Implementation of a GPU Accelerated Total Focusing Reconstruction Method within Civa Software

Gilles Rougeron[a], Jason Lambert[a], Ekaterina Iakovleva[a], Lionel Lacassagne[b] and Nicolas Dominguez [a]

[a] CEA, LIST, F-91191 Gif-sur-Yvette, France
[b] LRI, Université Paris-Sud,F-91405 Orsay cedex, France.

Abstract. This paper presents results of a TFM implementation for Full Matrix Capture acquisitions in CIVA, proposed as a post-processing tool for accurate analysis. This implementation has been made on GPU architecture with OpenCL to minimize the processing time and offer computational device flexibility (GPU/CPU). Examples on immersion configurations on isotropic 2D CAD specimen with planar extrusion are proposed to illustrate the performances. Reconstructions on 2D or 3D areas of direct echoes with mode conversion are allowed. Probe scanning can also be taken into account. Reconstruction results and a benchmark explaining the speedup are presented. Further improvements are also reviewed.

Keywords: Total Focusing Method, FMC, GPU, OpenCL.

INTRODUCTION

Ultrasonic Testing has been greatly transformed and modernized thanks to the introduction of phased array capabilities in the last decades. NDT performances have benefited from this technology mainly because of two factors: i) the increase in testing speed because of large aperture arrays combined with electronic scanning, ii) the possibility to replace or limit complex mechanical scanning by electronic scanning. Now the nature of phased arrays offers much wider possibilities than just sweeping and steering ultrasonic beams, and the step further is to go towards reconstructed data in the specimen framework. First examples of such reconstructed views are the so-called “true scan” or “corrected scan” views which represent the ultrasonic information along ultrasonic paths in the specimen. Other approaches aim at processing the data so that to produce reconstructed images of the defects in the part, like for instance the Total Focusing Method (TFM) [1] which applies a Synthetic Focusing [2] to phased array data. Such methods offer enhanced spatial resolution and signal to noise ratio but still suffer from increased computation times which limit the industrial deployment. Previous work in order to accelerate computations of TFM with GPUs were limited in the case of [4] to probes in contact with the specimen, and for [5] to planar surfaces. This paper presents recent progress made on the acceleration of the TFM processing in CIVA, by implementation on massively parallel hardware such as GPU (Graphical Processing Units) or multi-core CPU for probes in immersion and complex surfaces. In the remainder of this paper, we will recall the basic principle of TFM algorithm. Then we will provide details on parallelized CPU and GPU implementations of an optimized version of the algorithm. We will then show performance benchmark results. Finally after briefly reviewing the industrialization of the algorithm within CIVA software [5] by use of OpenCL emerging standard [6], we will draw perspectives and conclude.

TFM METHOD

Basic Principle

The TFM is a technique used to post-process the data from Full Matrix Capture (FMC) or Sparse Matric Capture (SMC) to produce a scalar image, I(P), of the inspected region, where the array is focused in transmission and reception at every point P in the image. The intensity of the TFM image I(P) is given by:

$$I(P)=\sum_{i,j=1}^{N_t} S_{ij}(T_{ip}+T_{jp}). \tag{1}$$

where Sij is the acquired temporal signal, Tip and Tjp are the time delays relative to the point P for i'th and j'th elements, respectively.

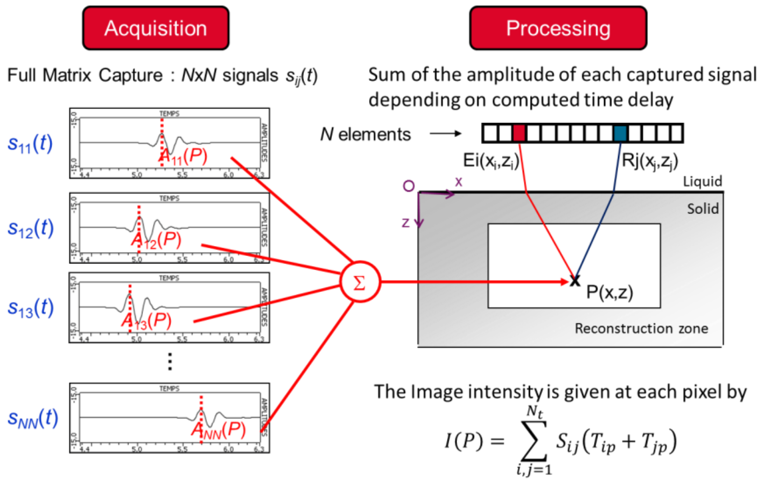
Acquisition
Processing
Full Matrix Capture : NxN signals $s_{ij}(t)$
Sum of the amplitude of each captured signal depending on computed time delay
$s_{11}(t)$
$A_{11}(P)$
$s_{12}(t)$
$A_{12}(P)$
$s_{13}(t)$
$A_{13}(P)$
$s_{NN}(t)$
$A_{NN}(P)$
N elements
Σ
Ei(x_i,z_i)
Rj(x_j,z_j)
O
x
z
Liquid
Solid
P(x,z)
Reconstruction zone
The Image intensity is given at each pixel by
$I(P)=\sum_{i,j=1}^{N_t} S_{ij}(T_{ip}+T_{jp})$

FIGURE 1. TFM basic principle.

In immersion mode, the transducer is separated from the mechanical part by a coupling liquid medium, which is generally water. For this inspection mode, propagation of ultrasonic waves in two media with different sound velocities results in refraction at the interface. As is shown in Figure 2., for a planar surface, ultrasound beam is refracted at the point I(x'$_i$; 0) at the liquid/isotropic solid interface given by z = 0. The propagation time T_{Eip} can be expressed as follows:

$$T_{Eip}=\frac{\sqrt{(x_i-x_i^{'})^2+z_i^2}}{c_1}+\frac{\sqrt{(x_i^{'}-x)^2+z^2}}{c_2} \tag{2}$$

where c1 and c2 are the velocity of ultrasonic waves in liquid and solid, respectively, (x_i; z_i) is the position of the i'th element, (x'$_i$; 0) is the positions of the refraction point I and (x; z) are the coordinates of the point P.

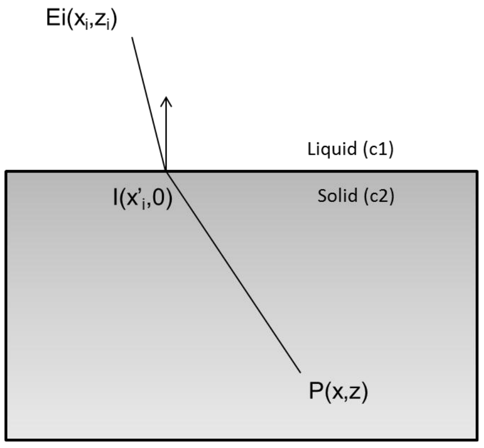

FIGURE 2. Evaluating Snell-Descartes law at a planar surface.

The x-coordinate of the refraction point I can be determined using Snell's law which leads, as detailed in (3), to the following nonlinear equation:

$$\frac{x_i - x_i^{'}}{c_1\sqrt{(x_i - x_i^{'})^2 + z_i^2}} = .\frac{x_i^{'} - x}{c_2\sqrt{(x_i^{'} - x)^2 + z^2}} \qquad (3)$$

The solution of equation 3 can be approximated by using some iterative methods in order to find the roots of the associated polynomial of fourth degree. In our case, the selected root finding method is Laguerre's because of its capacity to always converge for any initial guess and with a cubical rate of convergence for simple roots. For each root found, polynomial deflation is applied using Horner's Method to reduce the polynomial degree: in the case of a real root P(X) is divided by (X -xroot), whereas in the case of a complex root P(X) is divided by (X-xroot) (X – xroot_conjugate).

Curved surfaces imply solving higher degree polynomials depending on the surface: from 4th to 10th degree for cylindrical surface depending on the type of control, up to 16th degree for the case of a torical surface..

ALGORITHM IMPLEMENTATION

Optimizations

The following two optimizations are of algorithmic nature and are used both by CPU and GPU implementation.

Time-of-Flight Symmetry

For a given point P and a given element, the time delay between them depends only on their positions which do not change during the computation. Thus, this time delay is the same whether the element acts as a transmitter or as a receiver. That is why, it is possible to say that there is a symmetry regarding the role of each element. Then, the algorithm can be optimized by computing each time delay only once.

Time-of-Flight Interpolation

For homogeneous specimen with smooth front surface time of flights can be computed on a coarser grid and then interpolated for each pixel of the image. Step grid value is set to the wavelength of the type of wave used for inspection.

With previous two optimizations, the algorithm is organized as follows:

1. For each grid point of a coarse grid time of flight with each element is computed.
2. For each pixel and each couple of element in emission (i) and in reception (j)
   Time of flight t_{pixel_ij} is computed by interpolation of the grid point time of flight in the neighborhood of the pixel, $t_{pixel\_ij} = t_{interp\_i} + t_{interp\_j}$
   Pixel intensity is updated by accumulation of the signal S_{ij} amplitude at time t_{pixel_ij}.

CPU Implementation

Parallelization of the algorithm on CPU is straightforward. An OpenMP implementation has been made with:

1. Parallel computation of time of flights on coarser grid (one CPU thread per grid point).
2. Parallel interpolation of time of flights + amplitude summation (one CPU thread per pixel)

No synchronization is required as computation and interpolation of times of flight are independent for each grid point / each pixel. Furthermore amplitude summations for each pixel are made in separate threads.

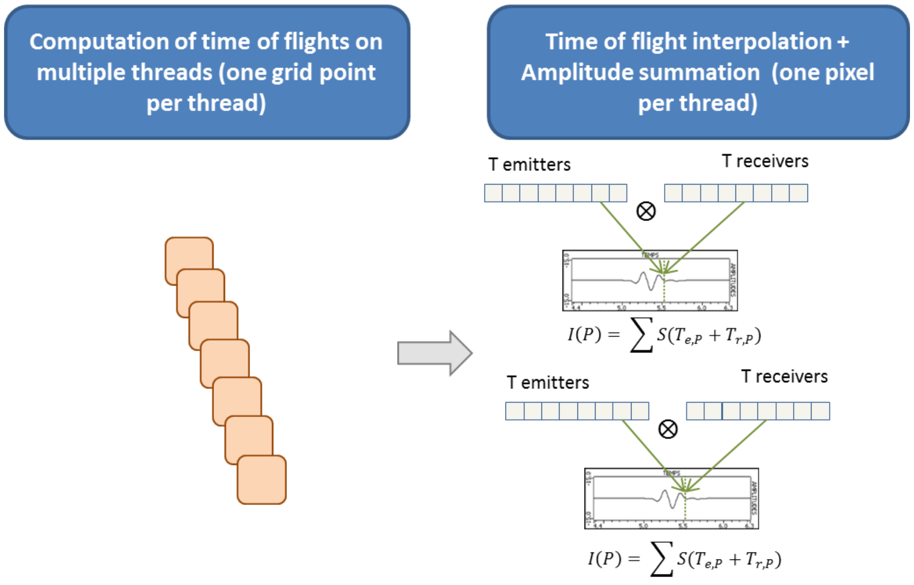
Computation of time of flights on multiple threads (one grid point per thread)
Time of flight interpolation + Amplitude summation (one pixel per thread)
T emitters
T receivers
$I(P) = \sum S(T_{e,P} + T_{r,P})$
T emitters
T receivers
$I(P) = \sum S(T_{e,P} + T_{r,P})$

FIGURE 3. CPU implementation.

GPU Implementation

The CUDA Toolkit version 4.2 was used to develop codes aimed at CUDA 2.x enabled device. With this programming architecture, computing task will be distributed among the GPU's streaming multiprocessors (SM). To fit to this architecture, the CUDA model subdivides a compute task on two levels to form a grid:

- A thread level with each disposing of its own program counter and of its own registers.
- It defines a block as a group of contiguous threads distributed to a single SM where its threads can share its resources (shared memory, registers...).

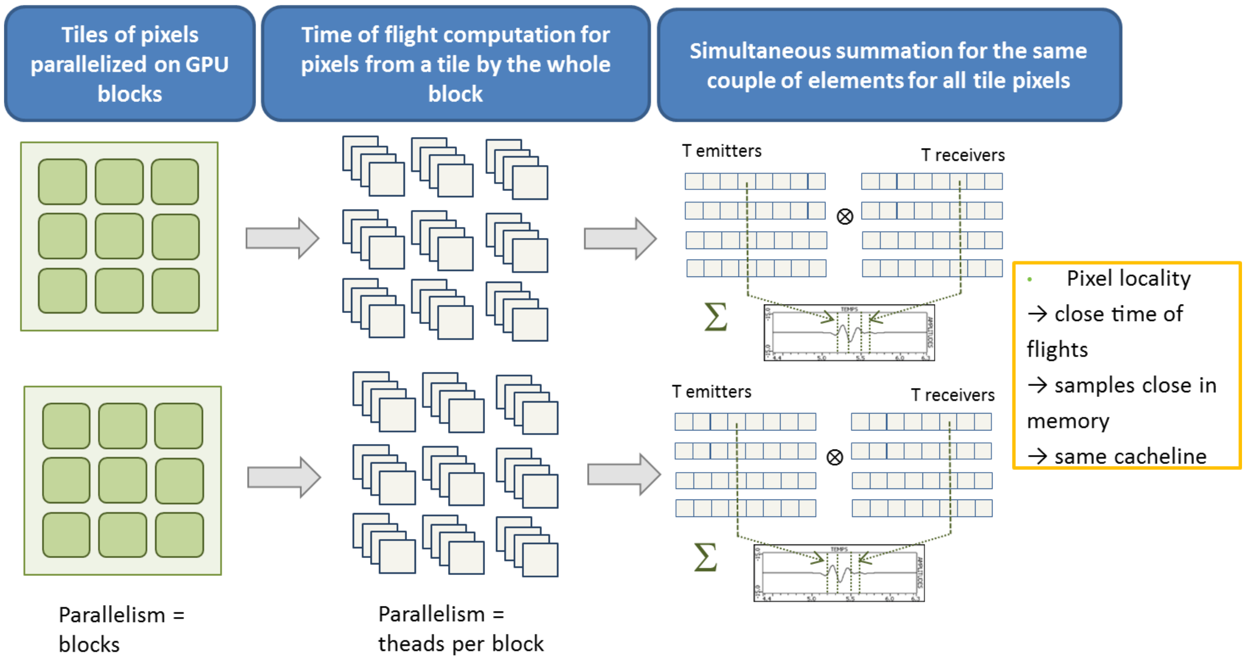
Tiles of pixels parallelized on GPU blocks
Time of flight computation for pixels from a tile by the whole block
Simultaneous summation for the same couple of elements for all tile pixels
T emitters
T receivers
Σ
T emitters
T receivers
Σ
Pixel locality
→ close time of flights
→ samples close in memory
→ same cacheline
Parallelism = blocks
Parallelism = theads per block

FIGURE 4. GPU implementation.

This architecture is most efficient with regular computations, where every thread of a block do the same computations and when threads access GPU's global memory sequentially.

The two steps of a TFM optimized algorithm (with symmetry and interpolation of times of flight) have been implemented as followed:

1. *Computation of time of flights:* a block of threads is assigned to a rectangular group of pixels corresponding to a coarser grid cell, and they simultaneously compute times of flight at each 4 corners of this cell. Results are stored in shared memory to accelerate data access. Threads are synchronized at the end of this step.
2. *Time of flight interpolation plus amplitude summation*: this step requires careful attention to improve signal data access. As they are stored in global memory, threads should make coalesced access. This way, they can access global memory through a single memory transaction. To reach this goal, threads of a block are divided and interlaced in N work-groups. N contiguous threads work simultaneously on N different pixels for a given couple of element, thus a given signal. As time of flight for neighbor pixels are very similar, access to same signal data are coalesced. Finally amplitude summation over a workgroup for each pixel requires a reduction.

OpenCL Implementation

GPU implementation described in previous paragraph, initially developed with Cuda, has been ported to OpenCL ([5]). The same code then could be executed both on CPU (through Intel and AMD OpenCL implementation) and on GPU (Nvidia and AMD HD Radeon GPU ones).

PERFORMANCE BENCHMARK

In this paragraph performance results are provided comparing the different implementations on different hardware.

Test Cases

Two test cases were used, one on a planar specimen, and the other on a cylindrical one (Figure 3.). Characteristics of both cases are presented below in Table 1.

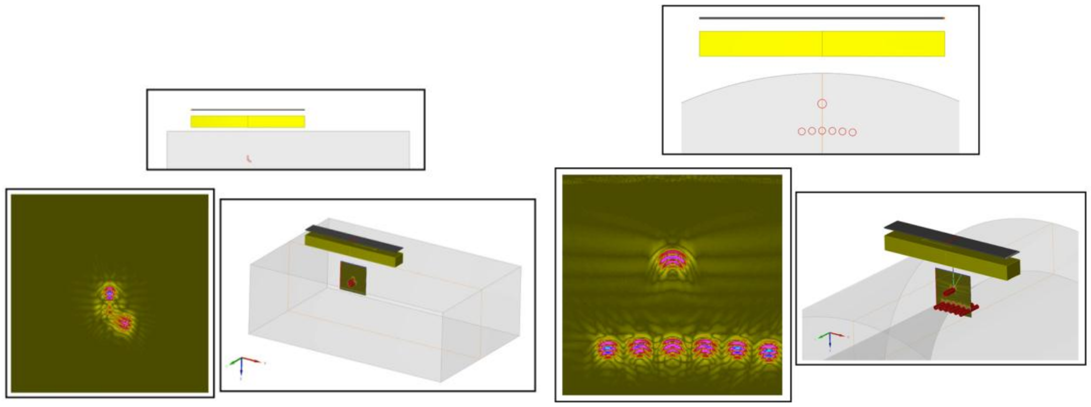

FIGURE 5. The two tests cases used for benchmark performance.

TABLE 1. Test cases characteristics.

	Planar Specimen	Cylindrical Specimen
Phased-array elements	128	128
Central frequency (MHz)	2	2
Number of samples per signal	1031	2007
FMC Data (MB)	130	250
Reconstruction zone (mm)	40x40	40x40
Image definition (pixels)	200x200	200x200

Hardware and Software

CPU used was: 2x Xeon X5690 (6 cores) @3,47GHz, while GPUs used were an Nvidia GTX580 (1.5 GB), Nvidia Tesla C2070 (6 GB), and an AMD Radeon HD6970 (2 GB).

On CPU, OpenMP implementation was made with Microsoft Visual C++ 2010 compiler. OpenCL implementation was executed through Intel SDK 1.5 and AMD APP SDK 2.7.

On GPU, Cuda implementation was made with Cuda SDK 4.2. OpenCL implementation was executed on Nvidia GPUs through Nvidia OpenCL 1.1. provided with Cuda SDK 4.2. For AMD Radeon HD6970, AMD APP SDK 2.7 was used.

Results

TABLE 2. Performances (en s)

	CPU				**GPU**				
	OpenMP		**OpenCL**		**Cuda**		**OpenCL**		
	Mono	**Multi**	**AMD**	**Intel**	**GTX580**	**C2070**	**GTX580**	**C2070**	**HD6970**
Planar	4.927	0.446	1.570	2.284	0.192	0.227	0.255	0.348	0.381
Cylindrical	5.431	0.473	1.654	2.276	0.197	0.235	0.266	0.368	0.384

On CPU, best performances are obtained with OpenMP implementation. It provides a good scaling. OpenCL implementation is 3 to 5 times slower compared with OpenMP. As this implementation is an adaptation of a Cuda code optimized for GPU architecture, it is not necessarily well suited for a CPU.

On GPU, best performances are obtained with Cuda implementation. OpenCL performances are nearly comparable to Cuda.

CIVA PERSPECTIVE

OpenCL implementation of TFM algorithm has been integrated in Civa software ([6]) developed by CEA. This language has been chosen as it is an emerging industrial standard. It offers portability among a vast variety of massively parallel architectures (multi-many cores CPU/GPU/FPGA etc…). It provides good to acceptable performances compared with native platforms languages/library : i.e. Cuda on Nvidia, OpenMP on CPU[1].

This implementation offers the following features:

- Specimen can be planar, cylindrical or 2D CAD with planar extension (Figure 4.)
- Signal data set with multiple probe positions can be taken into account (Figure 4.)
- Reconstructions can be done on 2D or 3D areas (Figure 5.)
- Echo with direct mode conversion can be reconstructed (Figure 6.)

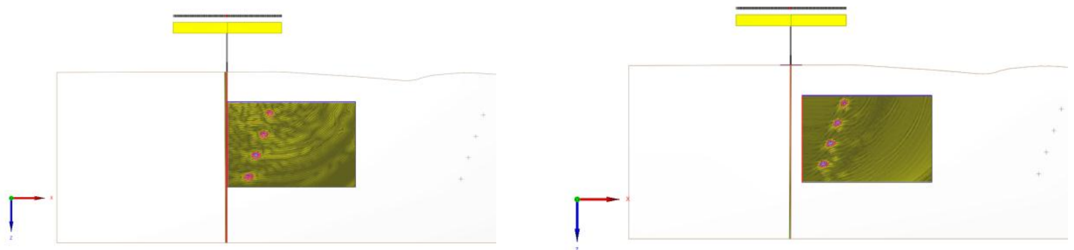

FIGURE 6. Reconstruction over a complex 2D CAD specimen. On the left with one position, on the right with a scanning of ten positions.

[1] OpenCL is improving on CPU (see latest Intel SDK 2013 with automatic parallelisation and code vectorization)

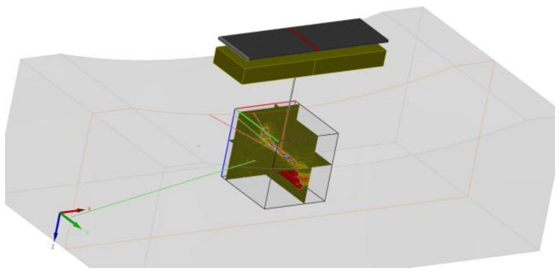

FIGURE 7. Reconstruction over a 3D area.

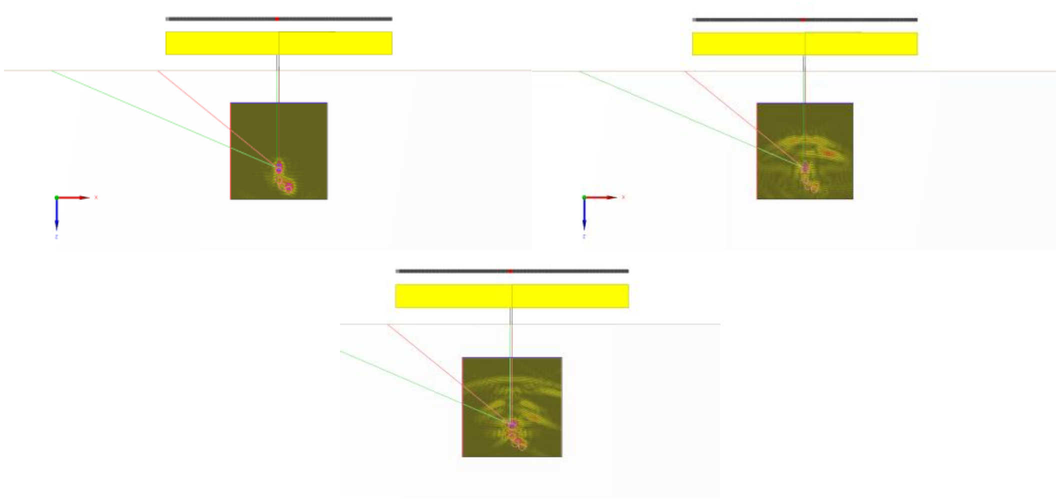

FIGURE 8. Reconstruction for different direct echo modes (LL top-left, LT=TL top-right, TT bottom).

FURTHER IMPROVEMENTS

New features shall be added to TFM OpenCL implementation in Civa in order to reach the same level of functionalities as native previous version:

- More specimen: 2D CAD with cylindrical extension shall be taken into account. As they can potentially contain conical or torical surfaces, polynomial of high degrees shall have to be solved.
- Reconstruction with skips (corner and indirect echoes) shall be available
- With current integration, signal data have to be transferred entirely on GPU memory before launching OpenCL code execution. An out-of-core solution when FMC data exceeds GPU memory shall be provided.

Performances shall be enhanced:

- A Newton method for simple surfaces (ex. Planar) could be used instead of Laguerre. It is should be more efficient for such simple geometries.

The goal is to reach a level of performance high enough in order to do TFM reconstruction in real-time (with a frame rate of at least 25 images per second.

CONCLUSION

Implementations of an optimized and fully parallelized Total Focusing Reconstruction method, both on CPU and GPU with native languages (OpenMP and Cuda) have been presented. Performance results on two test cases have been showed and analyzed. Industrialization of this fast reconstruction method within CIVA platform through an OpenCL port has been made. Current features of this integration have been presented. TFM on GPU should be available in a CIVA 11.X version.

ACKNOWLEDGMENTS

This work has been partly supported by the French Agence Nationale de la Recherche OPARUS project (ANR-10-COSI-010-01).

REFERENCES

1. Seydel J., “Ultrasonic synthetic-aperture focusing techniques in ndt,” Research Techniques in Nondestructive Testing, vol. 6, pp. 1–47, 1982.
2. C. Holmes, B. Drinkwater, P. Wilcox, “Post-processing of the full matrix of ultrasonic transmit receive array data for non-destructive evaluation”, NDT & E International, 38 (8), 2005, pp 701-711.
3. D. Romero-Laorden, O.Martnez-Graullera, C.J.Martn-Arguedas, M.Prez, and L.G.Ullate, “Paralelizaci´on de los procesos de conformaci´on de haz para la implementaci´on del total focusing method,” in CAEND Comunicaciones congresos, 12 Congreso Espaol de Ensayos No Destructivos, 2011.
4. M. Sutcliffe, M. Weston, B. Dutton, P. Charlton, K. Donne, “Real-time full matrix capture for ultrasonic non destructive testing with acceleration of post-processing through graphic hardware”, NDT & E International, Volume 51, October 2012, pp 16-23.
5. OpenCL, http://www.khronos.org/registry/cl/specs/opencl-1.2.pdf
6. “CIVA : State of the art simulation software for Non Destructive Testing,” http://www-civa.cea.fr/.