-3. The good accuracy of the NUFFT-based version is witnessed by the very low relative root mean square error between the two compared cases and equal to $8.78 \cdot 10^{-11}$.

V. CONCLUSIONS AND FUTURE DEVELOPMENTS

We have discussed the parallel implementation on GPUs of a NUFFT-3. State-of-art implementation of NUFFT-3 are only sequential CPU ones. Here, the performance of the GPU approach has been compared to that of a purposely developed parallel CPU one using OpenMP directives. The provided parallelizations amount at a proper organization of the computations, but they do not alter the accuracy of the parallelized NUFFT-3 algorithm.

We now plan to extend the approach to the use of more efficient interpolating window functions.

REFERENCES

[1] J.-Y. Lee and L. Greengard, "The type 3 non-uniform FFT and its applications," *J. Comput. Phys.*, vol. 206, no. 1, pp. 1-5, June 2005.

[2] C. Liu, *et al.*, "Focusing translational variant bistatic forward-looking SAR data based on two-dimensional non-uniform FFT," *Progr. Electromagn. Res. M*, vol. 37, pp. 1-10, 2014.

[3] A. Capozzoli, *et al.*, "Fast CPU/GPU pattern evaluation of irregular arrays," *ACES J.*, vol. 25, no. 4, pp. 355-372, Apr. 2010.

[4] A. Capozzoli, C. Curcio, A. Liseno, and G. Toso, "Fast, phase-only synthesis of aperiodic reflect-arrays using NUFFTs and CUDA," *Progr. Electromagn. Res.*, vol. 156, pp. 83-103, 2016.

[5] A. Capozzoli, C. Curcio, and A. Liseno, "NUFFT-accelerated plane-polar (also phaseless) near-field/far-field transformation," *Progr. Electromagn. Res. M*, vol. 27, pp. 59-73, 2012.

[6] A. Capozzoli, C. Curcio, A. Liseno, and A. Riccardi, "Parameter selection and accuracy in type-3 non-uniform FFTs based on Gaussian gridding," *Progr. Electromagn. Res.*, vol. 142, pp. 743-770, 2013.

[7] A. Capozzoli, C. Curcio, A. Liseno, and G. Toso, "Fast, phase-only synthesis of aperiodic reflect-arrays using NUFFTs and CUDA," *Progr. Electromagn. Res.*, vol. 156, pp. 83-103, 2016.

[8] A. Dutt and V. Rokhlin, "Fast fourier transforms for nonequispaced data," *SIAM J. Sci. Comp.*, vol. 14, no. 6, pp. 1368-1393, 1993.

[9] F. Knoll, *et al.*, "Reconstruction of undersampled radial PatLoc imaging using total generalized variation," *Magn. Reson. Med.*, vol. 70, no. 1, pp. 40-52, July 2013.

[10] S. Tao, *et al.*, "NonCartesian MR image reconstruction with integrated gradient nonlinearity correction," *Med. Phys.*, vol. 42, no. 12, pp. 7190-7201, 2015.

[11] R. M. Trigub and E. S. Belinsky, *Fourier Analysis and Approximation of Functions*. Springer Science+ Business Media, Dordrecht, NL, 2004.

[12] A. Capozzoli, *et al.*, "Efficient computing of far-field radiation in two dimension," *IEEE Antennas Wireless Prop. Lett.*, vol. 16, pp. 2034-2037, 2017.

[13] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *Proc. of the AFIPS '67 Conf.*, Atlantic City, NJ, pp. 483-485, Apr. 18-20, 1967.

[14] M. Vikram and B. Shanker, "An incomplete review of fast multipole methods–from static to wideband–as applied to problems in computational electromagnetics," *ACES J.*, vol. 24, no. 2, pp. 79-108, Apr. 2009.

[15] C. Craeye, *et al.*, "Efficient numerical analysis of arrays of identical elements with complex shapes," *ACES J.*, vol. 24, no. 2, pp. 224-232, Apr. 2009.