Journal of Applied Mathematics and Physics, 2015, 3, 1056-1061
Published Online August 2015 in SciRes. http://www.scirp.org/journal/jamp
http://dx.doi.org/10.4236/jamp.2015.38131

Cheetah: A Library for Parallel Ultrasound Beamforming in Multi-Core Systems

David Romero-Laorden[1], Carlos Julián Martín-Arguedas[2], Javier Villazón-Terrazas[1], Oscar Martinez-Graullera[1], Matilde Santos Peñas[3], César Gutierrez-Fernandez[2], Ana Jiménez Martín[2]

[1]ITEFI, Spanish National Research Council, Madrid, Spain
[2]Department of Electronics, University of Alcalá, Alcalá de Henares, Spain
[3]Department of Computer Architecture and Automation, Complutense University of Madrid, Madrid, Spain
Email: cj.martin@uah.es

Received 15 August 2015; accepted 19 August 2015; published 26 August 2015

Abstract

Developing new imaging methods needs to establish some proofs of concept before implementing them on real-time scenarios. Nowadays, the high computational power reached by multi-core CPUs and GPUs have driven the development of software-based beamformers. Taking this into account, a library for the fast generation of ultrasound images is presented. It is based on Synthetic Aperture Imaging Techniques (SAFT) and it is fast because of the use of parallel computing techniques. Any kind of transducers as well as SAFT techniques can be defined although it includes some pre-built SAFT methods like 2R-SAFT and TFM. Furthermore, 2D and 3D imaging (slice-based or full volume computation) is supported along with the ability to generate both rectangular and angular images. For interpolation, linear and polynomial schemes can be chosen. The versatility of the library is ensured by interfacing it to Matlab, Python and any programming language over different operating systems. On a standard PC equipped with a single NVIDIA Quadro 4000 (256 cores), the library is able to calculate 262,144 pixels in ≈105 ms using a linear transducer with 64 elements, and 2,097,152 voxels in ≈ 5 seconds using a matrix transducer with 121 elements when TFM is applied.

Keywords

GPGPU, Ultrasound Image, Beamforming, Array Transducer

1. Introduction

During recent years, computing industry has opened a way to parallel computing. First, dual-core processors (CPUs) were introduced in personal systems at the beginning of 2005, and it is currently common to find them in laptops as well as 8 and 16-core workstation computers, which means that parallel computing is not relegated to big supercomputers. On the other hand, Graphics Processor Units (GPUs), as their name suggests, came about as accelerators for graphics applications, predominantly those using the OpenGL and DirectX programming interfaces. Although originally they were pure fixed-function devices, the demand for real-time and 3D graphics made them evolve into small computational units, multithreaded processors with extremely high computational power and very high memory bandwidth that are now available to anyone with a standard PC or laptop.

Since 2006 GPUs can be programmed directly in C/C++ using CUDA or OpenCL [1], allowing each and every arithmetic logic unit on the chip to be used by programs intended for general-purpose computations (GPGPU). A CUDA program consists of one or more stages that are executed on either the host (CPU) and a NVIDIA GPU. The stages that exhibit little or no data parallelism are implemented in CPU code whereas those that exhibit rich amount of data parallelism are implemented in the GPU code. These parallel functions are called kernels, and typically generate a large number of threads to exploit data parallelism

From an architectural analysis viewpoint, beamforming techniques are pretty interesting in particular because they can be seen as a data parallel process, making possible its implementation on machines with diverse computational and I/O capabilities. Some previous works as [2] show toolboxes for beamforming computation over CPUs and multi-core CPUs obtaining very good timing results. Our research group has been working on GPUs applied for field modeling acceleration [3] [4] as well as for fast beamforming [5]-[7] achieving speed ups of 150× over conventional CPU beamforming. Nowadays, GPU beamformers are a reality and there are lot of research groups working on solutions for NDT and medical applications [8].

The aim of this work is to present CHEETAH, a fast ultrasonic imaging library to assist on fast development of new ultrasound beamforming strategies (currently, only SAFT methods are supported) making possible to generate 2D and 3D images on a standard PC or laptop in just few milliseconds. The input data can originate from either a simulation program or from an experimental setup. The library is composed by several routines written in CUDA for fast execution, thus a NVIDIA© GPU is required at the present time. Nowadays, the 1.0 version (Windows and Linux OS) will be soon available.

2. CHEETAH Core Features

CHEETAH has been designed as a free multi-platform library written in C++ and CUDA which can handle multitude of focusing methods, interpolation schemes and apodizations, to generate images from real RF signals obtained from any application and any acquisition process. The main features currently supported are:

- **Custom transducers.** Commands for defining linear and matrix arrays are given. Likewise, arbitrary geometry transducers such as sparse arrays can be also specified. Thereby different transducers can be used depending of the concrete application.
- **Custom SAFT techniques.** Commands for easily defining specific SAFT sequences of emission/reception are given. Anyway, the library comes with some predefined techniques, like 2R-SAFT [6], and TFM [9].
- **2D/3D imaging.** Commands for composing bidimensional and volumetric images (also C-Scan images) are given what makes possible to span a wide range of applications.
- **Coherence factor.** Commands for the application of coherence factors are given [10].
- **Matlab©/Python bindings.** As the library is written in C++ and CUDA, its functionality is available in Windows or Linux OS. Likewise, we have developed specific bindings to connect it to Matlab© what allows to reuse existing code or utilize specific toolbox functionalities.

3. TFM Design on Cheetah Library

In this section, TFM method has been chosen as the case of study to analyze the main implementation aspects of the beamforming algorithm on the library.

3.1. TFM Imaging Principles

Synthetic Aperture Focusing Techniques (SAFT) are based on the sequential activation of the array elements in emission and reception, and the separate acquisition of all the signals involved in the process. Then, a beamforming algorithm is applied to focus the image dynamically in emission and reception obtaining the maximum quality at each image point. One of the most common beamforming methods is Total Focusing Method (TFM) [9] [11] and it is based on Full Matrix Array (FMA), which is the complete data matrix $X(t)$ created by any transmitter-receiver combination:

$$\boldsymbol{X}(t) = \left\{ X_{tx,rx}(t) \mid \forall 1 \le (tx, rx) \le N \right\} \tag{1}$$

where $X_{tx,rx}(t)$ is the corresponding signal to *tx* transmitter and *rx* receiver, and *N* is the number of array elements. For ease the envelope computation, acquired signals are decomposed into their analytic signals form (in-phase *I* and quadrature components *Q*) applying the Hilbert Transform being now expressed as:

$$\mathbf{X}(t) = \mathbf{I}(t) + j\mathbf{Q}(t) \tag{2}$$

According to the Hilbert transformation applied in Equation (2), a complex data matrix has been created. Then, for the case of a 2D scenario 2 a Delay-and-Sum (DAS) beamforming process is used to calculate two images in the following way:

$$A_I(x,z) = \sum_{i=1}^{N}\sum_{k=1}^{N} I_{tx_i,rx_k}\left(D(x,z)\right) \tag{3}$$

$$A_Q(x,z) = \sum_{i=1}^{N}\sum_{k=1}^{N} Q_{tx_i,rx_k}\left(D(x,z)\right) \tag{4}$$

where $A_I(x, z)$ and $A_Q(x, z)$ are the in-phase and quadrature images respectively, and $D(x, z)$ is the delay corresponding to the focus point (x_{fp}, z_{fp}) in the space which is calculated as follows:

$$D(x,z) = \frac{\sqrt{(x_{fp}-x_{tx})^2 + z_{fp}^2} + \sqrt{(x_{fp}-x_{rx})^2 + z_{fp}^2}}{c} \tag{5}$$

being x_{tx} and x_{rx} the coordinates of the transducer elements *tx* and *rx*, respectively. Over this general scheme it is possible to introduce any type of apodizations.

$$A = \sqrt{{A_I}^2 + {A_Q}^2} \tag{6}$$

Finally, the envelope is calculated according to the equation6 to obtain the final image.

3.2. Parallel GPU Implementation

The background behind CHEETAH library comes from the knowledge that our group has on beamforming acceleration using GPUs [5]-[7]. Nevertheless, it has been considered to briefly review the main concepts involved in the parallel implementation of the TFM process on the library.

During the processing the input raw data and the output image pixels are stored as single-precision floating-point numbers (4 bytes). We have also used fast intrinsic math routines which provide better performance at the price of IEEE compliance (the precision is slightly reduced). In our case, this causes a minimal numerical difference who has little influence on the final output ultrasound image quality.

Designed parallel strategies perfectly fit the SIMD (Single Instruction Multiple Data) model of GPU architecture [1]. The general parallelization scheme can be resumed as **Figure 1** suggests:

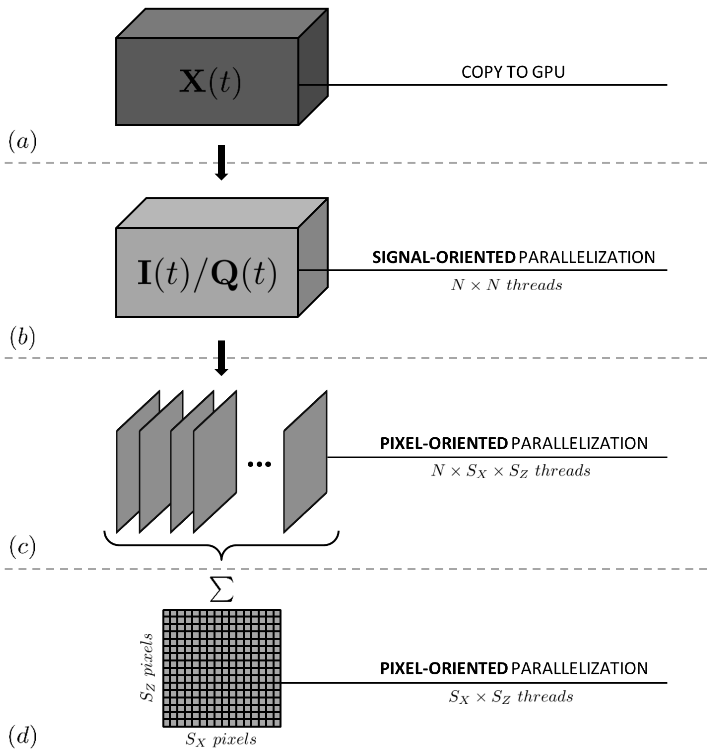
a) The process starts when the Full Matrix Array $X(t)$ defined in Equation (1) is transferred from CPU memory to GPU global memory via PCI Express bus.

b) According to Equation (2) the Hilbert Transform is applied to every signal using CUFFT libraries [1]. The parallelism strategy is signal-oriented which properly splits the algorithm and computes the FFT of the data. This means that *NxN* threads work concurrently. The result of these operations is a complex data matrix $I(t)$ and $Q(t)$ which is stored in GPU texture memory.

c) DAS *kernel* is applied to the complex data matrix to calculate *N* low resolution images (LRI_I and LRI_Q) each one corresponding to an emitter-all receivers [8] as Equations (3) and (4) suggest. The parallelization is performed launching as many threads as image pixels what, supposing image dimensions of $S_X \times S_Z$, means $N \times S_X \times S_Z$ threads. Several optimizations (reuse computed values, symmetries, shared/constant memories [7]) have been used to maximize performance. The focusing delay is calculated on-the-fly and it is indexed in the $I(t)$ and $Q(t)$ complex signals interpolating real and imaginary parts. Finally, the complex samples are multiplied by the corresponding apodization gains and added together to beamform each of the images.

d) A new kernel is defined to calculate the final A_I and A_Q images performing the sum of the (LRI_I and LRI_Q) images respectively. Once the final values for both images are computed, the envelope calculation is carried out. This strategy follows a pixel-oriented parallelism launching $S_X \times S_Z$ threads.

We must mention that, this scheme has been further optimized for those SAFT strategies based on the coarray



$\mathbf{X}(t)$
COPY TO GPU
(a)
$\mathbf{I}(t)/\mathbf{Q}(t)$
SIGNAL-ORIENTED PARALLELIZATION
$N \times N$ threads
(b)
PIXEL-ORIENTED PARALLELIZATION
$N \times S_X \times S_Z$ threads
(c)
$\sum$
S_Z pixels
PIXEL-ORIENTED PARALLELIZATION
$S_X \times S_Z$ threads
(d)
S_X pixels

Figure 1. Parallel scheme for beamforming acceleration.

model and their particular characteristics as well as for 2D and 3Dimaging, in order to squeeze the maximum performance and speed [6] [7].

4. Library Performance Evaluation

To evaluate the performance of the library we have chosen two experimental scenarios, a Multi-Tissue ultrasound phantom (040 GSE model by CIRS Inc. company) to evaluate the 2D-imaging; and a methacrylate piece with five drills for 3D imaging.TFM has been chosen as the beamforming method for both cases. Details are given in **Table 1**.

As a 2D imaging example, we use a linear 64 elements transducer which results in a FMA matrix of 4096 signals. The result of the processing is shown in **Figure 2(a)**.

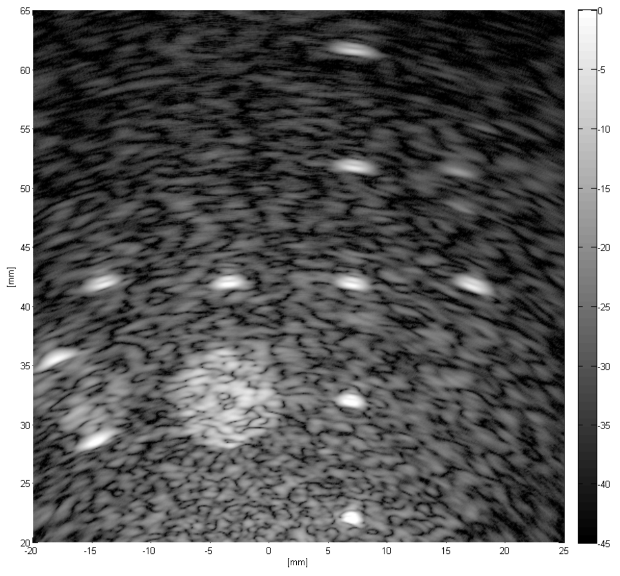
In **Table 2** and **Table 3** computing times can be observed for several CPUs (testing in mono-core and multi-core). Respect to GPUs, they are equipped with different number of cores.

Given that a standard PC cannot use a NVIDIA Tesla, we underline the results of NVIDIA Quadro 4000, which has 256 cores and 2 GB RAM and took around 105 ms for an image of 512 × 512 pixels, as well as, the results of the NVIDA QuadroK5000, which took less than 50 ms.

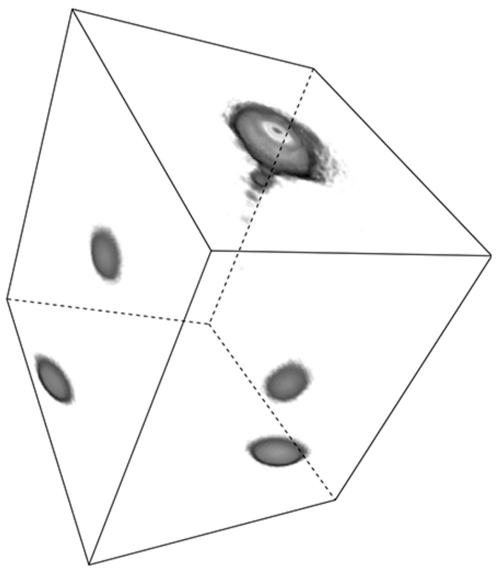
As a 3D imaging example we use a matrix array of 11 × 11 elements. 3D-TFM method has been applied, using an FMA of 14,641 signals and the same GPUs to generate a volume of 128 × 128 × 128 pixels which is shown in **Figure 2(b)**. The results of the imaging generation took less than 5 seconds in the NVIDA Quadro 4000, and less than 2 seconds in the NVIDA Quadro K5000, more details are described in **Table 4**.

5. Conclusions and Future Work

A fast and versatile library for the generation of ultrasound images has been created which exploits GPU technology to implement the beamformer via software. We have made a detailed description of the library features and it has been quantified the benefits of using the GPU as a processing tool. So by using a simple graphics card equipped with NVIDIA CUDA technology now is possible to accelerate the development of new imaging techniques.



(a)



(b)

Figure 2. (a) TFM image from Multi-Tissue ultrasound phantom; (b) Volumetric image from 5 drills using 3D-TFM method.

Table 1. Beamformation scenarios tested for 2D and 3D imaging.

Parameters	Image	
	2D Imaging	3D Imaging
Scenario	Multi-Tissue Phantom	Methacrylate piece
Medium velocity	1540 [m/s]	2690 [m\s]
Array size	64 elements	121 elements
Array pitch	0.28 [mm]	1.0 [mm] both directions
Imaging frequency	2.6 [MHz]	3.16 [MHz]
Sample frequency	40 [MHz]	40 [MHz]

Table 2. Performance of CPU-based TFM algorithm.

CPU processor	RAM	# Cores	TFM time [seg]
Intel Core 2 Quad Q9450	4 GB	1	24.13
		4	6.96
Intel Core i7 3632Q M	8 GB	1	6.91
		8	2.08
Intel Xeon E51650	30 GB	1	6.84
		4	2.23
		8	1.80
		12	1.64

Table 3. Performance of GPU-based TFM algorithm using CHEETAH.

NVIDIA GPU	RAM	# Cores	TFM time [mseg]
GeForce 540 M	1 GB	96	265.57
GeForce 9800 GTX+	512 MB	128	215.01
GeForce 635 M	1 GB	144	239.27
Quadro 4000	2 GB	256	105.07
Quadro K2000	2 GB	384	119.23
Quadro K5000	6 GB	1536	45.11

Table 4. Performance of GPU-based TFM-3D algorithm using CHEETAH.

NVIDIA GPU	RAM	# Cores	TFM time [mseg]
GeForce 540 M	1 GB	96	7972.70
GeForce 9800 GTX+	512 MB	128	7056.01
GeForce 635 M	1 GB	144	7587.25
Quadro 4000	2 GB	256	4874.28
Quadro K2000	2 GB	384	6697.05
Quadro K5000	6 GB	1536	1929.13

We are currently working on the implementation of more beamforming algorithms (e.g. adaptative beamforming and PA), on supporting CPU capabilities for parallel computing (OpenACC, OpenMP, MPI) and multi-GPU processing and on improving the overall performance. We are open to receive collaboration/feedback of any group that will be interested in using our library in their own research.

Acknowledgements

This work has been supported by the Spanish Government and the University of Alcalá under projects DPI2010-19376 and CCG2014/EXP-084, respectively.

References

[1] Hwu, W.-M.W. and Kirk, D.B. (2010) Programming Massively Parallel Processors: A Hands-On Approach.

[2] Hansen, J.M., *et al.* (2011) An Object-Oriented Multi-Threaded Software Beamformation Toolbox, SPIE Medical Imaging: Ultrasonic Imaging, Tomography, and Therapy. In: D'hooge, J. and Doyley, M.M., Eds., 79680Y.

[3] Romero-Laorden, D., *et al.* (2011) Field Modelling Acceleration on Ultrasonic Systems Using Graphic Hardware. *Computer Physics Communications*, **182**, 590-599. http://dx.doi.org/10.1016/j.cpc.2010.10.032

[4] Villazón-Terrazas, J., *et al.* (2012) A Fast Acoustic Field Simulator. In 43o Congreso Español de Acústica (TECNIACUSTICA), Evora, 1-9.

[5] Romero-Laorden, D., *et al.* (2009) Using GPUs for Beamforming Acceleration on SAFT Imaging. *IEEE International Ultrasonics Symposium*, Rome, 1334-1337.

[6] Martín-Arguedas, C.J., *et al.* (2012) An Ultrasonic Imaging System Based on a New SAFT Approach and a GPU Beamformer. *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, **59**, 1402-1412. http://dx.doi.org/10.1109/TUFFC.2012.2341

[7] Romero-Laorden, D., *et al.* (2013) Strategies for Hardware Reduction on the Design of Portable Ultrasound Imaging Systems. Advancements and Breakthroughs in Ultrasound Imaging, G. P. P. Gunarathne, Ed., 2013. http://dx.doi.org/10.5772/55910

[8] So, H.K.H., *et al.* (2011) Medical Ultrasound Imaging: To GPU or Not to GPU. *IEEE Micro*, **31**, 54-65. http://dx.doi.org/10.1109/MM.2011.65

[9] Holmes, C., *et al.* (2008) Advanced Postprocessing for Scanned Ultrasonic Arrays: Application to Defect Detection and Classification in Non-Destructive Evaluation. *Ultrasonics*, **48**, 636-642. http://dx.doi.org/10.1016/j.ultras.2008.07.019

[10] Martínez-Graullera, O., *et al.* (2011) A New Beamforming Process Based on the Phase Dispersion Analysis. *The International Congress on Ultrasonics*, Gdansk, 185-188.

[11] Hunter, A.J., *et al.* (2008) The Wavenumber Algorithm for Full-Matrix Imaging Using an Ultrasonic Array. *IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control*, **55**, 2450-2462. http://dx.doi.org/10.1109/TUFFC.952