NDT&E International 51 (2012) 16–23

Contents lists available at SciVerse ScienceDirect

NDT&E International

journal homepage: www.elsevier.com/locate/ndteint

Real-time full matrix capture for ultrasonic non-destructive testing with acceleration of post-processing through graphic hardware

Mark Sutcliffe [a,b,*], Miles Weston [b], Ben Dutton [b], Peter Charlton [a], Kelvin Donne [a]

[a] Swansea Metropolitan University, Mt Pleasant Campus, Swansea SA1 6ED, UK
[b] TWI NDT Validation Centre (Wales), ECM^2, Heol Cefn Gwrgan, Margam, Port Talbot, SA13 2EZ, UK

A R T I C L E I N F O

Article history:
Received 23 February 2012
Received in revised form
19 June 2012
Accepted 20 June 2012
Available online 4 July 2012

Keywords:
Ultrasonics
Graphic-processing
Arrays
Post-processing
CUDA
Full matrix capture

A B S T R A C T

Full matrix capture allows for the complete ultrasonic time domain signals for each transmit and receive element of a linear array probe to be retrieved. While it is more common to use full matrix capture to post-process data to allow for electronic steering, focusing and imaging after the initial inspection processes, due to the data-acquisition and performance limitations of the focusing algorithms real-time inspection systems are not yet common place.

This paper investigates several algorithm optimisation techniques utilising standard in-expensive PC architecture with parallelisation undertaken by the graphic processing unit. This approach is further combined with several other software engineering optimisation techniques including threaded data-capture, the use of look-up tables and half-matrix implementation to produce a real-world inspection scenario for benchmark and performance analysis.

Experimental results are presented indicating that high frame rates inclusive of data-acquisition and image render are achievable with 32 active transmit and receive elements. This approach is shown to offer significant performance advantages with low implementation and development costs.

1. Introduction

In recent years the use of array transducers in the field of ultrasonic non-destructive testing (NDT) has become common place [1]. They are now routinely used for lab and site based inspections across a range of fields which include oil and gas, power generation, aerospace and marine. The main attraction of arrays over conventional single crystal transducers is their ability to electronically focus, steer and sweep ultrasonic energy with an almost infinite number of combinations. There are several benefits which can result from this: electronic beam steering and sweeping minimises mechanical transducer movements. This improves the inspection coverage on components with limited surface access.

Full Matrix Capture (FMC) is an ultrasonic data acquisition process that utilises phased array probes to capture the complete time domain signals for each transmit and receive element [2]. The technique has evolved from the synthetic transmit aperture method commonly used in medical ultrasound [3], and uses a ‘transmit on one and receive on all’ data capture approach. Initially a single element in the array is used as a transmitter, while all elements then receive (see Fig. 1). This process repeats until all elements in the array have been fired. Therefore a transducer containing n elements will generate a matrix of n^2 A-scans, known as the full matrix of data. Since only energy from a single element is present in the test structure at any moment in time, the FMC data acquisition process is commonly referred to as a sequential transmission technique.

The sequential nature of FMC, and the subsequent focusing algorithm requires a more computationally intensive post-processing approach than traditional inspection techniques, as the full matrix of data is shown to be substantially larger. Given these limiting factors FMC systems tend to provide a much slower inspection technique, with focusing often done after the inspection process on data acquired at an earlier point in time.

While computing processing power has increased exponentially over many years high performance post-processing of large data is often best performed in parallel. Suitability for parallelisation must first be explored, with several parallelisation options currently available; including execution of code on a PC Cluster, targeting the system to a Field Programmable Gate Array (FPGA) or the execution of code over multiple threads on a standard PC. For real-time execution issues of latency (e.g. the speed at which data can be sent and received from the PC Cluster) is also a factor, as is complexity of code and threading concurrency.

This paper suggests several optimisation approaches to speed up the FMC inspection process with particular emphasis on data

* Corresponding author at: Swansea Metropolitan University, Mt Pleasant Campus, Swansea, SA1 6ED, UK.
E-mail address: dmasutcliffe@gmail.com (M. Sutcliffe).

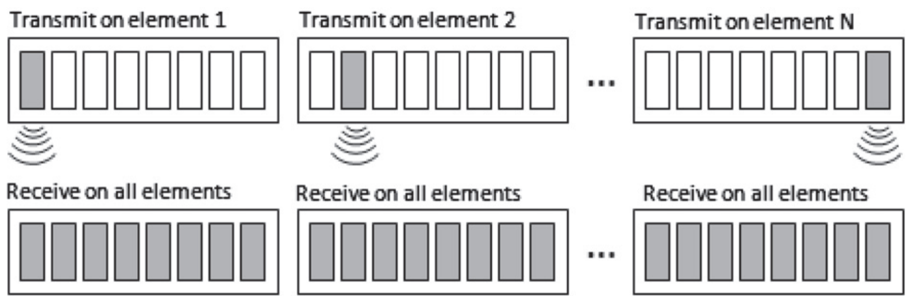

Fig. 1. Transmitting and receiving with FMC.

parallelisation over the Graphic Processing Unit (GPU) and provides experimental results based on real-world scenarios where FMC can be used as a real-time inspection process.

2. FMC focusing algorithm

TWI have developed a software implementation of the core focusing algorithm for use with FMC which is used to generate cross-sectional imagery from the full matrix of Data. A technique first introduced to the field of NDT by the University of Bristol in 2005 [2], the core of this algorithm is expressed mathematically in Eq. (1), and defines a grid of pixels representative of the cross-sectional area of inspection with pixel intensities (I) determined from the time of flight calculation for each tx, rx pair. A Hilbert transform (h) is used to convert the real time domain signal into complex form. This is useful as taking the modulus of the complex signal allows the signal magnitude (envelope) to be found

$$I(x,z) = \left| \sum h_{tx,rx} \left(\frac{\sqrt{(x_{tx}-x)^2+z^2} + \sqrt{(x_{rx}-x)^2+z^2)}}{cl} \right) \right| \tag{1}$$

In other words, for every focal x,z point there will be n^2 time of flight calculations. This time of flight information is then used to extract relevant amplitude information from the full matrix of data using a delay and sum beam forming approach. Since every pixel in the image acts as a focal point, fully focused imagery of the region-of-interest can be obtained.

The Hilbert transform in signal processing is a mathematical operator which acts on a real signal to return a complex signal. The benefit of this in the context of image reconstruction is it allows the signal magnitude to be determined as shown in Fig. 2. Here it can be seen that if only the real signal is used to render an image then indications from a single reflector may show up as multiple responses due to the nature of the ultrasonic signal (typically a five cycle Gaussian). This can hinder data interpretation and in the worst case scenario lead to false flaw characterisation. Complex signals can be represented mathematically by Eulers equation given in Eq. (2). It is straightforward to show that this equation can be used to derive the well known sine and cosine identities given in Eqs. (3) and (4). These identities show that sine and cosine waves consist of both negative and positive frequency components, and when positive and negative frequency components of a cosine signal are added together according to the identity in Eq. (2), the imaginary component cancels, leaving a purely real signal

$$e^{j2\pi ft} = \cos(2\pi ft) + j\,\sin(2\pi ft) \tag{2}$$

$$\sin(2\pi ft) = \frac{e^{j2\pi ft} - e^{-j2\pi ft}}{2j} \tag{3}$$

$$\cos(2\pi ft) = \frac{e^{j2\pi ft} - e^{-j2\pi ft}}{2} \tag{4}$$

It should be observed that performance of this algorithm is directly proportional to the size of the full-matrix of data with performance in the order of $O(n^4)$ excluding computational requirements for the Hilbert transform. This poor performance is attributed to the fact that for each x, z pixel location the corresponding tx, rx data has to be evaluated. As this algorithm performs the same mathematical operation on each x, z, tx, rx it is an ideal candidate for optimisation through parallelisation under the Single Instruction-Multiple Thread (SIMT) architecture. Originally developed for the vector processors found in the supercomputers of the 1970s and 1980s [4], the SIMT architecture is now commonly found in graphic processors. With the introduction of NVIDIA CUDA, allowing custom code to operate on and exploit its hardware, supercomputing capability is now available at a low cost [5].

3. CUDA

CUDA is a parallel programming model introduced in 2006 by NVIDA to allow complex computational problems to operate over its GPU architecture [6]. For years the computing gaming industry has exploited the power of the GPU for 3D computer animation and in response to this, the GPU has evolved to perform many math operations simultaneously through a highly parallelised and optimised architecture. Unlike CPU architecture the GPU is not a general purpose processor, and has limited capability to perform different tasks in parallel. Instead the GPU allows for the same operation to be applied to a predefined data-set. This relationship is explored in Fig. 3, where it is indicated that the CPU has evolved for data access, caching and flow control while the GPU architecture has evolved to be dedicated to doing one task well—the same problem executed in parallel.

This relationship is more clearly understood when examining a typical function of the GPU. In the case of 3D computer animation this data-set would be the vector information for objects within the virtual world, while an operation may involve a rotation of all objects along a specific axis. The implementation of this scenario is achieved by performing the rotation operation on all vectors within the data-set. Given that each vector is independent of all other vectors this operation is highly suitable for parallelisation, as no dependencies exist within the data-set. Conversely, the CPU is capable of more advanced levels of parallelisation, where one task maybe dedicated to the capture of data, while another dedicated to refreshing the display, and another for post-processing, and through thread management dependencies may be carefully controlled. However a limiting factor for parallelisation over the CPU is the number of cores available to the software application. While a typical CPU may have 4–8 cores, it is common for an entry level GPU to have in excess of 96 cores.